HEWLETT PACKARD

HP 9000 Computers

HP-UX Reference Volume 1

HP-UX Reference

Volume 1: Sections 1 and 9

HP 9000 Computers

HP-UX Release 9.0

HEWLETT
PACKARD

HP Part No. B2355-90033
Printed in USA August 1992

Third Edition
E0892

Legal Notices

Printing History

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. However, minor changes may be made at reprint without changing the printing date. The manual part number changes when extensive changes are made.

To ensure that you receive new editions of this manual when changes occur, you may subscribe to the appropriate product support service, available through your HP sales representative.

August 1992. Third Edition. This edition is an update to the Second Edition and is valid for HP-UX Release 9.0 on all HP 9000 systems. Replaces Second Edition, HP part number B2355-90004.

June 1991. Second Edition. Update to the First Edition for HP-UX Release 8.05 on Series 700 systems. Also valid for HP-UX Release 8.0 on Series 300/400 and Series 800 systems. Replaces First Edition, HP part number B1864-90000.

January 1991. First Edition. Replaces manual part number 09000-90013. Valid for HP-UX Release 8.0 on Series 300/400, 700, and Series 800 systems. The *Networking Reference* was merged into this manual at Release 8.0.

New Features

This edition contains several new features.

Typography has been changed to conform to style used in other HP manuals as well as industry standards (conversion complete execpt for parts of Volume 3). Command names, argument names, and such appear on the printed page in exactly the same form as when they are typed in commands or applications, eliminating much confusion regarding capitalization of letters, which items are literals or otherwise, etc.

Progressive bleed tabs in each section are positioned vertically on the page edge according to the first letter in the name of the manual entry for easier access.

As part of an on-going effort to improve the quality and usability of this manual, several entries have been expanded and rewritten for better clarity and many examples have been added or expanded in many entries. Many changes are a direct result of comments, requests, and suggestions from users outside of HP.

Manual is expanded considerably to conver new functionality from Open Software Foundation and several other sources as well as newer versions of NFS Services and other software contained in previous releases.

Do You Have Comments or Suggestions?

Comments and suggestions from users about this manual are always welcome because they are an important part of our on-going process of improving the *HP-UX Reference*.

Internal HP users send electronic mail to:

`hpuxref@fc.hp.com`

Other users, please use the reply card provided in the manual or send a note or letter by ordinary mail to:

HP-UX Reference Comments, MS 11
Hewlett-Packard Company
3404 East Harmony Road
Fort Collins, CO 80525-9988, U.S.A.

Notes

Table of Contents for Volume 1

Table of Contents
Volume 1

Section 1: User Commands

Table of Contents
Volume 1

Table of Contents
Volume 1

Table of Contents
Volume 1

Table of Contents
Volume 1

Introduction to HP-UX Reference

INTRODUCTION

The HP-UX operating system is Hewlett-Packard Company's implementation of an operating system originally based on UNIX* System V Release 2, with important features from Berkeley Software Distribution 4.2. The HP-UX operating system has been evolving for nearly a decade. Literally thousands of bug fixes on the original code, countless internal improvements, plus a large selection of enhanced capabilities and other features developed by HP, other companies, educational institutions, and standards organizations have combined to form a robust operating system that is compatible with numerous industry standards.[1]

Release 9.0 of the HP-UX system contains features from:

- UNIX System V Releases 2 and 3, plus features similar to those found in Release 4.
- Features from Berkeley Software Distribution 4.2 and 4.3.
- Software subsystems from Open Software Foundation, Inc. (OSF), Carnegie-Mellon University, Cornell University, Massachussetts Institute of Technology, and numerous other commercial and educational firms and institutions.
- Complete or partial conformance to numerous industry and international standards including, but not limited to AES, SVID2, XPG2, XPG3, XPG4, FIPS 151-1 and 2, POSIX.1, POSIX.2, and ANSI C.

This combination makes HP-UX a very powerful, useful, and reliable operating system capable of supporting a wide range of applications ranging from simple text processing to sophisticated engineering graphics and design, as well as commercial business applications. A broad selection of interfaces and network support capabilities make the HP-UX system suitable for solving tough problems in design, manufacturing, business, and other areas where responsiveness and performance are important.

HP-UX systems are now commonly used in scientific research, automated manufacturing, business management information systems, military applications and space exploration, computer-aided engineering and sofware development, telecommunications, automated merchandising networks, and numerous other applications. Extensive international language support enables HP-UX to interact with users in any of dozens of human languages.

HP-UX interfaces easily with local area networks and resource-sharing facilities. Networking, based on industry-standard protocols, provides flexible interaction with other computers and operating systems. Optional software products extend HP-UX capabilities into a broad range of specialized needs.

This manual is not intended for use as a learning tool for beginners. It is a reference guide that is most useful to experienced users of UNIX or UNIX-like systems. If you are not already familiar with UNIX and HP-UX, refer to the series of Users Guides and other learning documents supplied with your system or available separately. System implementation and maintenance details are explained in the *HP-UX System Administrator* manuals normally furnished with each system.

---

* UNIX and System V are trademarks of UNIX System Laboratories, Inc..

[1] See STANDARDS CONFORMANCE later in this introduction.

MANUAL ORGANIZATION

Due to the size and complexity of the HP-UX operating system, the *HP-UX Reference* is divided into several sections contained in three volumes. Volume 1 contains User commands in Section 1. Sections 2 (System Calls) and 3 (Function Libraries) are in Volume 2. These topics are of interest mainly to programmers. Volume 3 contains a potpourri of subjects not contained in the first two: Section 1M (System Administration Commands) is related to system installation and maintenance. Most commands in this section require super-user privilege before they can be used. Section 4 (File Formats) is of interest mostly to administrators and programmers. Section 5 (Miscellaneous Topics) contains a haphazard collection of miscellany of interest to widely various users. Section 7 (Device Files), like Section 4, contains information commonly needed by administrators and programmers. The Glossary in Section 9 is also of interest to a variety of users, but is not a simple beginner's definition of terms. Rather, it contains often very precise definitions of terms as used in the HP-UX environment. A separate Table of Contents and Index is provided for each respective volume to assist you in finding needed information. The index to Volume 1 also contains references to built-in features in the various command interpreters ("shells").

Section 1	(*User Commands*) describes programs that are usually invoked directly by users or from command language procedures, as opposed to system calls (Section 2) or functions (Section 3) that are called by user and application programs. Most heavily-used commands reside in the directory **/bin** (for *bin*ary programs). Other less frequently used programs reside in **/usr/bin** to save space in **/bin** and to reduce search time for commonly-used commands. These directories are normally searched automatically by the command interpreter called a shell (*sh*(1), *csh*(1)), *ksh*(1)), or other command interpreter facilities. Most other Section 1 commands are located in **/lib** and **/usr/lib**. Refer to *hier*(5) and tutorial manuals supplied with your system for more information about file system structure.
Section 1M	(*System Administration Commands*) describes commands used for system maintenance including boot processes, crash recovery, system integrity testing, and other needs. This section contains topics that pertain primarily to system administrator and super-user tasks.
Section 2	(*System Calls*) describes entries into the HP-UX kernel, including the C-language interface.
Section 3	(*Functions and Function Libraries*) describes available functions that reside (in binary form) in various system libraries stored in directories **/lib** and **/usr/lib**. Refer to *intro*(3C) for descriptions of these libraries and the files where they are stored.
Section 4	(*File Formats*) documents the structure of various types of files. For example, the link editor output-file format is described in *a.out*(4). Files that are used only by a single command (such as intermediate files used by assemblers) are not described. C-language **struct** declarations corresponding to the formats in Section 4 can be found in directories **/usr/include** and **/usr/include/sys**.
Section 5	(*Miscellaneous*) contains a variety of information such as descriptions of header files, character sets, macro packages, and other topics.
Section 6	(*Games*) is absent because no games are currently supported on HP-UX.
Section 7	(*Device Special Files*) discusses the characteristics of special (device) files that provide the link between HP-UX and system I/O devices. The names for each topic usually refer to the type of I/O device rather than to the names of individual special files.
Section 9	(*Glossary*) is located in Volume 3 after Section 9. It defines selected terms used in this manual.
Index	An alphabetical listing of keywords and topics based on the NAME line on the first page of each manual entry as well as other information. The right-hand column refers to the manual entry name (Section number is in parentheses).

Each section (except 9) contains a number of independent entries usually referred to as **manual entries**, but also called **manpages** or **manual pages**. Each manual entry consists of one or more printed pages, with the entry name and section number printed in the upper corners of each page. Entries are arranged alphabetically within each section of the reference, except for the introductory entry at the beginning of each section. Textual cross-references to other manual entries are of the form *pagename*(nL) where n is the section number and L is a letter representing a subsection where applicable. For example, *io_burst*(3I) refers to an entry in the Device I/O function library (Section 3, subsection I) named *io_burst*.

Each printed page has two page numbers printed at the bottom of the page. The center page number is numbered starting over with page 1 at the beginning of each entry, and is placed between two dashes in normal typeface. Another number in bold is printed at the outside corner on each page. This page number is part of a continuous sequence as in a normal book, and is primarily for the convenience of support and manufacturing personnel. Normal users most typically locate entries by the alphabetical headings at the top of the page as when using a dictionary.

Some manual entries describe two or more commands or routines in a single entry. In such cases, the entry is not duplicated for each topic, but appears only once, usually arranged under the first keyword appearing in the NAME section of the entry. Occasionally, an entry name does not appear on the NAME line. In such instances, the name describes the keywords in more general terms such as the entry for *acct* or *acctsh* in Section 1M or *string* in Section 3.

SYSTEM STANDARDIZATION

This reference is based on extensive system-design control documents that have been used to ensure software compatiblity across HP-UX computer model lines, as well as with widely used industry standards. HP-UX is compatible with the UNIX System V Interface Definition (SVID), but also includes many popular features from the Berkeley Software Distributions (BSD), plus HP and other enhancements for international language support, and numerous other features which are now specified by industry and international standards.

As of this printing, HP-UX has been implemented on HP 9000 Series 200, 300, 400, 500, 600, 700, and 800 computers. This document is valid for HP-UX Release 9.0 on Series 300/400, 700, and 800 systems.

Page Headers

Great effort has been expended to make the HP-UX operating system compatible between all series. However, there are significant hardware differences between various series, making some features that may be highly desirable on one series inappropriate or useless on the other (such as *isl*(1M) or *hpuxboot*(1M) utilities used on Series 800 that have no meaning for Series 300 systems). If an entry pertains to one or more, but not all series, it is clearly noted at the top of each page of that entry.

Some entries in this manual pertain to optional software subsystems (such as LISP, for example) that are not normally shipped with HP-UX. Such manual entries are clearly marked as such. They are provided here for the convenience of users whose systems have the optional software installed on them.

MANUAL ENTRY FORMATS

All manual entries follow an established topic format, but not all topics are included in each entry.

NAME	Gives the name(s) of the entry and briefly states its purpose.
SYNOPSIS	Summarizes the use of the entry or program entity being described. A few conventions are used: `Computer font` strings are literals, and are to be typed exactly as they appear in the manual (except for parameters in the SYNOPSIS section of entries in Sections 2 and 3). *Italic* strings represent substitutable argument names and names of manual entries found elsewhere in the manual. **Boldface** is used primarily for new terms the first time they are used. in some entries in Volume 3, boldface is used for literals where the conversion to computer font was not completed in time for this edition. Square brackets [ ] around an argument name indicate that the argument is optional. Ellipses (...) are used to show that the previous argument can be repeated. A final convention is used by the commands themselves. An argument beginning with a dash (-), a plus sign (+), or an equal sign (=) is often taken to be some sort of option argument, even if it appears in a position where a file name could appear. Therefore it is unwise to have files names that begin with -, +, or =.
DESCRIPTION	Discusses the function and behavior of each entry.
EXTERNAL INFLUENCES	Information under this heading pertains to programming for various spoken languages. Typical entries indicate support for single- and/or multi-byte characters, the effect of language-related environment variables on system behavior, and other related information.
NETWORKING FEATURES	Information under this heading is applicable only if you are using the network feature described there (such as NFS).
RETURN VALUE	Discusses various values returned upon completion of program calls.
DIAGNOSTICS	Discusses diagnostic indications that may be produced. Self-explanatory messages are not listed.
ERRORS	Lists error conditions and their corresponding error message or return value.
EXAMPLES	Provides examples of typical usage, where appropriate.
WARNINGS	Points out potential pitfalls.
DEPENDENCIES	Points out variations in HP-UX operation that are related to the use of specific hardware or hardware combinations.
AUTHOR	Indicates the origin of the software documented by the manual entry.

FILES	Lists file names that are built into the program or command.
SEE ALSO	Provides pointers to related topics.
BUGS	Discusses known bugs and deficiencies, occasionally suggesting fixes.
STANDARDS CONFORMANCE	For each HP-UX utility, function, or other component addressed by one or more of the following formal and defacto industry standards, this section lists the standard specifications to which that HP-UX component conforms. The organization which produced the industry standard is shown below in parentheses.

The meanings of the notations used for the various standards are:

AES	Application Environment Specification, Operating System Programming Interfaces Volume, Revision A (OSF). Currently supported only on Series 300, 400, and 700 systems.
SVID2	System V Interface Definition Issue 2, Volumes 1-3 (AT&T).
XPG2	X/Open Portability Guide Issue 2 (X/Open, Ltd.)
XPG3	X/Open Portability Guide Issue 3 (X/Open, Ltd.)
XPG4	X/Open CAE Specifications – System Interface Definitions, and System Interfaces and Headers, Issue 4 (X/Open, Ltd.)
FIPS	151-2 Federal Information Processing Standards 151-1 and 151-2 (National Institute of Standards and Technology)
POSIX.1	IEEE Standard 1003.1-1988, IEEE Standard 1003.1-1990, and ISO/IEC 9945-1:1990 (IEEE Computer Society and ISO)
POSIX.2	IEEE Standard 1003.2-1992, and ISO/IEC 9945-2:1992, including all options except the Localedef and User Portability Utilities Options. (IEEE Computer Society and ISO/IEC)
ANSI	C American National Standard for Programming Language C and ISO/IEC 9899 (ANSI X3J11 and ISO)

The table of contents included at the beginning of each volume contains a complete listing of all manual entries in the order they appear in each section, as well as alphabetically intermixed lists of all keywords that appear in manual entries covering multiple keywords. This combination provides an easy path for locating commands and features whose keyword names are not the same as the title heading on the corresponding manual entry.

HOW TO GET STARTED

This discussion provides a very brief overview of how to use the HP-UX system: how to log in and log out, how to communicate through your machine, how to run a program, and how to access an electronic copy of this manual on your system. If you are a beginning user, refer to other tutorial manuals for a more complete introduction to the system.

Logging In

To log in you must have a valid user name, which can be obtained from your system administrator. Press the **BREAK** key to get a **login:** message if it is not present.

When a connection has been established, the system displays **login:** on your terminal. Type your user name then press the **RETURN** key. If a password is required (strongly recommended!), the system asks for it, but does not print it on the terminal.

It is important that you type in your login name in lowercase if possible. If you type uppercase letters, HP-UX assumes that your terminal cannot generate lowercase letters, and treats subsequent uppercase input as lowercase. When you have logged in successfully, the shell displays a **$** prompt unless programmed for a different prompt (the shell is described below under the heading: "How to Run a Program").

For more information, consult *login*(1) and *getty*(1M), which discuss the login sequence in more detail, and *stty*(1), which tells you how to describe the characteristics of your terminal to the system (*profile*(4) explains how to accomplish this last task automatically every time you log in).

Logging Out

You can log out by typing an end-of-file indication (ASCII **EOT** character, usually typed as control-d) to the shell (see *csh*(1) and *ksh*(1) for information about **ignoreeof** if you are using C shell or Korn shell). The shell will terminate and the **login:** message will appear again.

How to Communicate Through Your Terminal

HP-UX gathers keyboard input characters and saves them in a buffer. The accumulated characters are not passed to the shell or other program until a **RETURN** is typed.

HP-UX terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is printing on your display or terminal. Of course, if you type during output, the output will have the input characters interspersed in it. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is severely overloaded or operating abnormally. When the read-ahead limit is exceeded, the system throws away **all** the saved characters.

Erase, Kill, and Output Stop/Resume Characters

By default, the character @ "kills" all characters typed before it on an input line from the terminal. The character # erases the last character typed. Successive uses of # will erase characters back to, but not beyond, the beginning of the input line; @ and # can be used as normal text characters by preceding them with \ (thus to erase a \, you need two #s). These default erase and kill characters can be changed, and usually are (see *stty*(1)).

The ASCII **DC3** (control-S) character can be used to temporarily stop output. It is commonly used on video terminals to suspend output to the display while you read what is already being displayed. You can then resume output to the display by typing a **DC1** (control-Q). When **DC1** (control-Q) or **DC3** (control-S) are used to suspend or restart output, they are not sent to the keyboard command-line buffer for passing to the program. However, any other characters typed on the keyboard are saved and used as input later in the program.

Interrupt and Quit Characters

The ASCII **DEL** character (sometimes labelled "rubout" or "rub") is not passed to programs, but instead generates an *interrupt signal*. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed*(1), for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editing operation without losing the file being edited.

The *quit* signal is generated by typing the ASCII **FS** (control-\) character. It causes a running program to terminate.

End-of-Line and Tab Characters

Besides adapting to the speed of the terminal, HP-UX tries to be intelligent as to whether you have a terminal with a new-line (line-feed) key, or whether it must be simulated with a carriage-return and line-feed pair. In the latter case, all incoming carriage-return characters are changed to line-feed characters (the standard line delimiter), and a carriage-return/line-feed pair is echoed to the terminal. If you get into the wrong mode, see *stty*(1).

Tab characters are used freely in HP-UX source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. The *stty*(1) command sets or resets this mode. The system assumes that tabs are set every eight character positions. The *tabs*(1) command can set tab stops on your terminal, if the terminal supports tabs.

How to Run a Program

When you have successfully logged into HP-UX, a program called a shell is monitoring input from your terminal. The shell accepts typed lines from the terminal, splits them into command names and arguments, then executes the command which is nothing more than an executable program. Usually, the shell looks first in your current directory (discussed below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space and/or tab characters.

When a program terminates, the shell will ordinarily regain control and prompt you with a **$** (unless redefined to some other prompt), to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh*(1).

The Current Directory

HP-UX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is assumed to be in that directory by default. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. The permissions you have in other directories and files will have been granted or denied to you by their respective owners, or by the system administrator. To change the current working directory use *cd*(1).

On-Line Reference Manual

The *HP-UX Reference* is also available on-line by using the *man*(1) command if manual entries are present on the system. Refer to the *man*(1) manual entry in Volume 1 for more information.

Path Names

To refer to files not in the current directory, you must use a path name. Full (absolute) path names begin with /, which is the name of the *root* directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a /), until finally the file name is reached (e.g., **/usr/ae/filex** refers to file **filex** in directory **ae**, while **ae** is itself a subdirectory of **usr**; **usr** is a subdirectory of the root directory). See the glossary (Section 9) for a formal definition of *path name*.

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (without a prefixed /). Generally, a path name can be used anywhere a file name is required.

Important commands that modify the contents of directories are *cp*(1), *mv*(1), and *rm*(1), which respectively copy, move (i.e., rename, relocate, or both), and remove files. To determine the status of files or the contents of directories, use *ls*(1). Use *mkdir*(1) for making directories, *rmdir*(1) for destroying them, and *mv*(1) for renaming them.

For a more complete discussion of the file system, see the references cited at the beginning of the *Introduction* above. It may also be useful to glance through Section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Writing a Program

To enter the text of a source program into an HP-UX file, use *vi*(1) (preferred by most users), *ex*(1), or *ed*(1). The three principal languages available under HP-UX are C (see *cc*(1)), FORTRAN (see *f77*(1)), and Pascal (see *pc*(1)). After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file named **a.out** in the current directory. Since the results of a subsequent compilation may also be placed in **a.out**, thus overwriting the current output, you may want to use *mv*(1) to give the output a unique name. If the program is written in assembly language, you will probably need to link library functions with it (see *ld*(1)). FORTRAN, C, and Pascal call the linker automatically.

When you have gone through this entire process without encountering any diagnostics, the resulting program can be run by giving its name to the shell in response to the prompt.

Your programs can receive arguments from the command line just as system programs do by using the *argc* and *argv* parameters. See the supplied C tutorial for details.

Text Processing

Almost all text is entered through a text editor. The editor preferred above all others provided with HP-UX is the *vi* editor. For batch-processing text files, the *sed* editor is very efficient. Other editors are used much less frequently. The *ex* editor is useful for handling certain situations while using *vi* but most other editors are rarely used except in various scripts.

The following editors are the same program masquerading under various names: *vi*, *view*, and *vedit* (see *vi*(1)), *ex* (see *ex*(1)), and *edit* (see *edit*(1)). For information about the streaming editor *sed*, see *sed*(1). The *ed* line editor is described in the *ed*(1) manual entry.

The commands most often used to display text on a terminal are *more*(1), *cat*(1), and *pr*(1). The *cat*(1) command simply dumps ASCII text on the terminal, with no processing at all. The *more*(1) command displays text on the terminal a screenful at a time, pausing for an acknowledgement from the user before continuing. The *pr*(1) command paginates text, supplies headings, and has a facility for multi-column output. It is most commonly used in conjunction with *lp*(1) to pipe formatted text to a line printer.

Inter-User Communication

Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be beneficial to learn about them, because someone else may direct them toward you. To communicate with another user currently logged in, *write*(1) can be used to transfer text directly to that user's terminal display (if permission to do so has been granted by the user). Otherwise, *mailx*(1) or *mail*(1) sends a message to that user's mailbox. The user is then informed by HP-UX that mail has arrived (if currently logged in) or mail is present (when he or she next logs in). Refer to the *mailx*, *mail*, and *write* manual entries in Section 1 for explanations of how each of these commands is used.

When you log in, a message-of-the-day may greet you before the first prompt.

HP-UX FILE SYSTEMS

HP-UX supports two basic file systems, depending on which series you are using. A third file system is described for historical purposes.

HFS	High-performance File System. This file system format is implemented on all current HP 9000 HP-UX systems.
SDF	Structured Directory Format. This file system format is implemented on all Series 500 releases. Use the SDF utilities such as *sdfcp*(1), *sdfinit*(1), *sdfrm*(1), *sdffind*(1), etc. to access SDF files from other systems.
BFS	Bell File System. This obsolete file system was implemented on the Integral PC and Series 200 systems prior to HP-UX Release 5.0. BFS files are no longer supported on HP-UX systems (starting at Release 7.0).

File system formats are transparent to most users, and are of little importance in most applications. Most of the time, formats only prevent direct reading of disks of a particular format on a machine that supports a different format. Thus, SDF cannot be read on an HFS system without using SDF utilities. However, an SDF-based system can readily transfer files to an HFS-based system over UUCP, LAN, or other supported data communication facilities.

When transportable data is needed, a tape cartridge, flexible disk, or optical read/write disk can be used. Flexible disks can be readily formatted and read or written in LIF (Logical Interchange Format) by using the *lifinit*, *lifcp*, *lifls*, *lifrename*, and *lifrm* commands in Section 1. LIF media is readily usable on other non-HP-UX systems that support the HP LIF format.

IMPORTANT NOTES ABOUT HP-UX RELEASES 9.0

This edition of the *HP-UX Reference* documents HP-UX Release 9.0 on HP 9000 Series 300, 400, 700, and 800 systems. However, there are some important differences in versions of the HP-UX operating system corresponding to each series as noted below:

Series 300/400	Series 300 and 400 systems are architecturally very similar. Except for the very few isolated cases where indicated otherwise, they are functionally identical.
Series 800	HP 9000 Series 800 computers use a PA-RISC CPU architecture with an I/O architecture that is better suited to the needs of large multi-user systems. In addition, some Series 800 models can be equipped with multiple, parallel processors for higher system speeds.
Series 700	HP 9000 Series 700 computers use a PA-RISC CPU architecture that is similar to the processors used on Series 800 systems, but Series 700 I/O architecture resembles that of Series 300/400 systems. Therefore, there are important differences in certain aspects of system behavior as documented under DEPENDENCIES in applicable manual entries.

Release 9.0 contains numerous improvements and enhancements, many of which are invisible to ordinary users, other than in overall system performance. Many new features such as dynamic buffer cache, logical volume management, software disk striping, compiler optimizations, and memory-mapped files provide significant performance increases in addition to the improvements in raw hardware performance of new computer models over earlier machines.

Online help facilities and other features have been added at 9.0 to improve overall system usability.

For more information, consult the release notes files contained in directory **`/etc/newconfig`** on your system.

Notes

Section 1:
User Commands

NAME

intro - introduction to command utilities and application programs

DESCRIPTION

This section describes commands accessible by users, as opposed to system calls in Section (2) or library routines in Section (3), which are accessible by user programs.

Command Syntax

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option* (*s*)] [*cmd_arg* (*s*)]

where the elements are defined as follows:

name Name of an executable file.

option One or more *option*s can appear on a command line. Each takes one of the following forms:

-*no_arg_letter*
A single letter representing an option without an argument.

-*no_arg_letters*
Two or more single-letter options combined into a single command-line argument.

-*arg_letter*<>*opt_arg*
A single-letter option followed by a required argument where:
arg_letter
is the single letter representing an option that requires an argument,
opt_arg
is an argument (character string) satisfying the preceding *arg_letter*,
<> represents optional white space.

cmd_arg Path name (or other command argument) *not* beginning with -, or - by itself indicating the standard input. If two or more *cmd_arg*s appear, they must be separated by white space.

RETURN STATUS

Upon termination, each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (for descriptions, see *wait*(2) and *exit*(2)). The system-supplied byte is 0 for normal termination. The byte provided by the program is customarily 0 for successful execution and non-zero to indicate errors or failure such as incorrect parameters in the command line, or bad or inaccessible data. Values returned are usually called variously "exit code", "exit status", "return code", or "return value", and are described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings, and therefore become confused when they encounter a null character (the string terminator) within a line.

SEE ALSO

getopt(1), exit(2), wait(2), getopt(3C), hier(5).

The introduction to this manual.

NAME

adb - absolute debugger

SYNOPSIS

adb [**-w**] [**-I***dir*] [*objfil* [*corfil*]]

DESCRIPTION

adb is a general-purpose debugging program that is sensitive to the underlying architecture of the processor on which it runs. It can be used to examine files and provide a controlled environment for executing HP-UX programs.

objfil is normally an executable program file, preferably containing a symbol table; if not, the symbolic features of **adb** cannot be used, although the file can still be examined. The default for *objfil* is **a.out**. *corfil* is assumed to be a core image file produced after executing *objfil*. The default for *corfil* is **core**.

Requests to **adb** are read from standard input and **adb** responds on standard output. If the **-w** flag is present, *objfil* is created (if necessary) and opened for reading and writing, to be modified using **adb**. The **-I** option specifies a directory where files read with **$<** or **$<<** (see below) are sought; the default is **/usr/lib/adb**. **adb** ignores QUIT; INTERRUPT causes return to the next **adb** command.

Requests to **adb** follow the form:

[*address*] [**,** *count*] [*command*] [**;**]

If *address* is present, *dot* is set to *address*. Initially *dot* is set to **0**. For most commands, *count* specifies the number of times the command is to be executed. The default *count* is **1**. *address* and *count* are expressions.

The interpretation of an address depends on the context in which it is used. If a subprocess is being debugged, addresses are interpreted in the address space of the subprocess. (For further details of address mapping see Addresses below.)

Expressions

Expressions are interpreted as follows:

.	The value of *dot*.
+	The value of *dot* increased by the current increment.
^	The value of *dot* decreased by the current decrement.
"	The last *address* typed.
integer	A number. The prefix **0** (zero) forces interpretation in octal radix; the prefixes **0d** and **0D** force interpretation in decimal radix; the prefixes **0x** and **0X** force interpretation in hexadecimal radix. Thus **020** = **0d16** = **0x10** = sixteen. If no prefix appears, the *default radix* is used; see the **$d** command. The radix is initialized to the base used in the assembly language for the processor involved. Note that a hexadecimal number whose most significant digit would otherwise be an alphabetic character must have a **0x** (or **0X**) prefix.
integer **.** *fraction*	A 32-bit floating-point number.
'*cccc***'**	The ASCII value of up to 4 characters. A backslash (\\) can be used to escape a single quote (**'**).
< *name*	*name* can have the value of either a variable or a register. **adb** maintains a number of variables named by single letters or digits; see Variables below. If *name* is a register, the value of the register is obtained from the CORE_PROC segment in *corfil* (before the subprocess is initiated) or from the user area of the subprocess. Register names are implementation dependent; see the **$r** command.
symbol	A *symbol* is a sequence of uppercase or lowercase letters, underscores, or digits, not starting with a digit. A backslash (\\) can be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial underscore (_) is prefixed to *symbol*, if needed.
_ *symbol*	If the compiler prefixes _ to an external symbol, it may be necessary to cite this name to distinguish it from a symbol generated in assembly language.

(*exp*)	The value of the expression *exp*.

The following are monadic operators:

**exp*	The contents of the location addressed by *exp* in *corfil*.
@ *exp*	The contents of the location addressed by *exp* in *objfil*.
-*exp*	Integer negation.
~*exp*	Bitwise complement.

The following dyadic operators are left associative and are less binding than monadic operators:

e1+*e2*	Integer addition.
e1-*e2*	Integer subtraction.
*e1***e2*	Integer multiplication.
e1%*e2*	Integer division.
e1&*e2*	Bitwise conjunction.
e1 | *e2*	Bitwise disjunction.
e1#*e2*	*e1* rounded up to the next multiple of *e2*.

Commands

Most commands consist of an action character followed by a modifier or list of modifiers. The following action characters can take format specifiers. (The action characters `?` and `/` can be followed by `*`; see Addresses for further details.)

?*f*	Locations starting at *address* in *objfil* are printed according to the format *f*. *dot* is incremented by the sum of the increments for each format letter. If a subprocess has been initiated, *address* references a location in the address space of the subprocess instead of *objfil*.
/*f*	Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is increased like `?`. If a subprocess has been initiated, *address* refers to a location in the address space of the subprocess instead of *corfil*.
=*f*	The value of *address* is printed in the styles indicated by the format *f*. (For `i` format `?` is printed for the parts of the instruction that refer to subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character can be preceded by an integer that indicates how many times the format is repeated. While stepping through a format, *dot* is increased by the amount given for each format character. If no format is given then the last format is used.

The following format characters are available:

`o` 2	Print 2 bytes in octal. All octal numbers output by `adb` are preceded by `0`.
`O` 4	Print 4 bytes in octal.
`q` 2	Print 2 bytes in signed octal.
`Q` 4	Print 4 bytes in signed octal.
`d` 2	Print 2 bytes in decimal.
`D` 4	Print 4 bytes in decimal.
`x` 2	Print 2 bytes in hexadecimal.
`X` 4	Print 4 bytes in hexadecimal.
`u` 2	Print 2 bytes as an unsigned decimal number.
`U` 4	Print 4 bytes as an unsigned decimal number.
`f` 4	Print the 32 bit value as a floating point number.

`F` 8	Print double floating point.
`b` 1	Print the addressed byte in hexadecimal.
`B` 1	Print the addressed byte in octal.
`c` 1	Print the addressed character (the sign bit is ignored).
`C` 1	Print the addressed character using the following escape convention. First, the sign bit is discarded, then character values `000` to `040` are printed as @ followed by the corresponding character in the range `0100` to `0140`. The character @ is printed as @@.
`s` *n*	Print the addressed characters until a zero character is reached.
`S` *n*	Print a string using the @ escape convention. The value *n* is the length of the string including its zero terminator.
`Y` 4	Print 4 bytes in date format (see *ctime*(3C)).
`i` *n*	Print as machine instructions. The value of *n* is the number of bytes occupied by the instruction.
`a` 0	Print the value of *dot* in symbolic form.
`p` *n*	Print the addressed value in symbolic form. The value of *n* is a machine-dependent constant.
`t` 0	When preceded by an integer, moves to the next appropriate tab stop. For example, `8t` moves to the next 8-space tab stop.
`r` 0	Print a space.
`n` 0	Print a new-line character.
"..." 0	Print the enclosed string.
`^`	*dot* is decreased by the current increment. Nothing is printed.
`+`	*dot* is increased by 1. Nothing is printed.
`-`	*dot* is decreased by 1. Nothing is printed.
new-line	Repeat the previous command with a *count* of 1. The value of *dot* continues from the end of the previous format, unlike a [`?` `/`] command with no *address,* which repeats the previous *address* value. New-line can also be used to repeat a `:s` or `:c` command; however, any arguments to the previous command are lost.

[`?` `/`]`l` *value mask*

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If `L` is used, `adb` looks to match 4 bytes at a time instead of 2. If no match is found, *dot* is left unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted `-1` is used.

[`?` `/`]`w` *value ...*

Write the 2-byte *value* into the addressed location. If the command is `W`, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

`=m`

Toggle the address mapping of *corfil* between the initial map set up for a valid core file and the default mapping pair which the user can modify with `/m`. If the *corfil* was invalid, only the default mapping is available.

[`?`/]`m` *b1 e1 f1*[`?`/]

Record new values for (*b1*, *e1*, *f1*). If fewer than three expressions are given, the remaining map parameters are left unchanged. If the `?` or `/` is followed by `*`, the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If the list is terminated by `?` or `/`, the file (*objfil* or *corfil*, respectively) is used for subsequent requests. (For example, `/m?` causes `/` to refer to *objfil*.) A `/m` command switches the *corfil* mapping to the default mapping pair. For a valid core file, the `=m` command can be used to switch back to the initial mapping.

>*name*	Assign *dot* to the variable or register named.
`!`	Call a shell to read the remainder of the line following `!`.

The following `$` commands take the form `$`*modifier*:

`$<`*f*	Read commands from the file *f*. If this command is executed in a file, further commands in the file are not seen. If a *count* is given, and is zero, the command is ignored. The value of the count is placed in variable 9 before the first command in *f* is executed.
`$<<`*f*	Similar to `$<` except it can be used in a file of commands without causing the file to be closed. Variable 9 is saved when the command executes and is restored when it completes. Only five `$<<` files can be open at once.
`$>`*f*	Send output to the file *f*, which is created if it does not already exist.
`$r`	Print the general registers and the instruction addressed by the process counter. *dot* is set to the process counter contents.
`$f`	Print the floating-point registers.
`$b`	Print all breakpoints and their associated counts and commands.
`$c`	C stack backtrace. If *address* is given, it is taken as the address of the current frame (instead of the normal stack frame pointer). If *count* is given, only the first *count* frames are printed.
`$e`	The names and values of external variables are printed.
`$w`	Set the page width for output to *address* (default 80).
`$s`	Set the limit for symbol matches to *address*. The default is system dependent.
`$o`	The default for all integers input is octal.
`$d`	Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus `10$d` never changes the default radix. To make decimal the default radix, use `0d10$d`. To make decimal the default radix, use `0t10$d`.
`$x`	The default for all integers input is hexadecimal.
`$q`	Exit from `adb`.
`$v`	Print all non-zero variables in the current radix.
`$m`	Print the address map. This includes both the initial and default maps for a valid *corfil* with an indication of which is currently active.
`$`new-line	print the process id and register values.

The available `:` commands manage subprocesses, and take the form `:`*modifier*:

`:b`*c*	Set breakpoint at *address*. The breakpoint is executed *count*−1 times before causing a stop. Each time the breakpoint is encountered, the command *c* is executed. If this command sets *dot* to zero, the breakpoint causes a stop.
`:d`	Delete breakpoint at *address*. `:d*` deletes all breakpoints.
`:r`	Run *objfil* as a subprocess. If *address* is given explicitly, the program is entered at this point; otherwise the program is entered at its standard entry point. The value *count* specifies how many breakpoints are ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with `<` or `>` causes the standard input or output to be established for the command. All signals are turned on when entering the subprocess.
`:e`	Set up a subprocess as in `:r`; no instructions are executed.
`:c`*s*	Continue the subprocess with signal *s* (see *signal*(5)). If *address* is given, the subprocess continues at this address. If no signal is specified, the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for `:r`.

:ss	As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess, *objfil* is run as a subprocess as for **:r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
:Ss	Same as **:c** except that a temporary breakpoint is set at the next instruction. Useful for stepping across subroutines.
:x *a* [*b*]...	Execute subroutine *a* with parameters [*b*]...
:k	Terminate the current subprocess, if any.

Variables

adb provides named and numbered variables. Named variables are set initially by **adb** but are not used subsequently. Numbered variables are reserved for communication as follows:

0	The last value printed.
1	The last offset part of an instruction source.
2	The previous value of variable 1.
9	The count on the last **$<** command.

On entry, the following named variables are set from the coreheaders in the *corfil*. If *corfil* does not appear to be a **core** file, these values are set from *objfil*.

b	The base address of the data segment.
d	The data segment size.
s	The stack segment size.
t	The text segment size.

The following variables are set from *objfil*.

e	The entry point.
m	The "magic" number as defined in <**magic.h**>.

Addresses

The file address associated with a written address is determined by a mapping described below; see **$m**. Both the *objfil* mapping and the default *corfil* mapping are represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*). The initial mapping for a valid *corfil* contains a triple for each segment (coreheader).

The *file address* corresponding to a written *address* is calculated as follows:

If

$b1 \le address < e1,\ file\ address = address + f1 - b1,$

otherwise, if

$b2 \le address < e2,\ file\ address = address + f2 - b2,$

otherwise, the requested *address* is not valid. For a valid *corfil*, this pattern repeats as many times as there are segments (coreheaders) in the *corfil*, rather than twice. If **?** or **/** is followed by *****, only the second triple is used, or (when using the initial mapping of a valid *corfil*) only segments with a **CORE_STACK** coreheader.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected, **adb** sets *b1* to **0**, *e1* to the maximum file size, and *f1* to **0**; in this way the entire file can be examined with no address translation.

adb keeps all appropriate values as signed 32-bit integers so that it can be used on large files.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

adb comments about inaccessible files, syntax errors, abnormal termination of commands, etc. It echoes **adb** when there is no current command or format. Exit status is **0**, unless the last command failed or returned non-zero status.

DEPENDENCIES

Shared Libraries

Setting breakpoints in shared libraries is not supported.

`adb` does not read the linker symbol table for shared libraries, and cannot access locations in shared libraries by name. In a stack backtrace (`$c`), `adb` does not know the names of shared library procedures. If this is a problem, consider using *xdb*(1) instead.

On PA-RISC machines, if the core file was created when the program was in a shared library function, the `$c` command does not work. When a stack backtrace for the core file encounters a shared library procedure on the stack
it aborts at that point. If this is a problem, consider using *xdb*(1) instead.

Series 300/400

The `-I` option is not currently supported.

The `I` format prints machine instructions, such as `i`, except that immediate constants are printed in decimal.

The command `$n` is provided to set the number of significant digits for floating-point dumps to *address.*

The `$d` command sets the default for all integers input to decimal, regardless of the value of *address.*

The *count* is ignored for the `$<` command (i.e., variable `9` is not updated and a *count* of zero does not cause the command to be ignored), and the `$<<` command is not supported.

The following variables are also supported:

`f`	If this is set to a non-zero value, any sequence of machine instructions that effectively constitute a single floating-point accelerator instruction are treated as a single instruction for machine level single-stepping and display.
`r`	If `f` is set to a non-zero value, `r` indicates which address register is used in floating-point accelerator instruction sequences. A `0` corresponds to register `a0`, `1` to `a1`, etc. The default value is `2`.

Series 700/800

A leading zero by itself is not recognized as a radix indicator. Use the prefixes `0o` or `0O` (zero-oh) to force interpretation in octal radix. The prefixes `0t` and `0T` are also accepted to force interpretation in decimal radix. Thus `0o20` = `0t16` = sixteen. A hexadecimal number whose most significant digit would otherwise be an alphabetic character may begin with a leading zero instead of `0x` (or `0X`), if the default radix is hexadecimal.

The `$f` command prints floating point registers as 32-bit single precision and `$F` prints these registers as 64-bit doubles.

`$R` prints all registers available to `adb` users.

The `:x` and `:S` commands are not currently supported.

The following options are also supported:

`-k` Allows virtual-to-physical address translation, useful for kernel debugging. In this case, *core* should be an HP-UX crash dump or `/dev/mem`.

When `adb` is invoked with this option, it sets up the context of the currently running process using space registers four through seven. A user specified address is dereferenced by combining it with the appropriate space register, depending on the quadrant in which the 32-bit address lies. The `$p` command is provided to change the current context. The *address* argument is the address of the process structure corresponding to the desired context.

When the current radix is not (decimal) ten, the `-k` option allows `adb` to support the notion of long pointers or addresses in the form *space* `.` *offset.* Once a space is specified, all subsequent addresses are dereferenced using that space until the user enters another long address. If a space equal to (hexadecimal) 0xffffffff is used, `adb` reverts to the previous context and uses space registers four through seven to dereference 32-bit addresses.

-P*pid* Causes `adb` to adopt process *pid* as a "traced" process (see *ptrace*(2)). This option is helpful for debugging processes that were not originally run under the control of `adb`.

`adb` can be used to inspect relocatable object files; it reads the symbol table and sets up the appropriate mappings for text and data. Note that relocatable object files do not necessarily contain an exact image of the initialized data; however, if this is the case, the data mapping is not set.

AUTHOR

`adb` was developed by AT&T and HP.

FILES

```
a.out
core
/dev/mem
/dev/kmem
/dev/swap
```

SEE ALSO

ptrace(2), crt0(3), ctime(3C), end(3C), a.out(4), core(4), signal(5).

adb Debugger tutorial in *Assembler Reference and Tools* manual.

NAME

adjust - simple text formatter

SYNOPSIS

`adjust` [`-bcjr`] [`-m`*column*] [`-t` *tabsize*] [*files* ...]

DESCRIPTION

`adjust` is a simple text formatter for filling, centering, left and right justifying, or right-justifying text paragraphs, and is designed for interactive use. It reads the concatenation of input files (or standard input if none are given) and produces on standard output a formatted version of its input, with each paragraph formatted separately. If `-` is given as an input filename, `adjust` reads standard input at that point (use `--` as an argument to separate `-` from options.)

`adjust` reads text from input lines as a series of words separated by space characters, tabs, or newlines. Text lines are grouped into paragraphs separated by blank lines. By default, text is copied directly to the output, subject only to simple filling (see below) with a right margin of 72, and leading spaces are converted to tabs where possible.

Options

`adjust` recognizes the following command-line options:

`-b` Do not convert leading space characters to tabs on output (output contains no tabs, even if there were tabs in input).

`-c` Center text on each line. Lines are pre- and post-processed, but no filling is done.

`-j` Justify text. After filling, insert spaces in each line as needed to right-justify it (except in the last line of each paragraph) while keeping the justified left margin.

`-r` After filling text, adjust the indentation of each line for a smooth right margin (ragged left margin).

`-m`*column*

Set the right fill margin to the given column number, instead of 72. Text is filled, and optionally right justified, so that no output line extends beyond this column (if possible). If `-m0` is given, the current right margin of the first line of each paragraph is used for that and all subsequent lines in the paragraph.

By default, text is centered on column 40. With `-c`, the `-m` option sets the middle column of the centering "window", but `-m0` auto-sets the right side as before (which then determines the center of the "window").

`-t`*tabsize* Set the tab size to other than the default (eight columns).

Only one of the `-c`, `-j`, and `-r` options is allowed in a single command line.

Details

Before doing anything else to a line of input text, `adjust` first handles backspaces, rubbing out preceding characters in the usual way. Next it ignores all non-printable characters except tab. It then expands all tabs to spaces.

For simple text filling, the first word of the first line of each paragraph is indented the same amount as in the input line. Each word is then carried to the output followed by one space. "Words" ending in *terminal_character*[*quote*][*closing_character*] are followed by two spaces, where *terminal_character* is any of `.`, `:`, `?`, or `!`; *quote* is a single closing quote (`'`) character or double-quote character (`"`), and *close* is any of `)`, `]`, or `}`. Here are some examples:

end. of? sentence.' sorts!" of.) words?"]

(*adjust* does not place two spaces after a pair of single closing quotes (`''`) following a *terminal_character*).

`adjust` starts a new output line whenever adding a word (other than the first one) to the current line would exceed the right margin.

`adjust` understands indented first lines of paragraphs (such as this one) when filling. The second and subsequent lines of each paragraph are indented the same amount as the second line of the input paragraph if there is a second line, else the same as the first line.

- * `adjust` also has a rudimentary understanding of tagged paragraphs (such as this one) when filling. If the second line of a paragraph is indented more than the first, and the first line has a word beginning at the same indentation as the second line, the input column position of the tag word or words (prior to the one matching the second-line indentation) is preserved.

Tag words are passed through without change of column position, even if they extend beyond the right margin. The rest of the line is filled or right-justified from the position of the first non-tag word.

When `-j` is given, `adjust` uses an intelligent algorithm to insert spaces in output lines where they are most needed, until the lines extend to the right margin. First, all one-space word separators are examined. One space is added to each separator, starting with the one having the most letters between it and the preceding and following separators, until the modified line reaches the right margin. If all one-space separators are increased to two spaces and more spaces must be inserted, the algorithm is repeated with two-space separators, and so on.

Output line indentation is held to one less than the right margin. If a single word is larger than the line size (right margin minus indentation), that word appears on a line by itself, properly indented, and extends beyond the right margin. However, if `-r` is used, such words are still right-justified, if possible.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines what are valid space and printable characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `adjust` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

DIAGNOSTICS

`adjust` complains to standard error and later returns a non-zero value if any input file cannot be opened (it skips the file). It does the same (but quits immediately) if the argument to `-m` or `-t` is out of range, or if the program is improperly invoked.

Input lines longer than `BUFSIZ` are silently split (before tab expansion) or truncated (afterwards). Lines too wide to center begin in column 1 (no leading spaces).

WARNINGS

This program is designed to be simple and fast. It does not recognize backslash to escape white space or other characters. It does not recognize tagged paragraphs where the tag is on a line by itself. It knows that lines end in new-line or null, and how to deal with tabs and backspaces, but it does not do anything special with other characters such as form feed (they are simply ignored). For complex operations, standard text processors are likely to be more appropriate.

This program could be implemented instead as a set of independent programs, fill, center, and justify (with `-r` option). However, this would be much less efficient in actual use, especially given the program's special knowledge of tagged paragraphs and last lines of paragraphs.

EXAMPLES

This command is useful for filtering text while in *vi*(1). For example,

```
!}adjust
```

reformats the rest of the current paragraph (from the current line down), evening the lines.

The `vi` command:

```
:map ^X {!}adjust -j^V^M
```

(where `^` denotes control characters) sets up a useful "finger macro". Typing `^X` (Ctrl-X) reformats the entire current paragraph.

`adjust -m1` is a simple way to break text into separate words without white space, except for tagged-paragraph tags.

AUTHOR

`adjust` was developed by HP.

SEE ALSO

nroff(1).

NAME

admin - create and administer SCCS files

SYNOPSIS

`admin` [`-n`][`-i` [*name*]][`-r` [*rel*]][`-t` [*name*]][`-f`*flag* [*flag-val*]] ... [`-d`*flag* [*flag-val*]] ... [`-a` [*login*]] ... [`-e` [*login*]] ... [`-m` [*mrlist*]] ... [`-y` [*comment*]][`-h`][`-z`] *file* ...

DESCRIPTION

`admin` is used to create new SCCS files and change parameters of existing ones. Arguments to `admin`, which may appear in any order, consist of option arguments, beginning with `-`, and named *file*s (note that SCCS file names must begin with the characters `s.`). If a named *file* does not exist, it is created and its parameters are initialized according to the specified option arguments. Parameters not initialized by a option argument are assigned a default value. If a named *file* does exist, parameters corresponding to specified option arguments are changed, and other parameters are left unaltered.

If a *directory* is named instead of *file*, `admin` acts on each *file* in *directory*, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read, and each line of the standard input is assumed to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

`admin` option arguments apply independently to all named *file*s, whether one file or many. In the following discussion, each option is explained as if only one file is specified, although they affect single or multiple files identically.

Options

`admin` supports the following options and command-line arguments:

`-n` This option indicates that a new SCCS file is to be created.

`-i`[*name*] The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see `-r` option for delta numbering scheme). If the `i` option is used but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this option is omitted, the SCCS file is created with an empty initial delta. Only one SCCS file can be created by an `admin` command on which the `i` option is supplied. Using a single `admin` to create two or more SCCS files requires that they be created empty (no `-i` option). Note that the `-i` option implies the `-n` option.

`-r`*rel* The *rel*ease into which the initial delta is inserted. This option can be used only if the `-i` option is also used. If the `-r` option is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

`-t`[*name*] The *name* of a file from which descriptive text for the SCCS file is to be taken If the `-t` option is used and `admin` is creating a new SCCS file (the `-n` and/or `-i` options are also used), the descriptive text file name must also be supplied. In the case of existing SCCS files:

- A `-t` option without a file name causes removal of descriptive text (if any) currently in the SCCS file, and
- A `-t` option with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

`-f`*flag* This option specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several `f` options can be supplied on a single `admin` command line. The allowable *flag*s and their values are:

`b` Allows use of the `-b` option on a *get*(1) command to create branch deltas.

`c`*ceil* The highest release (i.e., "ceiling"), a number less than or equal to 9999, which can be retrieved by a *get*(1) command for editing. The default value for an unspecified `c` flag is 9999.

`f`*floor* The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get*(1) command for editing.

The default value for an unspecified `f` flag is 1.

`d`*SID*	The default delta number *SID* to be used by a `get` command (see *get*(1)).
`i`*str*	Causes the `No id keywords (cm7)` message issued by `get` or `delta` to be treated as a fatal error (see *delta*(1)). In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string. However the string must contain a keyword, and no embedded newlines.
`j`	Allows concurrent `get` commands for editing on the same *SID* of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file. Only one user can perform concurrent edits. Access by multiple users is usually accomplished by using a common login or a set-user-ID program (see *chmod*(1) and *exec*(2)).
`l`*list*	A *list* of releases to which deltas can no longer be made (`get -e` against one of these "locked" releases fails). The *list* has the following syntax: *list* `::=` *range* `|` *list* `,` *range* / *range* `::=` *RELEASE NUMBER* `|` `a` — The character `a` in the *list* is equivalent to specifying *all releases* for the named SCCS file. Omitting any list is equivalent to `a`.
`n`	Causes `delta` to create a "null" delta in each of those releases being skipped (if any) when a delta is made in a *new* release (such as when making delta 5.1 after delta 2.7, release 3, and release 4 are skipped). These null deltas serve as "anchor points" so that branch deltas can later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.
`q`*text*	User-definable text substituted for all occurrences of the `%Q%` keyword in SCCS file text retrieved by `get`.
`m`*mod*	*module* name of the SCCS file substituted for all occurrences of the `%M%` keyword in SCCS file text retrieved by `get`. If the `m` flag is not specified, the value assigned is the name of the SCCS file with the leading `s.` removed.
`t`*type*	*type* of module in the SCCS file substituted for all occurrences of `%Y%` keyword in SCCS file text retrieved by `get`.
`v`[*pgm*]	Causes *delta*(1) to prompt for Modification Request (*MR*) numbers as the reason for creating a delta. The optional value specifies the name of an (*MR*) number validity checking program (see *delta*(1)). (If this flag is set when creating an SCCS file, the `m` option must also be used even if its value is null).

`-d`*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The `-d` option can be specified only when processing existing SCCS files. Several `-d` options can be supplied on a single `admin` command. See the `-f` option for allowable *flag* names.

`l`*list*	A *list* of releases to be "unlocked". See the `-f` option for a description of the `l` flag and the syntax of a *list*.

`-a`*login* A *login* name, or numerical HP-UX group ID, to be added to the list of users allowed to make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several `a` options can be used on a single `admin` command line. As many *login*s or numerical group IDs as desired can be on the list simultaneously. If the list of users is empty, anyone can add deltas. If *login* or group ID is preceded by a `!` they are to be denied permission to make deltas.

-e*login*
: A *login* name or numerical group ID to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several `e` options can be used on a single `admin` command line.

-y[*comment*]
: The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the `-y` option results in a default comment line being inserted in the form:

  `date and time created`*YY* / *MM* / *DD* / *HH* / *MM* / *SS* `by` *login*

  The `-y` option is valid only if the `-i` and/or `-n` options are specified (i.e., a new SCCS file is being created).

-m[*mrlist*]
: The list of Modification Request (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta, in a manner identical to *delta*(1). The `v` flag must be set and the (*MR*) numbers are validated if the `v` flag has a value (the name of an (*MR*) number validation program). Diagnostic messages occur if the `v` flag is not set or (*MR*) validation fails.

-h
: Causes `admin` to check the structure of the SCCS file (see *sccsfile*(4)), and to compare a newly computed checksum (the sum of all the characters in the SCCS file except those in the first line) with the checksum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

  This option inhibits writing on the file, thus cancelling the effect of any other options supplied, and is, therefore, only meaningful when processing existing files.

-z
: The SCCS file checksum is recomputed and stored in the first line of the SCCS file (see `-h`, above).

  Note that use of this option on a truly corrupted file can prevent future detection of the corruption.

Access Control Lists (ACLs)

Do not add optional ACL entries to SCCS files. SCCS removes them, possibly causing unexpected and undesirable access modes.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters.

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `admin` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte file names are *not* supported.

WARNINGS

SCCS files can be any length, but the number of lines in the text file itself cannot exceed 99 999 lines.

DIAGNOSTICS

Use *help*(1) for explanations.

FILES

The last component of all SCCS file names must be of the form `s.`*file-name*. New SCCS files are given mode 444 (see *chmod*(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by `admin` is to a temporary x-file, called `x.`*file-name*, (see *get*(1)), created with mode 444 if the `admin` command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed to the name of the SCCS file This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of any given directory allows only the owner to modify SCCS files contained in that directory. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode can be changed to 644 by the owner, thus allowing use of `vi` or any other suitable editor. *Care must be taken!* The edited file should *always* be processed by an `admin -h` to check for corruption followed by an `admin -z` to generate a proper checksum. Another `admin -h` is recommended to ensure the SCCS file is valid.

`admin` also makes use of a transient lock file called `z.`*file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get*(1) for further information.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4), acl(5).

SCCS User's Guide, in *Programming on HP-UX.*

STANDARDS CONFORMANCE

admin: SVID2, XPG2, XPG3

NAME

ar - maintain portable archives and libraries

SYNOPSIS

`ar` *key* [*posname*] *afile* [*name*] ...

DESCRIPTION

ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor (see *ld*(1). It can be used, however, for any similar purpose. The magic string and file headers used by **ar** consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

Individual files are inserted without conversion into the archive file. When **ar** creates an archive, it creates headers in a format that is portable across all machines. See *ar*(4) for a detailed description of the portable archive format and structure. The archive symbol table (described in *ar*(4)) is used by the link editor (see *ld*(1)) to search repeatedly and efficiently through libraries of object files. An archive symbol table is created and maintained by **ar** only when the archive contains at least one object file. The archive symbol table is in a specially named file that is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *ar*(1) command is used to create or update the contents of an archive, the symbol table is rebuilt. The **s** modifier described below forces the symbol table to be rebuilt.

key must be present, and consists of an optional **-**, followed by one operation character from the set **drqtpmx**, optionally concatenated with one or more modifier characters from the set **vuaibcls**. *afile* is the archive file. Constituent files in the archive file are specified by *name* arguments.

The **TMPDIR** environment variable can be set to specify a directory for temporary files (see *tmpnam*(3S)). The **l** modifier overrides the **TMPDIR** variable, and **TMPDIR** overrides the default directory **/usr/tmp**

The following *key* operation characters are recognized:

d Delete the named files from the archive file.

r Replace the named files, or add a new file to the archive:

- If the modifier **u** is used with the operation character **r**, only those files with dates of modification later than the archive files are replaced.
- If an optional positioning character from the set **abi** is used, the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. In the absence of a positioning character, new files are placed at the end.
- **ar** creates *afile* if it does not already exist.
- If no *name* is specified and:
  - the specified archive file does not exist, **ar** creates an empty archive file containing only the archive header (see *ar*(4)).
  - the archive contains one or more files whose names match names in the current directory, each matching archive file is replaced by the corresponding local file without considering which file may be newer unless the **u** modifier is also specified.

q Quickly append the named files to the end of the archive file. Positioning characters are invalid. The operation does not check to determine whether the added members are already in the archive. This is useful only to avoid quadratic behavior when creating a large archive piece-by-piece. **ar** creates *afile* if it does not already exist.

t Print a table of contents of the archive file. If no names are given, all files in the archive are described. If names are given, information about only those files appears.

p Print the named files in the archive.

m Move the named files. By default, the files are moved to the end of the archive. If a positioning character is present, the *posname* argument must be present and, as in **r**, *posname* specifies where the files are to be moved. Note that, when used with a positioning character, the files are moved in the same order that they currently appear in the archive, *not* in the order specified on the command line. See EXAMPLES.

`x`	Extract the named files. If no names are given, all files in the archive are extracted. In neither case does `x` alter (i.e., delete entries from) the archive file.

The following optional modifiers are recognized:

`c`	Create. For `r` and `q` operations, `ar` normally creates *afile* if it does not already exist. The `c` modifier suppresses the message normally produced when *afile* is created.
`f`	Force. Truncate filenames to 14 characters before comparing with existing filenames in the archive, which are already truncated to 14 characters. When used with the `r` operation, the first existing file that matches the truncated filename is replaced. The `f` modifier can also be used with other operations to allow the full filenames to be specified, rather than the truncated filenames.
`l`	Local. Place temporary files in the local current working directory rather than in the directory specified by the environment variable **TMPDIR** or in the default directory **/usr/tmp**. Only the `d`, `m`, and `r` operations and the `s` modifier use temporary files.
`s`	Force the regeneration of the archive symbol table even if `ar` is not invoked with an operation that modifies the archive contents. This modifier is useful for restoring the archive symbol table after the `strip` command has been used on the archive (see *strip*(1)).
`u`	Update. (`r` operations only) Do not copy the local file to the archive unless the local file is newer than the corresponding existing file in the archive.
`v`	Verbose. Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with `t`, `v` gives a long listing of all information about the files. When used with the `d`, `m`, `p`, `q`, or `x` operations, the verbose modifier causes `ar` to print each *key* operation character and file name associated with that operation. For the `r` operation, `ar` shows an `a` if it adds a new file or an `r` if it replaces an existing one.
`A`	Suppress warning messages regarding optional access control list entries. *ar*(1) does not archive optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file having optional access control list entries.

Only the following combinations are meaningful; no other combination of modifiers with operations have any effect on the operation:

`d`:	`v`, `f`, `l`
`r`:	`u`, `v`, `c`, `f`, `l`, `A`, and `a` | `b` | `i`
`q`:	`v`, `c`
`t`:	`v`, `s`
`p`:	`v`, `f`, `s`
`m`:	`v`, `f`, `l`, and `a` | `b` | `i`
`x`:	`v`, `f`, `s`

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `ar` behaves as if all internationalization variables are set to "C". See *environ*(5).

DIAGNOSTICS

`phase error` on *file name*
: The named file was modified by another process while `ar` was copying it into the archive. When this happens, `ar` exits and the original archive is left untouched.

`ar write error:` file system error message
: `ar` could not write to a temporary file or the final output file. If `ar` was trying to write the final output file, the original archive is lost.

`ar` reports `cannot create` *file.a*, where *file.a* is an `ar`-format archive file, even if *file.a* already exists. This message is triggered when *file.a* is write-protected or inaccessible.

EXAMPLES

Create a new file (if one does not already exist) in archive format with its constituents entered in the order indicated:

```
ar r newlib.a f3 f2 f1 f4
```

Replace files f2 and f3 such that the new copies follow file f1, and f3 follows f2:

```
ar ma f1 newlib.a f2 f3
ar ma f2 newlib.a f3
ar r newlib.a f2 f3
```

The archive is then ordered:

```
newlib.a:  f1 f2' f3' f4
```

where the single quote marks indicate updated files. The first command says "move f2 and f3 after f1 in *newlib.a*", thus creating the order:

```
f1 f3 f2 f4
```

Note that the relative order of f2 and f3 has not changed. The second command says "move f3 after f2 in *newlib.a*", creating the order:

```
f1 f2 f3 f4
```

The third command then replaces files f2 and f3. Since files f2 and f3 both already existed in the archive, this sequence of commands could not be simply replaced by:

```
ar ra f1 newlib.a f2 f3
```

because the previous position and relative order of f2 and f3 in the archive are preserved (no matter how the files are specified on the command line), producing the following archive:

```
newlib.a:  f3' f2' f1 f4
```

WARNINGS

If you are a user who has appropriate privileges, `ar` alters any archive file, even if it is write-protected.

If the same file is mentioned twice in an argument list, it might be put in the archive twice.

`ar` automatically creates an archive symbol table, a task performed in early HP-UX versions by `ranlib`. If a `ranlib` command is executed, the following message is displayed:

```
ranlib: ar already did it for you, see ar(1).
```

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP BLS environment.

FILES

`/usr/tmp/ar*`
temporary files

SEE ALSO

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4), acl(5).

STANDARDS CONFORMANCE

`ar`: SVID2, XPG2, XPG3, POSIX.2

NAME

as - assembler

SYNOPSIS

as [**-A**] [**-a***afile*] [**-o***objfile*] [*file*]

REMARKS

This is a generic entry for a machine-dependent assembler. A specific entry is provided for each assembler. Refer to manual entry *as_300*(1) for information about the Series 300/400 assembler or *as_800*(1) for information about the Series 700/800 assembler.

DESCRIPTION

as assembles the named *file*, or the standard input if no file name is specified. The optional arguments **-A** or **-a** can be used to obtain an assembly listing with offsets and instruction codes. If **-A** is used the listing goes to standard output. If **-a** is used the listing goes to *afile*.

All undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file *objfile*; if that is omitted, **.s** is stripped from the end of the file name (if there) and **.o** is appended to it. This becomes the name of the output file. **as** does not invoke **ld**.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

FILES

`/usr/tmp/*`	temporary files
`file.o`	object file

SEE ALSO

adb(1), as_300(1), as_800(1), ld(1), nm(1), nm_300(1), nm_800(1), crt0(3), a.out(4), a.out_300(4), a.out_800(4).

DIAGNOSTICS

If the name chosen for the output file is of the form ***?.[cs]**, the assembler issues an appropriate complaint and quits. When syntactic or semantic errors occur, a single-line diagnostic is displayed on standard error together with the line number and the name of the file where it occurred.

STANDARDS CONFORMANCE

as: SVID2, XPG2

NAME

as - assembler for MC68000, MC68010, MC68020, MC68030 and MC68040

SYNOPSIS

`as` [*options*] [*file*]
`as20` [*options*] [*file*]

DESCRIPTION

`as` assembles the named *file* (which usually has a `.s` suffix as in `my_prog.s`). If *file* is not specified or if - is given, standard input is used instead.

The assembler can be invoked as `as` (`/bin/as`) or `as20` (`/bin/as20`).

By default `as` produces object code for the MC68020, MC68030 and MC68040 processors. The `-d` option can be used to produce object code for use with the MC68000 and MC68010 processors.

All undefined symbols in the assembly are treated as global.

Options

`as` recognizes the following options:

`-L`	Generate entries in the linker symbol table for local as well as global symbols. Normally, only global and undefined symbols are entered into the table. This option is useful when using `adb` to debug assembly language programs (see *adb*(1)).
`-l`	Generate entries in the linker symbol table for all global and undefined symbols, and for all local symbols except those with `.` or `L` as the first character. This option is useful when using tools such as `prof` on files generated by *cc*(1) or *f77*(1) (see *prof*(1)). Linker symbol table entries are generated for user-defined local names, but not for compiler-generated local names.
`-m`	Process the input file using the `m4` macro preprocessor before assembling it (see *m4*(1)).
`-d`	Cause *as* to generate short-displacement forms for MC68010-compatible syntaxes, including forward references.
`-o` *objfile*	Cause output object code to be placed in file *objfile*. If `-o` is not specified and the source file is read from the standard input and the object file is written to `a.out`. If `-o` is not specified and the source file is not the standard input, the object file is written to a file whose name is created by removing the `.s` suffix (if present) from the basename of filename *file*, then adding a `.o` suffix to the base filename. The object `.o` file is placed in the current directory. To prevent improper interpretation of other options, the name of *objfile* cannot begin with the character - or +. To prevent accidental overwriting of source files, *objfile* cannot end with *.s* or *.c*.
`-w`	Suppress warning messages (errors are not suppressed).
`-A`	Generate an assembly listing with offsets, a hexadecimal dump of the generated code, and the source text. The assembly listing goes to standard output (*stdout*). This option cannot be used when input is *stdin*.
`-a` *listfile*	Generate an assembly listing in file *listfile*. The listing option cannot be used when input is *stdin*. The name of *listfile* cannot end with *.c* or *.s* and cannot start with the character - or +.
`-O`	Enable span-dependent optimization. Optimization is disabled by default.
`-V` *number*	Set the `a_stamp` field in the `a.out` header to be *number*. The `-V` option overrides any `version` pseudo-op in the assembly source. By default the `a_stamp` field is set to zero (see the *HP Assembler Reference Manual*).
`+z`	Generate an object file for use in a shared library. Instructions with a 16-bit offset are used to access globals.
`+Z`	Generate an object file for use in a shared library. Instructions with a 32-bit offset are used to access globals. This option results in slower code for global variable access than the `+z` option, and should only be used if necessary (`ld` gives an error if

+z is insufficient — see *ld*(1)).

+s	Generate an object file for use in a dynamic load library.
-i	For external subroutine calls in position-independent code, generate a PC-relative fixup rather than procedure linkage table fixup. This option is only meaningful when used with either the +z or +Z option.

Wherever possible, the assembler should be accessed through a compilation system interface program such as *cc*(1).

The MC68010 instruction set is a superset of that of the MC68000. The MC68020 and MC68030 instruction sets are identical and are a superset of the MC68010. The MC68040 supports all non-privileged instructions of the MC68030 (see the *HP-UX Assembler Reference Manual* for details).

The **as** assembler supports the complete MC68000, MC68010, MC68020, MC68030 and MC68040 instruction sets. However, if you are writing code for an MC68000 or MC68010 processor, you must limit instructions and program structures to those supported by the microprocessor. Executing an unsupported instruction on an MC68000 or MC68010 processor causes an illegal instruction trap during program execution, but might not produce an error during program assembly and loading. In addition, the following instructions are not fully supported by Series 300/400 hardware, and should not be used in assembly code written for Series 300/400 HP-UX machines: **tas**, **cas**, **cas2**, and **bkpt**.

The +z, +Z, and -i options are used to assemble code for inclusion in a shared library. However, use of these options is not sufficient; the code must be PIC (position independent code). For details on shared libraries and PIC refer to the manual *Programming on HP-UX*.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

If the name chosen for the output file ends with .c or .s or starts with the character + or -, the assembler issues an appropriate complaint and quits. When a syntactic or semantic error occurs, a single-line diagnostic is produced that includes the line number and file name where the error occurred.

WARNINGS

If the -m option is used, keywords for **m4** cannot be used as symbols in the input file because **m4** cannot determine which are assembler symbols and which are real **m4** macros.

The displacement value for the **movp** instruction must be a first-pass absolute 16-bit value.

Expressions cannot have more than one forward-referenced symbol, except for the special form <symbol>-<symbol>.

AUTHOR

as was developed by HP.

FILES

/usr/tmp/*	temporary files, which can be changed by using TMPDIR (see *tmpnam*(3S)).
file.o	object file

SEE ALSO

as_800(1) (Series 800 Implementation), adb(1), astrn(1), atime(1), atrans(1), cc(1), f77(1), ld(1), m4(1), nm_300(1), nm_800(1), prof(1), crt0(3), tmpnam(3s), a.out_300(4), a.out_800(4).

HP-UX Assembler Reference Manual and ADB Tutorial for Series 300/400 Computers,

Programming on HP-UX.

NAME
as - assembler (Precision Architecture)

SYNOPSIS
as [[*option*] ... [*file*] ...] ...

DESCRIPTION
as assembles the named source file *file*, or the standard input if *file* is not specified. The **-l** option causes the assembler to produce an assembly listing with offsets.

Assembler output is stored in file *objfile*. If the **-o***outfile* option is not specified, the **.s** suffix (if present) is stripped from the end of the source file name and **.o** is appended to the name to form the name of the default object code output file.

as output is not optimized. **as** creates relocatable object files which must be processed by **ld** before they can be successfully executed.

cc assembles **.s** files together with **/lib/pcc_prefix.s**, and subsequently invokes **ld**.

Options
as recognizes the following options.

-e	An unlimited number of errors will be tolerated before the assembly process is abandoned. By default, one hundred errors are allowed before aborting.
-f	Procedures by default will be callers of other procedures. The normal default is that procedures do not call other procedures.
-l	Listing to standard output is made of the program after assembly. This listing shows offsets of instructions and actual values for fields.
-o *outfile*	Produce an output object file by the name *outfile* instead of using the default **.o** suffix.
-s	The output file will have suffix **.ss** and be of a format suitable for conversion to the ROM burning programs.
-u	Unwind descriptors will not be created. To avoid the need for **.CALLINFO**, **.ENTER** and **.LEAVE** must not have been used.
-v *xrfile*	Provides the name of a file to which cross reference data is written.

EXTERNAL INFLUENCES
International Code Set Support
Single- and multi-byte character code sets are supported.

DIAGNOSTICS
When syntactic or semantic errors occur, a single-line diagnostic is displayed on standard error, together with the line number and the file name in which it occurred.

WARNINGS
as does not do macro processing.

Trailing operands (except for a **pc_relative** branch displacement) can be omitted, in which case they default to zero. Trailing commas can also be omitted. Leading commas are ignored.

FILES
/lib/pcc_prefix.s	space and register definitions
/usr/include/hard_reg.h	hardware register equates
/usr/include/soft_reg.h	follows calling convention
/usr/include/std_space.h	space and subspace declarations
/lib/as_msgs.cat	error message catalog
file.o	object file

SEE ALSO
adb(1), cc(1), ld(1), nm_800(1), nm_300(1), crt0(3).

Assembly Language Reference Manual,

Precision Architecture and Instruction Reference Manual, *Procedure Calling Conventions Reference Manual* .

NAME

asa - interpret ASA carriage control characters

SYNOPSIS

`asa` [*files*]

DESCRIPTION

`asa` interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments, or the standard input if `-` is specified or if no file names or given. The first character of each line is assumed to be a control character. The following control characters are interpeted as indicated:

(blank)	Output a single new-line character before printing.
`0`	Output two new-line characters before printing.
`1`	Output a new-page character before printing.
`+`	Overprint previous line.

Lines beginning with other than the above characters are treated the same as lines beginning with a blank. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic is sent to standard error. This program forces the first line of each input file to start on a new page.

To view the output of FORTRAN programs which use ASA carriage control characters and have them appear in normal form, `asa` can be used as a filter:

```
a.out | asa | lp
```

The output, properly formatted and paginated, is then directed to the line printer. FORTRAN output previously sent to a file can be viewed on a user terminal screen by using:

`asa` *file*

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

SEE ALSO

efl(1), f77(1), fsplit(1), ratfor(1).

STANDARDS CONFORMANCE

`asa`: POSIX.2

NAME

astrn - translate assembly language

SYNOPSIS

`astrn` [*filename*]

DESCRIPTION

`astrn` translates an assembly language *source file* from previous HP-UX `Series 300/400` assembly language syntax to new `Series 300/400` HP-UX assembly language syntax. If no *filename* is given, input is assumed to come from standard input.

If an opcode is not recognized, a warning message is given and the entire line is passed through unchanged. For any syntax error detected such that translation cannot continue, `astrn` reports an error and translation terminates.

Lines longer than 132 characters are truncated to 132 characters.

For a line beginning with * (indicating a comment), the * is translated to a # but is preceded by a blank to allow preprocessing by `cpp` (see *cpp(1))*.

Absolute displacements off the program counter cannot be guaranteed to translate correctly. Any line referencing the program counter will be flagged by a warning message.

Certain capabilities supported on the old assembler are not accepted by the new assembler. These include:

- The *alias* and *include* pseudo-ops are not supported. An error message is given and translation terminates.
- The new assembler restricts expressions involving forward references for which *astrn* makes no check. Such references may involve only a single symbol, a symbol plus or minus an absolute expression, or the subtraction of two symbols.
- The characters `$`, `@`, `?`, and `\177` are no longer accepted as valid identifier characters. These are translated to `S`, `A`, `Q`, and `D` respectively, and a warning is issued.
- Span-dependent branches `j`*cc* are translated to `b`*cc*`.w`.
- An identifier equated to a register name will be translated, but the assembler will report an error.
- Local labels are translated to a concatenation of the nearest previous ordinary label and the local label itself. This includes changing the `$` to a `S`.

DEPENDENCIES

`astrn` is implemented on Series 300/400 only.

SEE ALSO

as(1), atrans(1).

NAME

at, batch - execute commands at a later time

SYNOPSIS

at [**-m**] [**-f** *filename*] [**-q** *queue*] *time* [*date*] [[**next** | *+increment*] *time_designation*] *job* ...

at -r *job_id* ...
at -l [*job_id* ...]

batch

DESCRIPTION

at, in the first form shown above, and **batch** read commands from standard input to be executed at a later time:

at	Executes commands at a specified time.
batch	Executes commands when system load level permits.

In the second and third forms, **at** respectively removes one or more currently scheduled jobs, or lists some or all currently scheduled jobs.

An **at_job** consists of one or more executable commands exectuable by the shell. **at** creates a shell script in **/usr/spool/cron/atjobs**, the first part of which sets up the environment to match that of the invoking user. The second part of the script consists of the commands entered by the user. When **cron** dispatches the job it **exec**s a shell to execute the command file (script).

Options

at recognizes the following options and command-line arguments where *job* is any valid HP-UX command:

-l [*job_id* ...]
List all jobs currently scheduled for the invoking user. If *job_ids* are given, only the specified jobs are listed.

-r *job_id* ...
Remove the jobs with the specified *job_ids* that were previously scheduled by the **at** command. *Job_id* is the job number assigned by **at** when the job was originally scheduled. When removing multiple jobs, use blanks to separate *job_ids*.

-m Send mail to the invoking user after the job has run, announcing its completion. Standard output and standard error produced by the job are mailed to the user as well, unless they were redirected elsewhere within the job.

-f *filename* Specify the pathname of a file to be used as the source of the job, instead of standard input.

-q *queue* Submit the specified job to the *queue* indicated (see *queuedefs*(4)). Queues **a**, **b**, and **d** through **y** can be used. **at** uses queue **a** by default. All queues require a time designation except queue **b** which runs as soon as system load level permits. Queue **b** is reserved for use by the **batch** command.

time Can be specified as one, two, or four digits. One- and two-digit numbers represent hours; four digits represent hours and minutes. Alternately, *time* can be specified as two numbers separated by a colon (**:**), single quote (**'**), the letter "h" (**h**), a period (**.**), or comma (**,**). If defined in *langinfo*(3C), special time unit characters can be used. A suffix **am** or **pm** can be appended. Otherwise a 24-hour clock time is understood. For example, **8:15**, **8'15**, **8h15**, **8.15**, and **8,15** are read as 15 minutes after 8 in the morning. The suffixes **zulu** and **utc** can be used to indicate Coordinated Universal Time. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

date (optional) Can be specified as either a day of the week (fully spelled out or abbreviated) or a date consisting of a day, a month, and optionally a year. The day and year fields must be numeric, and the month can be either fully spelled out, abbreviated, or numeric. These three fields can be in any order, and separated by punctuation marks such as **/**, **-**, **.**, or **,**. If defined in *langinfo*(3C), special date unit characters can be present. Two special "days", **today** and **tomorrow**, are also recognized. If no *date* is given, **today** is assumed if the given time is greater than the current time; **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed. If a given date is ambiguous (such as 2/5), the D_T_FMT string (if defined in *langinfo*(3C)) is used to resolve the ambiguity.

`next` or *+increment*
: (optional) If followed by a *time_designation* of **`minutes`**, **`hours`**, **`days`**, **`weeks`**, **`months`**, or **`years`**, lets the user schedule a task to be executed when the specified *time_designation* has elapsed. A numerical operator, *+increment,* enables the user to schedule the task several hours, days, weeks, months, or years in advance (see EXAMPLES). Using the argument **`next`** is equivalent to using an *increment* of **`+1`**. Both plural and singular forms of *time_designation* are accepted.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, *umask* (see *umask*(1)) and *ulimit* (see *ulimit*(2)) are retained when the commands are executed (see *proto*(4)). Open file descriptors, traps, and priority are lost.

Only users whose names appear in file **`/usr/lib/cron/at.allow`** can run **`at`**. If that file does not exist, file **`/usr/lib/cron/at.deny`** is checked to determine if the user should be denied access to **`at`**. If neither file exists, only root is allowed to submit a job. If only **`at.deny`** exists but is empty, global usage is permitted. The allow/deny files consist of one user name per line.

The words **`today`**, **`tomorrow`**, **`noon`**, **`midnight`**, **`now`**, **`minutes`**, **`hours`**, **`days`**, **`weeks`**, **`months`**, **`years`** and their singular forms are replaced by the local language equivalent (see EXTERNAL INFLUENCES below).

`at` and **`batch`** write the *job_id* and schedule time to standard error.

`batch` submits a batch job. It is similar to **`at now`**, but with the following differences: **`batch`** goes into a different queue; **`at now`** responds with error messages.

`at -r` removes jobs previously scheduled by **`at`** or **`batch`**. The *job_id* is the number returned by the **`at`** or **`batch`** command. To get job numbers, typing **`at -l`**. Only users with appropriate privileges can remove jobs other than their own.

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings.

LANG determines the translation of the words **`today`**, **`tomorrow`**, **`noon`**, **`midnight`**, **`now`**, **`minutes`**, **`hours`**, **`days`**, **`weeks`**, **`months`**, **`years`**, **`next`**, and their singular forms. **LANG** also determines the language in which messages are displayed.

If **`LC_TIME`** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **`at`** behaves as if all internationalization variables are set to "C". See IR environ(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

Exit code 0 is returned upon successful completion, otherwise 1 is returned.

DIAGNOSTICS

`at` complains about syntax errors and out-of-range times.

If login shell is not **`/bin/sh`**, **`at`** produces a warning message as a reminder that **`at`** jobs are executed using **`/bin/sh`**.

EXAMPLES

The **`at`** and **`batch`** commands read from standard input the commands to be executed at a later time, unless the **`-f`** option is specified. *sh*(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

The following sequence can be used at a terminal to redirect output:

```
batch
nroff filename > outfile
<Ctrl-D>
```

This sequence demonstrates redirecting standard error to a pipe and is useful in a shell procedure. Note that the sequence of output redirection specifications is significant:

```
batch <<!
nroff filename 2>&1 > outfile | mail loginid
!
```

To perform a task at 5:00 am next Tuesday, use

```
at 5am tuesday next week
```

To perform a task at 5:00 am one week from Tuesday (that is, 2 Tuesdays in advance) use

```
at 5am tuesday + 2 weeks
```

To have a job reschedule itself, invoke `at` from within the shell procedure by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

The following commands show several forms recognized by `at` and include native language usage:

```
at 0815 Jan 24
at 8:15 Jan 24
at 9:30am tomorrow
at now + 1 day
at -f job 5 pm Friday
at 17:40 Tor.                /* in Danish */
at 17h46 demain           /* in French */
at 5:30  26. Feb. 1988          /* in German */
at 12:00 26-02            /* in Finnish */
```

WARNINGS

If the *date* argument begins with a number and the *time* argument is also numeric without suffix, the *time* argument should be a four-digit number that can be correctly interpreted as hours and minutes.

Do not use both `next` and + *increment* within a single `at` command; only the first operator is accepted and the trailing operator is ignored. No warning or error is produced.

If the FIFO used to communicate with `cron` fills up, `at` is suspended until `cron` has read sufficient messages from the FIFO to make room for the message `at` is trying to write. This condition can occur if `at` is writing messages faster than `cron` can process them or if `cron` is not executing.

AUTHOR

`at` was developed by AT&T and HP.

FILES

`/usr/lib/cron`	main cron directory
`/usr/lib/cron/at.allow`	list of allowed users
`/usr/lib/cron/at.deny`	list of denied users
`/usr/spool/cron/atjobs`	spool area
`/usr/lib/cron/queuedefs`	scheduling information
`/usr/lib/cron/.proto`	prototype information

SEE ALSO

cron(1M), crontab(1), queuedefs(4), proto(4), kill(1), mail(1), nice(1), ps(1), sh(1), hpnls(5).

STANDARDS CONFORMANCE

`at`: SVID2, XPG2, XPG3

`batch`: SVID2, XPG2, XPG3

NAME

atime - time an assembly language instruction sequence

SYNOPSIS

`atime` [*options*] *input_file* [*output_file*]

DESCRIPTION

`atime` provides the means to time MC680X0 assembly code sequences. It takes the *input_file* containing a code sequence and returns performance information to the user. This information can then be compared against information from other code sequence analyses to determine an optimal code sequence. Output is directed to standard output by default or if the *output_file* is -, or to *output_file* if specified.

Additional features allow specifying sets of input data and the relative probability that each of these will occur, obtaining an execution profile of a code sequence being evaluated, automatically cross-checking results between analyses, and conveniently logging results.

`atime` has three modes of operation. Performance analysis is the default mode where a code sequence is executed many times in a loop with `atime` calculating and reporting the average time per iteration. In the execution profiling mode, `atime` runs all or selected data sets and reports the number of times each executable instruction is hit. The third mode is assertion listing. Asserting particular values in a code sequence ensures that various algorithms produce identical results. This mode causes asserted values for all or selected data sets to be listed. This output can be used as verification data for subsequent performance analyses and execution profiles.

Options

`atime` recognizes the following options and command-line arguments:

`-a`*file*	Specify an assertion data file (created by a previous run with the `-l` option.)
`-i`*count*	Specify the minimum number of timing iterations.
`-l`[*name*]	Print asserted values. *name* specifies a `dataset` in the input file. Multiple `-l` options are allowed. Omitting *name* causes assertions for all data sets to be listed. Output can be used to create an assertion file for subsequent `atime` runs.
`-n`	Turn off code sequence listing.
`-p`[*name*]	Perform execution profiling and print the hit count for each timed instruction. *name* specifies a `dataset` in the input file to analyze. If there are multiple `-p` options, printed counts will be the sum for all designated data sets. Omitting *name* causes profiling for all data sets.
`-t` "*text*"	Specify an output title.

Instructions

The following `atime` instructions can appear in the input file and have a format similar to standard assembler instructions. However, they cannot be labelled, each must be placed on a separate line, and comments cannot follow on the same line. For instructions that have a corresponding command line option, the latter take precedence when used.

`assert.{b|w|l}` *name*`,`*location*

Verify a datum. The *name*, an alphabetic character followed by zero or more alphanumeric or underscore characters, identifies an asserted datum across `atime` executions. Any data addressing mode, such as `%d0` or `4(%a4,%d2.w)` can be used for *location*. Executing an `assert` instruction during performance analysis or execution profiling compares the asserted *location* with the corresponding name in an assertion data file. Verification also occurs in the assertion listing mode if an assertion file is specified, although the primary function here is to print name/value pairs.

`assert` "*file*" Specify an assertion data file created from a prior run of `atime` with the `-l` option.

`code even`
`code odd` Change the code to even or odd word alignment.

`comment` *text* Specify comment text for output.

`dataname` *name*`,`*name*`,`*...name*

Define the names of the data entries in `dataset` instructions. Names must begin with a `$` and are followed by one or more alphanumeric or underscore characters.

`dataset`*name*[(*count*)]`,` datum`,` datum,...datum
: Define one data set. *name* identifies the data set for use with the `-l` or `-p` options and must be an alphabetic character followed by zero or more alphanumeric or underscore characters. An optional *count* indicates the relative number of times that the data set will be used during timing. For example, a data set with a *count* of 10 will execute ten times for each seven times that a data set with a *count* of 7 is executed. The default *count* is 1. As a particular data set is under consideration, each of its data is treated as a string for replacing its corresponding **`dataname`** name in any assembly instruction where that name occurs.

`include "`*file*`"`
: Include text from the given *file*. Nested `include`s are not allowed.

`iterate` *count*
: Specify the number of timing iterations for performance analysis. Because the actual count used by **`atime`** must be an integer multiple of the sum of the counts in all **`dataset`** instructions, the *count* specified here is taken as a minimum. If neither this instruction nor a `-i` option appears, the default count is 1000000.

`ldopt` *options*
: Specify options to pass to the link editor.

`nolist`
: Turn off code sequence listing.

`output "`*file*`"`
: Append output from a performance analysis to *file*.

`stack even`
`stack odd`
: Adjust the stack for even or odd word alignment.

`time`
: Designate a code section for timing beginning after this instruction and continuing up to a **`verify`** instruction or end-of-file.

`title`
: Specify an output title.

`verify`
: Designate a section of code to be used for algorithm verification beginning after this line and continuing up to the end-of-file. This section will usually contain one or more **`assert`** instructions.

Input file

The input file contains assembly code source text and **`atime`** instructions and has the following four sections.

`atime` initialization section - This starts at the beginning of the file and continues up to the first MC680X0 assembly instruction or a **`time`**, **`code`**, or **`stack`** instruction. It can contain **`assert`** *file*, **`comment`**, **`dataname`**, **`dataset`**, **`include`**, **`iterate`**, **`ldopt`**, **`nolist`**, **`output`**, and **`title`**.

code initialization section - The code following the **`atime`** initialization section continues up to a **`time`** instruction and typically does the setup for the code to be timed. It can contain any valid MC680X0 assembler instruction or pseudo-op or any of the following **`atime`** instructions: **`code even`**, **`code odd`**, **`stack even`**, **`stack odd`**, or **`include`**. It is only in this section that **`dataname`** *name*s can be used, and each *name* replacement must yield a valid MC680X0 instruction.

timed section - This section starts at the **`time`** pseudo-op and continues up to a **`verify`** instruction or end-of-file. It can contain any valid MC680X0 assembler instruction or pseudo-op or the **`atime include`** instruction.

verify section - This section starts at the **`verify`** instruction and continues up to the end-of-file. It can contain any valid MC680X0 assembler instruction or pseudo-op or the **`atime`** instructions **`include`** or **`assert.{b|w|l}`**.

There must be no branching between input file sections. Each must be entered by falling into it from the previous section. Macros for **`m4`** are not supported, nor are multiple instructions per line (see *m4*(1)). Assembly code can have references to external variables or routines as long as it is guaranteed that these will be resolved during link editing.

DIAGNOSTICS

Error messages from **`atime`** are self-explanatory. Additional error messages may be generated from the assembler or link editor. If assembly fails, an intermediate temporary file is be retained with the error message, indicating its name. This file contains a substantial number of comments to aid in correlating assembly errors back to the actual errors in the input file.

EXAMPLES

To evaluate an algorithm to find the maximum of three integers, the input file to `atime` could contain the following code sequence:

```
          title      Find the maximum of three integers
          comment    Developed by T. R. Crew
          comment    August 15, 1988
          nolist
          dataname                 $arg1   $arg2,   $arg3
          dataset    max1(70),     10,     4,       2
          dataset    max2(35),     5,      11,      0
          dataset    max3(20),     8,      13,      21
          iterate    500000
          assert     "assertfile"
          output     "logfile"
          ldopt      -lm -lc
          stack      even
          mov.l      &$arg1,%d0
          mov.l      &$arg2,%d1
          mov.l      &$arg3,%d2
          code       even
          time
          cmp.l      %d0,%d1
          bge.b      L1
          exg        %d0,%d1
     L1:  cmp.l      %d0,%d2
          bge.b      L2
          exg        %d0,%d2
     L2:
          verify
          assert.l   max,%d0
```

WARNINGS

`atime` determines performance information empirically. Therefore valid results are obtained only if it is run on a quiescent single-user system.

AUTHOR

`atime` was developed by HP.

FILES

`/bin/as`	assembler, as(1)
`/bin/ld`	link editor, ld(1)

SEE ALSO

as(1), gprof(1), ld(1), prof(1).

NAME

atrans - translate assembly language

SYNOPSIS

`atrans` [`-n`] [*filename*]

DESCRIPTION

`atrans` translates an assembly language *source file* from *Series 300/400* Pascal workstation assembly language syntax to *Series 300/400* HP-UX assembly language syntax. If no *filename* is given, input is assumed to come from standard input.

If an opcode is not recognized, the entire line is passed through unchanged. For any syntax error detected such that a line cannot be translated, `atrans` issues an error message.

Lines longer than 132 characters are truncated to 132 characters.

Absolute displacements off the program counter cannot be guaranteed to translate correctly. Any line referencing the program counter will be flagged by a warning message.

The HP-UX assembler restricts expressions involving forward references for which `atrans` makes no check. Such references may involve only a single symbol, a symbol plus or minus an absolute expression, or the subtraction of two symbols.

The characters `$` and `@` are not accepted as valid identifier characters by the HP-UX assembler. These are translated to `S` and `A` respectively, and a warning is issued.

Lines containing the following list of *Series 300/400* Pascal workstation pseudo-ops have no parallel in *Series 300/400* HP-UX syntax and are translated as comment lines: `decimal`, `end`, `llen`, `list`, `lprint`, `nolist`, `noobj`, `nosyms`, `page`, `spc`, `sprint`, `ttl`.

Lines containing `mname`, `include`, or `src` pseudo-ops are translated as comment lines, and a warning is printed stating these are not supported by the *Series 300/400* HP-UX assembler.

The pseudo-ops, `def`, `refa`, and `refr`, are translated as `global`.

Certain pseudo-ops require manual intervention to translate. Each line containing these pseudo-ops causes a message to be printed stating that an error will be generated by the *Series 300/400* HP-UX assembler. These pseudo-ops are: `com`, `lmode`, `org`, `rorg`, `rmode`, `smode`, `start`.

When specifying certain addressing modes, the Pascal workstation assembler may allow operands to appear out of order, whereas the HP-UX assembler does not. `atrans` does not rearrange these into proper order.

The `-n` option converts groups of two or more spaces to tabs.

SEE ALSO

as(1), astrn(1).

NAME

awk - pattern-directed scanning and processing language

SYNOPSIS

awk [-F*fs*] [-v *var=value*] [*prog* | -f *file* ...] [*file* ...]

DESCRIPTION

awk scans each input *file* for lines that match any of a set of patterns specified literally in *prog* or in one or more files specified as **-f** *file*. With each pattern there can be an associated action that is to be performed when a line in a *file* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement, and the associated action is performed for each matched pattern. The file name - means the standard input. Any *file* of the form *var=value* is treated as an assignment, not a filename. An assignment is evaluated at the time it would have been opened if it were a filename, unless the **-v** option is used.

An input line is made up of fields separated by white space, or by regular expression **FS**. The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

Options

awk recognizes the following options and arguments:

-F *fs*	Specify regular expression used to separate fields. The default is to recognize space and tab characters, and to discard leading spaces and tabs. If the **-F** option is used, leading input field separators are no longer discarded.
-f *file*	Specify an awk program file. Up to 100 program files can be specified. The pattern-action statements in these files are executed in the same order as the files were specified.
-v *var=value*	Cause *var=value* assignment to occur before the **BEGIN** action (if it exists) is executed.

Statements

A pattern-action statement has the form:

pattern { *action* }

A missing { *action* } means print the line; a missing pattern always matches. Pattern-action statements are separated by new-lines or semicolons.

An action is a sequence of statements. A statement can be one of the following:

```
if ( expression ) statement [else statement ]
while ( expression ) statement
for ( expression ; expression ; expression ) statement
for ( var in array) statement
do statement while ( expression)
break
continue
{ [statement ...] }
expression                            # commonly var=expression
print [ expression-list ] [> expression ]
printf format [, expression-list ] [> expression ]
return [ expression ]
next                                   # skip remaining patterns on this input line.
delete array [ expression ]           # delete an array element.
exit [ expression ]                    # exit immediately; status is expression.
```

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for **$0**. String constants are quoted (" "), with the usual C escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators +, -, *, /, %, ^ (exponentiation), and concatenation (indicated by a blank). The operators ++, --, +=, -=, *=, /=, %=, ^=, **=, >, >=, <, <=, ==, !=, and ?: are also available in expressions. Variables can be scalars, array elements (denoted *x* [*i*]) or fields. Variables are initialized to the null string. Array subscripts can be any string, not necessarily numeric (this allows for a form of associative memory). Multiple subscripts such as [*i* ,*j* ,*k*] are permitted. The constituents are concatenated, separated by the value of **SUBSEP**.

The `print` statement prints its arguments on the standard output (or on a file if >*file* or >>*file* is present or on a pipe if |*cmd* is present), separated by the current output field separator, and terminated by the output record separator. *file* and *cmd* can be literal names or parenthesized expressions. Identical string values in different statements denote the same open file. The `printf` statement formats its expression list according to the format (see *printf*(3)).

Built-In Functions

The built-in function `close`(*expr*) closes the file or pipe *expr*. This function returns zero if successful, otherwise, it returns non-zero.

The customary functions `exp`, `log`, `sqrt`, `sin`, `cos`, `atan2` are built in. Other built-in functions are:

`length(`[*s*]`)`	Length of its associated argument (in characters) taken as a string, or of `$0` if no argument.
`rand()`	Random number on (0,1)
`srand(`[*expr*]`)`	Sets and returns seed for `rand`, and returns the seed set. If no argument is given, the time of day is used as the seed value; otherwise, [*expr*] is used.
`int(`*x*`)`	Truncates to an integer value
`substr(`*s*, *m*[, *n*]`)`	*n*-character substring of *s* that begins at position *m* counted from 1. If *n* is omitted, the substring is limited by the length of string *s*.
`index(`*s*, *t*`)`	Position in *s* where the string *t* occurs, or 0 if it does not.
`match(`*s*, *ere*`)`	Position in *s* where the extended regular expression *ere* occurs, or 0 if it does not. The variables `RSTART` and `RLENGTH` are set to the position and length of the matched string.
`split(`*s*, *a*[, *fs*]`)`	Splits the string *s* into array elements *a*`[1]`, *a*`[2]`, ..., *a*`[`*n*`]`, and returns *n*. The separation is done with the regular expression *fs*, or with the field separator `FS` if *fs* is not given.
`sub(`*ere*, *repl*[, *in*]`)`	Substitutes *repl* for the first occurrence of the extended regular expression *ere* in the string *in*. If *in* is not given, `$0` is used.
`gsub`	ame as `sub` except that all occurrences of the regular expression are replaced; `sub` and `gsub` return the number of replacements.
`sprintf(`*fmt*, *expr*, ... `)`	String resulting from formatting *expr* ... according to the *printf*(3S) format *fmt*
`system(`*cmd*`)`	Executes *cmd* and returns its exit status
`toupper(`*s*`)`	Converts the argument string *s* to uppercase and returns the result.
`tolower(`*s*`)`	Converts the argument string *s* to lowercase and returns the result.

The built-in function `getline` sets `$0` to the next input record from the current input file; `getline <` *file* sets `$0` to the next record from *file*. `getline` *x* sets variable *x* instead. Finally, *cmd* `| getline` pipes the output of *cmd* into `getline`; each call of `getline` returns the next line of output from *cmd*. In all cases, `getline` returns 1 for a successful input, 0 for end of file, and −1 for an error.

Patterns

Patterns are arbitrary Boolean combinations (with `! || &&`) of regular expressions and relational expressions. **awk** supports Extended Regular Expressions as described in *regexp*(5). Isolated regular expressions in a pattern apply to the entire line. Regular expressions can also occur in relational expressions, using the operators `~` and `!~`. `/`*re*`/` is a constant regular expression; any string (constant or variable) can be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern can consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression
expression `in` *array-name*
(*expr,expr,...*) *in* **`array-name`**

where a relop is any of the six relational operators in C, and a matchop is either `~` (matches) or `!~` (does not match). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of the two.

The special patterns `BEGIN` and `END` can be used to capture control before the first input line is read and after the last. `BEGIN` and `END` do not combine with other patterns.

Special Characters

The following special escape sequences are recognized by `awk` in both regular expressions and strings:

Escape	**Meaning**
`\a`	alert character
`\b`	backspace character
`\f`	form-feed character
`\n`	new-line character
`\r`	carriage-return character
`\t`	tab character
`\v`	vertical-tab character
`\`*nnn*	1- to 3-digit octal value *nnn*
`\x`*hhh*	1- to n-digit hexadecimal number

Variable Names

Variable names with special meanings are:

`FS`	Regular expression used to separate fields; also settable by option `-F`*fs*.
`NF`	Number of fields in the current record
`NR`	Ordinal number of the current record
`FNR`	Ordinal number of the current record in the current file
`FILENAME`	Name of the current input file
`RS`	Input record separator (default newline)
`OFS`	Output field separator (default blank)
`ORS`	Output record separator (default newline)
`OFMT`	Output format for numbers (default `%.6g`). If the value of `OFMT` is not a floating-point format specification, the results are unspecified.
`CONVFMT`	Internal conversion format for numbers (default `%.6g`). If the value of `CONVFMT` is not a floating-point format specification, the results are unspecified.
`SUBSEP`	Separates multiple subscripts (default 034)
`ARGC`	Argument count, assignable
`ARGV`	Argument array, assignable; non-null members are taken as filenames
`ENVIRON`	Array of environment variables; subscripts are names. For example, if environment variable `V=thing`, `ENVIRON["V"]` produces `thing`.

Functions can be defined (at the position of a pattern-action statement) as follows:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar, and by reference if array name. Functions can be called recursively. Parameters are local to the function; all other variables are global.

Note that if pattern-action statements are used in an HP-UX command line as an argument to the `awk` command, the pattern-action statement must be enclosed in single quotes to protect it from the shell. For example, to print lines longer than 72 characters, the pattern-action statement as used in a script (`-f` *file* command form) is:

```
length > 72
```

The same pattern action statement used as an argument to the **awk** command is quoted in this manner:

```
awk 'length > 72'
```

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used when evaluating regular expressions and by the relational operators when performing comparisons on string values.

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character-class expressions in regular expressions.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **awk** behaves as if all internationalization variables are set to "C". See *environ*(5).

The `LC_NUMERIC` category determines the radix character used to print floating-point numbers.

LANG defines the search path when looking for commands executed by **system(expr)**.

International Code Set Support

Single- and multi-byte character code sets are supported except that variable names must contain only ASCII characters and regular expressions must contain only single-byte characters.

DIAGNOSTICS

awk supports up to 199 fields (**$1**, **$2**, ..., **$199**) per record.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average:

```
        { s += $1 }"
END     { print "sum is", s, " average is", s/NR }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Simulate **echo** command (see *echo*(1)):

```
BEGIN   {                               # Simulate echo(1)
        for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
        printf "\n"
        exit }
```

SEE ALSO

lex(1), sed(1)
A. V. Aho, B. W. Kernighan, P. J. Weinberger: *The AWK Programming Language*, Addison-Wesley, 1988.

BUGS

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string, concatenate to it.

Scope rules for variables in functions are not well defined.

AUTHOR

awk was developed by AT&T.

STANDARDS CONFORMANCE

awk: SVID2, XPG2, XPG3, POSIX.2

NAME

banner - make posters in large letters

SYNOPSIS

`banner` *strings*

DESCRIPTION

`banner` prints its arguments (each up to 10 characters long) in large letters on the standard output.

Each argument is printed on a separate line. Note that multiple-word arguments must be enclosed in quotes in order to be printed on the same line.

EXAMPLES

Print the message "Good luck Susan" in large letters on the screen:

```
banner "Good luck" Susan
```

The words `Good luck` are displayed on one line, and `Susan` is displayed on a second line.

SEE ALSO

echo(1).

STANDARDS CONFORMANCE

`banner`: SVID2, XPG2, XPG3

NAME

basename, dirname - extract portions of path names

SYNOPSIS

basename *string* [*suffix*]
dirname [*string*]

DESCRIPTION

basename deletes any prefix ending in **/** and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. If *string* consists entirely of slash characters, *string* is set to a single slash character. If there are any trailing slash characters in *string*, they are removed. If the suffix operand is present but not identical to the characters remaining in *string*, but it is identical to a suffix of the characters remaining in *string*, the suffix is removed from *string*. **basename** is normally used inside command substitution marks (\`...\`) within shell procedures.

dirname delivers all but the last level of the path name in *string*. If *string* does not contain a directory component, **dirname** returns **.**, indicating the current working directory.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of string and, in the case of basename, suffix as single and/or multi-byte characters.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **basename** and **dirname** behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

The following shell script, invoked with the argument **/usr/src/cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

```
cc $1
mv a.out 'basename $1 .c'
```

The following example sets the shell variable **NAME** to **/usr/src/cmd**:

```
NAME=`dirname /usr/src/cmd/cat.c`
```

RETURNS

basename and **dirname** return one of the following values:

0	Successful completion.
1	Incorrect number of command-line arguments.

SEE ALSO

expr(1), sh(1).

STANDARDS CONFORMANCE

basename: SVID2, XPG2, XPG3, POSIX.2

dirname: SVID2, XPG2, XPG3, POSIX.2

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

`bc [-c] [-l] [file ...]`

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited-precision arithmetic. It takes input from any files given, then reads the standard input.

Options:

bc recognizes the following command-line options:

`-c`	Compile only. bc is actually a preprocessor for dc which bc invokes automatically (see *dc*(1)). Specifying `-c` prevents invoking *dc*, and sends the *dc* input to standard output.
`-l`	causes an arbitrary-precision math library to be predefined. As a side effect, the scale factor is set.

Program Syntax:

L	a single letter in the range a through z;
E	expression;
S	statement;
R	relational expression.

Comments:

Comments are enclosed in `/*` and `*/`.

Names:

Names include:

simple variables: L
array elements: L [E]
The words `ibase`, `obase`, and `scale`
stacks: L

Other Operands

Other operands include:

Arbitrarily long numbers with optional sign and decimal point.

(E)

sqrt (E)

length (E)	number of significant decimal digits
scale (E)	number of digits right of decimal point

L (E , ... , E)

Strings of ASCII characters enclosed in quotes (").

Arithmetic Operators:

Arithmetic operators yield an E as a result and include:

`+ - * / % ^`	(% is remainder (not mod, see below); ^ is power).
`++ --`	(prefix and append; apply to names)
`= += -= *= /= %= ^=`	

Relational Operators

Relational operators yield an R when used as **E** *op* **E**:

`== <= >= != < >`

Statements

E
{ S ; ... ; S }
if (R) S

while (R) S
for (E ; R ; E) S
null statement
break
quit

Function Definitions:

```
define L ( L ,..., L ) {
        auto L, ... , L
        S; ... S
        return ( E )
}
```

Functions in -l Math Library:

Functions in the `-l` math library include:

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value. Trigonometric angles are in radians where 2 pi radians = 360 degrees.

The value of a statement that is an expression is printed unless the main operator is an assignment. No operators are defined for strings, but the string is printed if it appears in a context where an expression result would be printed. Either semicolons or new-lines can separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(1). Assignments to `ibase` or `obase` set the input and output number radix respectively, again as defined by *dc*(1).

The same letter can be used simultaneously as an array, a function, and a simple variable. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

The `%` operator yields the remainder at the current scale, not the integer modulus. Thus, at scale 1, `7 % 3` is .1 (one tenth), not 1. This is because (at scale 1) `7 / 3` is 2.3 with .1 as the remainder.

EXAMPLES

Define a function to compute an approximate value of the exponential function:

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

Print approximate values of the exponential function of the first ten integers.

```
for(i=1; i<=10; i++) e(i)
```

WARNINGS

There are currently no `&&` (AND) or `||` (OR) comparisons.

The `for` statement must have all three expressions.

`quit` is interpreted when read, not when executed.

`bc`'s parser is not robust in the face of input errors. Some simple expression such as 2+2 helps get it back into phase.

The assignment operators: `=+` `=-` `=*` `=/` `=%` and `=^` are obsolete. Any occurances of these operators cause a syntax error with the exception of `=-` which is interpreted as `=` followed by a unary minus.

Neither entire arrays nor functions can be passed as function parameters.

FILES

`/usr/bin/dc`	desk calculator executable program
`/usr/lib/lib.b`	mathematical library

SEE ALSO

bs(1), dc(1).

bc tutorial in *Number Processing Users Guide*

STANDARDS CONFORMANCE

`bc`: POSIX.2

NAME

bdiff - diff for large files

SYNOPSIS

`bdiff` *file1 file2* [*n*] [`-s`]

DESCRIPTION

`bdiff` compares two files and produces output identical to what would be produced by `diff` (see *diff*(1)), specifying changes that must be made to make the files identical. `bdiff` is designed for handling files that are too large for `diff`, but it can be used on files of any length.

`bdiff` processes files as follows:

- Ignore lines common to the beginning of both files.
- Split the remainder of each file into *n*-line segments, then execute `diff` on corresponding segments. The default value of *n* is 3500.

Command-Line Arguments

`bdiff` recognizes the following command-line arguments:

file1 *file2*	Names of two files to be compared by `bdiff`. If *file1* or *file2* (but not both) is `-`, standard input is used instead.
n	If a numeric value is present as the third argument, the files are divided into *n*-line segments before processing by `diff`. Default value for *n* is 3500. This option is useful when 3500-line segments are too large for processing by `diff`.
`-s`	Silent option suppresses diagnostic printing by `bdiff`, but does not suppress possible error messages from `diff`). If the *n* and `-s` arguments are both used, the *n* argument must precede the `-s` option on the command line or it will not be properly recognized.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`both files standard input (bd2)`
Standard input was specified for both files. Only one file can be specified as standard input.

`non-numeric limit (bd4)`
A non-numeric value was specified for the *n* (third) argument.

EXAMPLES

Find differences between two large files: `file1` and `file2`, and place the result in a new file named `diffs_1.2`.

```
bdiff file1 file2 >diffs_1.2
```

Do the same, but limit file length to 1400 lines; suppress error messages:

```
bdiff file1 file2 1400 -s >diffs_1.2
```

WARNINGS

`bdiff` produces output identical to output from `diff`, and makes the necessary line-number corrections so that the output looks like it was processed by `diff`. However, depending on where the files are split, `bdiff` may or may not find a fully minimized set of file differences.

FILES

`/tmp/bd?????`

SEE ALSO

diff(1).

NAME

bfs - big file scanner

SYNOPSIS

bfs [**-**] *name*

DESCRIPTION

bfs is similar to **ed** except that it is read-only (see *ed*(1)) **bfs** can handle files with up to 32K – 1 lines; each line can contain up to 512 characters, including the new-line character. **bfs** is usually more efficient than **ed** for scanning a file, since the file is not copied to a buffer. Historically, this command was most useful for identifying sections of a large file where **csplit** could be used to divide it into more manageable pieces for editing (see *csplit*(1)). However, most editors now support files larger than the abovementioned limits.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional **-** suppresses printing of sizes. Input is prompted with ***** if **P** and a carriage-return are typed, as in **ed**. Prompting can be turned off again by inputting another **P** and pressing Return. Note that messages are given in response to errors if prompting is turned on.

bfs supports the Basic Regular Expression (RE) syntax (see *regexp*(5)) with the addition that a null RE (e.g., **//**) is equivalent to the last RE encountered. All address expressions described under **ed** are supported. In addition, regular expressions can be surrounded with two symbols besides **/** and **?**: **>** indicates downward search without wrap-around, and **<** indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** can be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **n**, **p**, **q**, **w**, **=**, **!** and null commands operate as described under **ed**. Commands such as **---**, **+++-**, **+++=**, **-12**, and **+4p** are accepted. Note that **1,10p** and **1,10** both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no *remembered* file name. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt**, and **xc** commands, below). The following additional commands are available:

xf *file*
: Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the **xf**. **Xf** commands may be nested to a depth of 10.

xo [*file*]
: Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label*
: This positions a *label* in a command file. *label* is terminated by a new-line, and blanks between the **:** and the start of *label* are ignored. This command can also be used to insert comments into a command file, since labels need not be referenced.

(**.**, **.**)**xb**/*regular expression*/*label*
: A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, **.** is set to the line matched and a jump is made to *label*. This command is the only one that does not issue an error message on bad addresses. Thus it can be used to test whether addresses are bad before other commands are executed. Note that the command

xb/*label*

is an unconditional jump.

The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xn
: List the marks currently in use (marks are set by the **k** command).

xt *number* Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

xv[*digit*][*spaces*][*value*]
The variable name is the specified *digit* following the **xv**. **xv5100** or **xv5 100** both assign the value 100 to the variable 5. **Xv61,100p** assigns the value 1,100p to the variable 6. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables 5 and 6:

```
1,%5p
1,%5
%6
```

all print the first 100 lines.

```
g/%5/p
```

globally searches for the characters **100** and prints each line containing a match. To escape the special meaning of **%**, a **** must precede it. For example, to match and list lines in a program file that contain **printf()** format strings specifying characters, decimal integers, or strings, the following could be used:

```
g/".*\%[cds]/p
```

Another feature of the **xv** command is that the first line of output from an HP-UX command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

each put the current line into variable 5, print it, and increment the variable 6 by one. To escape the special meaning of **!** as the first character of *value*, precede it with a ****.

```
xv7\!date
```

stores the value **!date** into variable 7.

xbz *label*
xbn *label* These two commands test the last saved *return code* from the execution of an HP-UX system command (**!***command*) for a zero or non-zero value, respectively, and cause a branch to the specified label. The two examples below both search for the next five lines containing the string **size**.

First example:

```
xv55
: 1
/size/
xv5!expr %5 - 1
!if [ %5 != 0 ]
xbn 1
```

Second Example:

```
xv45
: 1
/size/
xv4!expr %4 - 1
!if [ %4 = 0 ]
xbz 1
```

xc [*switch*] If *switch* is 1, output from the **p** and null commands is crunched; if *switch* is 0 it isn't. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank, and blank lines suppressed.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions.

`LC_CTYPE` determines the classification of characters as letters, and the characters matched by character class expressions in regular expressions.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang(5))* is used instead of **LANG**. If any internationalization variable contains an invalid setting, **bfs** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

DIAGNOSTICS

`?` for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

SEE ALSO

csplit(1), ed(1), lang(5), regexp(5).

NAME
bifchmod - change mode of a BIF file

SYNOPSIS
`bifchmod` *mode device* **:** *file ...*

DESCRIPTION
`bifchmod` is intended to mimic *chmod*(1).

A BIF file name is recognized by the embedded colon (**:**) delimiter (see *bif*(4) for BIF file naming conventions).

The permissions of each named file are changed according to *mode*, which can be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see *chmod*(2)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others.

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

who is a combination of the letters `u` (for user's permissions), `g` (group), and `o` (other). Specifying `a` is equivalent to `ugo`, which is the default if *who* is omitted.

op can be `+` to add *permission* to the file's mode, `-` to delete *permission*, or `=` to assign *permission* absolutely (all other bits are reset).

permission is any combination of the letters `r` (read), `w` (write), `x` (execute), `s` (set owner or group ID) and `t` (save text - sticky bit); `u`, `g`, or `o` indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with `=` to delete all permissions.

Multiple symbolic modes separated by commas can be given. Operations are performed in the order specified. The letter `s` is useful only with `u` or `g`; `t` works only with `u`.

EXAMPLES
Deny write permission to others:

```
bifchmod o-w file
```

Make a file executable (using symbolic mode):

```
bifchmod +x file
```

Assign read and execute permission to everybody, and set the set-user-id bit:

```
bifchmod 4555 file
```

Assign read and write permission to the file owner, and read permission to everybody else (using absolute mode):

```
bifchmod 644 file
```

Give read, write, and execute permission to the owner and read and execute permissions to everybody else for the BIF file `/etc/script` on `/dev/rdsk/1s0`:

```
bifchmod a=rx,u+w /dev/rdsk/1s0:/etc/script
```

or

```
bifchmod 755 /dev/rdsk/1s0:/etc/script
```

AUTHOR
`bifchmod` was developed by HP.

SEE ALSO
chmod(1), chmod(2), bif(4).

NAME
bifchown, bifchgrp - change file owner or group

SYNOPSIS
bifchown *owner device* **:** *file ...*
bifchgrp *group device* **:** *file ...*

DESCRIPTION
bifchown and **bifchgrp** are intended to mimic *chown*(1) and *chgrp*(1).

A BIF file name is recognized by the embedded colon (**:**) delimiter (see *bif*(4) for BIF file naming conventions).

bifchown changes the owner of the *files* to *owner*. *owner* can be either a decimal user ID or a login name found in the password file.

bifchgrp changes the group ID of the *files* to *group*. *group* can be either a decimal group ID or a group name found in the group file.

EXAMPLES
The examples that follow assume that a BIF directory structure exists on the HP-UX device file **/dev/rdsk/1s0**.

Set the owner of the BIF file **/users/abc/phone.num** to **adm**:

```
bifchown adm /dev/rdsk/1s0:/users/abc/phone.num
```

Set the group ID of the BIF file **/tmp/b.date** to the decimal number **105**:

```
bifchgrp 105 /dev/rdsk/1s0:/tmp/b.date
```

AUTHOR
bifchown was developed by HP.

FILES
/etc/passwd
/etc/group

SEE ALSO
bif(4), chown(1), group(4), passwd(4).

NAME

bifcp - copy to or from BIF files

SYNOPSIS

bifcp *file1 file2*
bifcp *file1* [*file2* ...] *directory*

DESCRIPTION

bifcp is intended to mimic *cp*(1).

A BIF file name is recognized by the embedded colon (**:**) delimiter (see *bif*(4) for BIF file naming conventions).

bifcp copies a BIF or HP-UX file to a BIF or HP-UX file, or list of files (HP-UX or BIF) to a directory. The last name on the argument list is the destination file or directory.

The file name **-** (dash) is interpreted to mean standard input or standard output, depending on its position in the argument list.

RETURNS

bifcp returns exit code 0 if the file is copied successfully. Otherwise it prints a diagnostic and returns non-zero.

EXAMPLES

Copy the HP-UX file **abc** to the BIF file **x/y/z** on HP-UX device **/dev/rdsk/1s0**:

```
bifcp abc /dev/rdsk/1s0:x/y/z
```

Copy BIF file **/backup/log** on **/dev/rdsk/1s0** to HP-UX file **logcopy** in the current directory:

```
bifcp /dev/rdsk/1s0:/backup/log logcopy
```

Copy BIF file **archive** on HP-UX device **/dev/dsk/2s5** to standard output:

```
bifcp /dev/dsk/2s5:archive -
```

The following example copies the BIF files **/a**, **/b**, and **/c** to the HP-UX directory **/users/dave**:

```
sdfcp /dev/rdsk/2s3:/a /dev/rdsk/2s3:/b /dev/rdsk/2s3:/c /users/dave
```

cat(1) can be used to concatenate BIF files using **bifcp** in a shell script:

```
if [ $# -lt 1 ]
then
        echo "Usage: bifcat file ..."
        exit 1
fi
for i in $*
do
        bifcp $i -
done
```

WARNINGS

Note that the media should *not* be mounted before using **bifcp**.

The **-** (standard input/output) notation does not work in some situations.

AUTHOR

bifcp was developed by HP.

SEE ALSO

bif(4), cp(1).

NAME

biffind - find files in a BIF system

SYNOPSIS

`biffind` *path-name-list expression*

DESCRIPTION

`biffind` is intended to mimic `find` (see *find*(1)).

A BIF file name is recognized by the embedded colon (`:`) delimiter (see *bif*(4) for BIF file naming conventions).

`biffind` recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below.

`-name` *pattern*
: True if *pattern* matches the current file name. Pattern can consist of ASCII characters as well as the meta characters:

  `*` match all characters.

  `?` match any character.

  `[ ... ]` match a range of characters.

`-perm` *onum* True if the file permission flags exactly match the octal number *onum*, see *chmod*(1). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

  (flags&onum)==onum

`-type` *c* True if the type of the file is *c*, where *c* is `b`, `c`, `d`, `p`, or `f` for block special file, character special file, directory, fifo (also called a named pipe), or plain file.

`-links` *n* True if the file has *n* links.

`-user` *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the `/etc/passwd` file, it is taken as a user ID.

`-group` *gname*
: True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the `/etc/group` file, it is taken as a group ID.

`-size` *n* True if the file is *n* blocks long.

`-exec` *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon `\;`. A command argument `{ }` is replaced by the current path name.

`-ok` *cmd* Similar to `-exec` except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing `y`.

`-print` Always true; causes the current path name to be printed. This option must be included on the `biffind` command line anytime you want `biffind` to print the path names it has found on the standard output. If `-print` is not specified, `find` locates the files, but fails to tell you about them!

  When `-print` is specified as the only *expression*, `find` prints the absolute path names of all files it finds, beginning at each directory in the *path-name-list*. If `-print` is included as the last component of an *expression*, `find` prints the absolute path names of only those files that satisfy the other primaries in the *expression*.

`-inum` *n* True if the file has inode number *n*.

EXAMPLES

Print the names of all files on the BIF volume `/dev/rdsk/2s0`:

```
biffind /dev/rdsk/2s0: -print
```

Find all files in `/dev/dsk/1s3:/usr/lib` that are directories:

```
    biffind /dev/dsk/1s3:/usr/lib -type d -print
```

Give a long listing of every directory under `/users` on device `/dev/rdsk/2s2`.

```
    biffind /dev/rdsk/2s2:/users -type d -exec bifls -l {} \;
```

AUTHOR

`biffind` was developed by HP.

FILES

`/etc/passwd`
`/etc/group`

SEE ALSO

find(1), bif(4).

NAME

bifls - list contents of BIF directories

SYNOPSIS

`bifls` [**`-AadFilp`**] [*device*:*names* ...]
`bifll` [**`-AadFilp`**] [*device*:*names* ...]

DESCRIPTION

`bifls` is intended to mimic *ls*(1).

A BIF file name is recognized by the embedded colon (**`:`**) delimiter (see *bif*(4) for BIF file naming conventions).

For each directory named, **`bifls`** lists the contents of that directory; for each file named, **`bifls`** repeats its name and any other information requested.

For users with appropriate privileges, **`bifls`** defaults to listing all files except **`.`** (current directory) and **`..`** (parent directory). If the command name **`bifll`** is used, the **`-l`** option is implied.

The following options are recognized by **`bifls`**:

`-a` List all entries; in the absence of this option, entries whose names begin with a period (**`.`**) are *not* listed.

`-A` The same as **`-a`**, except that the current directory **`.`** and parent directory **`..`** are not listed. For users with appropriate privileges, this flag defaults to ON, and is turned off by **`-A`**.

`-d` If argument is a directory, list only its name; often used with **`-l`** to get the status of a directory.

`-F` List with indicator of file type: **`/`** means a directory, **`*`** means executable.

`-i` List the inode of a file or files.

`-l` List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file.

`-p` Do not use **`/etc/passwd`** and **`/etc/group`** to interpret user and group ownership, but rather print out the numeric form.

EXAMPLES

The examples that follow assume that a BIF directory structure exists on the HP-UX device file **`/dev/rdsk/1s0`**.

List all the files in the root directory of the BIF directory structure:

```
bifls -a /dev/rdsk/1s0:
```

Give (in long format) all the information about the BIF directory **`/users/root`** itself (but not the files in the directory):

```
bifls -ld /dev/rdsk/1s0:/users/root
```

WARNINGS

Remember that to obtain a listing of BIF files on **`/dev/rdsk/1s0`**, a colon is required at the end of the device name. In other words, **`bifls /dev/rdsk/1s0`** does not work; use **`bifls /dev/rdsk/1s0:`** instead. If the colon is omitted, **`bifls`** produces a listing of the HP-UX file **`/dev/rdsk/1s0`**, not its BIF contents.

AUTHOR

`bifls` was developed by HP.

FILES

`/etc/passwd`	user IDs
`/etc/group`	group IDs

SEE ALSO

bif(4), ls(1).

NAME

bifmkdir - make a BIF directory

SYNOPSIS

`bifmkdir` *device*`:`*dirname* ...

DESCRIPTION

`bifmkdir` is intended to mimic `mkdir` (see *mkdir*(1)).

A BIF file name is recognized by the embedded colon (`:`) delimiter (see *bif*(4) for BIF file naming conventions).

`bifmkdir` creates specified directories in mode 777. The standard entries, `.` for the directory itself and `..` for its parent, are made automatically.

RETURN VALUE

`bifmkdir` returns exit code 0 if all directories were created successfully; otherwise, it prints a diagnostic and returns non-zero.

EXAMPLES

Create an empty subdirectory named sysmods under the directory `/usr/lib` on HP-UX device `/dev/dsk/2s0`:

```
bifmkdir /dev/dsk/2s0:/usr/lib/sysmods
```

AUTHOR

`bifmkdir` was developed by HP.

SEE ALSO

bif(4), mkdir(1).

NAME

bifrm, bifrmdir - remove BIF files or directories

SYNOPSIS

`bifrm [-fri]` *device*`:`*file* ...

`bifrmdir` *device*`:`*dir* ...

DESCRIPTION

`bifrm` and `bifrmdir` are intended to mimic *rm*(1) and *rmdir*(1).

A BIF file name is recognized by the embedded colon (`:`) delimiter (see *bif*(4) for BIF file naming conventions).

`bifrm` removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed.

If a designated file is a directory, an error comment is printed (unless the optional argument `-r` has been used, see below).

Recognized options are:

`-f`	Remove file with no questions asked, even if the file has no write permission.
`-r`	Recursively delete the entire contents of a directory, then the directory itself. `bifrm` can recursively delete up to 17 levels of directories.
`-i`	Causes `bifrm` to ask whether or not to delete each file. If `-r` is also specified, `bifrm` asks whether to examine each directory encountered.

`bifrmdir` removes entries for the named directories, which must be empty.

EXAMPLES

The following examples assume that a BIF directory structure exists on the HP-UX device file `/dev/rdsk/1s0`.

Recursively comb through the BIF directory `/tmp` and ask if each BIF file should be removed (forced, with no file mode checks):

```
bifrm -irf /dev/rdsk/1s0:/tmp
```

Remove BIF directory `/users/doug`:

```
bifrmdir /dev/rdsk/1s0:/users/doug
```

AUTHOR

`bifrm` was developed by HP.

SEE ALSO

rm(1), rmdir(1), bif(4).

NAME

bs - a compiler/interpreter for modest-sized programs

SYNOPSIS

`bs` [*file* [*args*]]

DESCRIPTION

`bs` is a remote descendant of BASIC and SNOBOL4 with some C language added. `bs` is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the `trace` and `dump` statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written, and vice versa.

If *file* is specified on the command-line, it is used for input before any input is taken from the keyboard. By default, statements read from *file* are compiled for later execution. Likewise, statements entered from the keyboard are normally executed immediately (see `compile` and `execute` below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

`bs` programs are made up of input lines. If the last character on a line is a \, the line is continued. `bs` accepts lines of the following form:

statement
label statement

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A `bs` statement is either an expression or a keyword followed by zero or more expressions. Some keywords (`clear`, `compile`, `!`, `execute`, `include`, `ibase`, `obase`, and `run`) are always executed as they are compiled.

Statement Syntax:

expression The expression is executed for its side effects (value, assignment, or function call). The details of expressions follow the description of statement types below.

`break` `break` exits from the innermost `for/while` loop.

`clear` Clears the symbol table and compiled statements. `clear` is executed immediately.

`compile` [*expression*]
Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A `clear` is associated with this latter case. `compile` is executed immediately.

`continue` `continue` transfers to the loop-continuation of the current `for/while` loop.

`dump` [*name*] The name and current value of every non-local variable is printed. Optionally, only the named variable is reported. After an error or interrupt, the number of the last statement is displayed. The user-function trace is displayed after an error or `stop` that occurred in a function.

`edit` A call is made to the editor selected by the `EDITOR` environment variable if it is present, or *ed*(1) if `EDITOR` is undefined or null. If the *file* argument is present on the command line, *file* is passed to the editor as the file to edit (otherwise no file name is used). Upon exiting the editor, a `compile` statement (and associated `clear`) is executed giving that file name as its argument.

`exit` [*expression*]
Return to system level. The expression is returned as process status.

`execute` Change to immediate execution mode (an interrupt has a similar effect). This statement does not cause stored statements to execute (see `run` below).

`for` *name* = *expression expression statement*
`for` *name* = *expression expression*
...
`next`

`for` *expression* , *expression* , *expression statement*
`for` *expression* , *expression* , *expression*
...
`next`

The `for` statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

`fun` *f* ([*a*, ...]) [v, ...]
...
`nuf`

`fun` defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions cannot be nested. Calling an undefined function is permissible; see function calls below.

`freturn` A way to signal the failure of a user-written function. See the interrogation operator (`?`) below. If interrogation is not present, `freturn` merely returns zero. When interrogation *is* active, `freturn` transfers to that expression (possibly by-passing intermediate function returns).

`goto` *name* Control is passed to the internally stored statement with the matching label.

`ibase` *n* `ibase` sets the input base (radix) to *n*. The only supported values for *n* are the constants `8`, `10` (the default), and `16`. Hexadecimal values 10-15 are entered as `a-f`. A leading digit is required (i.e., `f0a` must be entered as `0f0a`). `ibase` (and `obase` discussed below) are executed immediately.

`if` *expression statement*
`if` *expression*
...
[`else`...]
`fi`

The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings `0` and "" (null) evaluate as zero. In the second form, an optional `else` provides for a second group of statements to be executed when the first group is not. The only statement permitted on the same line with an `else` is an `if`; only other `fi`s can be on the same line with a `fi`. The concatenation of `else` and `if` into an `elif` is supported. Only a single `fi` is required to close an `if` ... `elif` ... [`else` ...] sequence.

`include` *expression*

expression must evaluate to a file name. The file must contain `bs` source statements. Such statements become part of the program being compiled. `include` statements cannot be nested.

`obase` *n* `obase` sets the output base to *n* (see `ibase` above).

`onintr` *label*
`onintr`

`onintr` provides program control of interrupts. In the first form, control passes to the label given, just as if a `goto` had been executed at the time `onintr` was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt causes `bs` to terminate.

`return` [*expression*]

The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

`run` The random number generator is reset. Control is passed to the first internal statement. If the `run` statement is contained in a file, it should be the last statement.

`stop` Execution of internal statements is stopped. `bs` reverts to immediate mode.

`trace` [*expression*]
The `trace` statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns is printed. Each `return` decrements the `trace` *expression* value.

`while` *expression statement*
`while` *expression*
...
`next` `while` is similar to `for` except that only the conditional expression for loop-continuation is given.

`!` *shell command*
An immediate escape to the shell.

`#` ... This statement is ignored (treated as a comment).

Expression Syntax:

name A name is used to specify a variable. Names are composed of a letter (uppercase or lowercase) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open*() below).

name ([expression [, expression] ...])
Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value. If the function is undefined, the call history to the call of that function is printed, and a request for a return value (as an expression) is made. The result of that expression is taken to be the result of the undefined function. This permits debugging programs where not all the functions are yet defined. The value is read from the current input file.

name [expression [, expression] ...]
This syntax is used to reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; `a[1,2]` is the same as `a[1][2]`. The truncated expressions are restricted to values between 0 and 32 767.

number A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an `e` followed by a possibly signed exponent.

string Character strings are delimited by " characters. The `\` escape character allows the double quote (`\"`), new-line (`\n`), carriage return (`\r`), backspace (\b), and tab (`\t`) characters to appear in a string. Otherwise, `\` stands for itself.

(expression) Parentheses are used to alter the normal order of evaluation.

(expression , expression [, expression ...]) [expression]
The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero.
The expression:

```
( False, True )[ a == b ]
```

has the value `True` if the comparison is true.

? expression The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the *Programming Tips* section below), the result of the `eval` built-in function, and for checking the return from user-written functions (see `freturn`). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

- expression	The result is the negation of the expression.
++ name	Increments the value of the variable (or array reference). The result is the new value.
- - name	Decrements the value of the variable. The result is the new value.
!expression	The logical negation of the expression. Watch out for the shell escape command.
expression	*operator* expression Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence):

=	= is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.
_	_ (underscore) is the concatenation operator.
& |	& (logical AND) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical OR) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.
< <= > >= == !=	The relational operators (<: less than, <=: less than or equal, >: greater than, >=: greater than or equal, ==: equal to, !=: not equal to) return one if their arguments are in the specified relation, or return zero otherwise. Relational operators at the same level extend as follows: a>b>c is equivalent to *a*>*b* & *b*>*c*. A string comparison is made if both operands are strings.
+ -	Add and subtract.
* / %	Multiply, divide, and remainder.
^	Exponentiation.

Built-in Functions:

Dealing with arguments

arg(*i*)	is the value of the *i*-th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument (*arg*(0) returns **bs**).
narg( )	returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

abs(*x*)	is the absolute value of *x*.
atan(*x*)	is the arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.
ceil(*x*)	returns the smallest integer not less than *x*.
cos(*x*)	is the cosine of *x* (radians).
exp(*x*)	is the exponential function of *x*.
floor(*x*)	returns the largest integer not greater than *x*.
log(*x*)	is the natural logarithm of *x*.
rand( )	is a uniformly distributed random number between zero and one.
sin(*x*)	is the sine of *x* (radians).
sqrt(*x*)	is the square root of *x*.

String operations

`size(`*s*`)` the size (length in bytes) of *s* is returned.

`format(`*f*`,` *a*`)`
returns the formatted value of *a*. *f* is assumed to be a format specification in the style of *printf*(3S). Only the `%...f`, `%...e`, and `%...s` types are safe. Since it is not always possible to know whether `a` is a number or a string when the `format` call is coded, coercing `a` to the type required by `f` by either adding zero (for `e` or `f` format) or concatenating (_) the null string (for `s` format) should be considered.

`index(`*x*`,` *y*`)` returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

`trans(`*s*`,` *f*`,` *t*`)`
Translates characters of the source *s* from matching characters in *f* to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

`substr(`*s*`,` *start*`,` *width*`)`
returns the sub-string of *s* defined by the *start*ing position and *width*.

`match(`*string*`,` *pattern*`)`

`mstring(`*n*`)` The *pattern* is a regular expression according to the Basic Regular Expression definition (see *regexp*(5)). `mstring` returns the *n*-th (1 <= *n* <= 10) substring of the subject that occurred between pairs of the pattern symbols `\(` and `\)` for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with `^`). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a-z]\)") == 6
mstring(1) == "b"
```

File handling

`open(`*name*`,` *file*`,` *function*`)`
`close(`*name*`)`
name argument must be a `bs` variable name (passed as a string). For the `open`, the *file* argument can be:

1. a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively;
2. a string representing a file name; or
3. a string beginning with an `!` representing a command to be executed (via `sh -c`). The *function* argument must be either `r` (read), `w` (write), `W` (write without new-line), or `a` (append). After a `close`, *name* reverts to being an ordinary variable. If *name* was a pipe, a `wait()` is executed before the close completes (see *wait*(2)). The `bs exit` command does not do such a wait. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

`access(`*s*`,` *m*`)`
executes `access()` (see *access*(2)).

`ftype(`*s*`)` returns a single character file type indication: `f` for regular file, `p` for FIFO (i.e., named pipe), `d` for directory, `b` for block special, or `c` for character special.

Tables

`table(`*name*`,` *size*`)`
A table in `bs` is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a `bs` variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. `bs` prints an error message and stops on table overflow. The result of *table* is

name.

`item(`*name*`,` *i*`)`
`key()`
The `item` function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the `item` function accesses values, the `key` function accesses the "subscript" of the previous `item` call. It fails (or in the absence of an `interrogate` operator, returns null) if there was no valid subscript for the previous `item` call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table; for example:

```
table("t", 100)
    ...
# If word contains "party", the following expression adds one
# to the count of that word:
++t[word]
    ...
# To print out the the key/value pairs:
for i = 0, ?(s = item(t, i)), ++i if key() put = key()_":"_s
```

If the interrogation operator is not used, the result of `item` is null if there are no further elements in the table. Null is, however, a legal "subscript".

`iskey(`*name*`,` *word*`)`
`iskey` tests whether the key *word* exists in the table *name* and returns one for true, zero for false.

Odds and ends

`eval(`*s*`)`
The string argument is evaluated as a `bs` expression. The function is handy for converting numeric strings to numeric internal form. `eval` can also be used as a crude form of indirection, as in:

```
name = "xyz" eval("++"_ name)
```

which increments the variable *xyz*. In addition, `eval` preceded by the interrogation operator permits the user to control `bs` error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named `XXX` (instead of halting the user's program). The following executes a `goto` to the label `L` (if it exists):

```
label="L"
if !(?eval("goto "_ label)) puterr = "no label"
```

`plot(`*request*`,` *args*`)`
If the `tplot` command is available, the `plot` function produces output on devices recognized by `tplot`. The *requests* are as follows:

Call	**Function**
`plot(0,` *term*`)`	causes further *plot* output to be piped into *tplot* with an argument of `-T`*term*. *Term* can be up to 40 characters in length.
`plot(1)`	"erases" the plotter.
`plot(2,` *string*`)`	labels the current point with *string*.
`plot(3,` *x1, y1, x2, y2*`)`	draws the line between (*x1*,*y1*) and (*x2*,*y2*).
`plot(4,` *x, y, r*`)`	draws a circle with center (*x*,*y*) and radius *r*.
`plot(5,` *x1, y1, x2, y2, x3, y3*`)`	draws an arc (counterclockwise) with center (*x1*,*y1*) and endpoints (*x2*,*y2*) and (*x3*,*y3*).
`plot(6)`	is not implemented.
`plot(7,` *x, y*`)`	makes the current point (*x*,*y*).

`plot(8,` *x*, *y*`)`	draws a line from the current point to (*x*,*y*).
`plot(9,` *x*, *y*`)`	draws a point at (*x*,*y*).
`plot(10,` *string*`)`	sets the line mode to *string*.
`plot(11,` *x1*, *y1*, *x2*, *y2*`)`	makes (*x1*,*y1*) the lower left corner of the plotting area and (*x2*,*y2*) the upper right corner of the plotting area.
`plot(12,` *x1*, *y1*, *x2*, *y2*`)`	causes subsequent x (y) coordinates to be multiplied by *x1* (*y1*) and then added to *x2* (*y2*) before they are plotted. The initial scaling is `plot(12, 1.0, 1.0, 0.0, 0.0)`.

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot*.

Each statement executed from the keyboard re-invokes `tplot`, making the results unpredictable if a complete picture is not done in a single operation. Plotting should thus be done either in a function or a complete program, so all the output can be directed to `tplot` in a single stream.

`last()` in immediate mode, `last` returns the most recently computed value.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the characters matched by character class expressions in regular expressions.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, `bs` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Using `bs` as a calculator (`$` is the shell prompt):

```
$ bs
# Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
          ...
# Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4 bal = bal + bal*int
bal - 1000
346.855007
          ...
exit
```

The outline of a typical `bs` program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
          ...
# compute:
while ?(str = read)
          ...
```

```
next
# clean up:
close("read")
                      ...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
# Copy file oldfile to file newfile.
open("read", "oldfile", "r")
open("write", "newfile", "w")
                      ...
while ?(write = read)
                      ...
# close "read" and "write":
close("read")
close("write")

# Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
                      ...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

WARNINGS

The graphics mode (`plot` ...) is not particularly useful unless the `tplot` command is available on your system.

bs is not tolerant of some errors. For example, mistyping a `fun` declaration is difficult to correct because a new definition cannot be made without doing a `clear`. The best solution in such a case is to start by using the `edit` command.

SEE ALSO

ed(1), sh(1), access(2), printf(3S), stdio(3S), lang(5), regexp(5).

See Section (3M) for a further description of the mathematical functions.
`pow()` is used for exponentiation — see *exp*(3M));
bs uses the Standard I/O package.

NAME

cal - print calendar

SYNOPSIS

`cal` [[*month*] *year*]

DESCRIPTION

`cal` prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *year* can be between 1 and 9999. *month* is a decimal number between 1 and 12. The calendar produced is that for England and English colonies.

EXAMPLES

The command:

```
cal 9 1850
```

prints the calendar for September, 1850 on the screen as follows:

```
September 1850
S       M       Tu      W       Th      F       S
1       2       3       4       5       6       7
8       9       10      11      12      13      14
15      16      17      18      19      20      21
22      23      24      25      26      27      28
29      30
```

WARNINGS

The year is always considered to start in January even though this is historically naive.

Beware that `cal 83` refers to the early Christian era, not the 20th century.

STANDARDS CONFORMANCE

`cal`: SVID2, XPG2, XPG3

NAME

calendar - reminder service

SYNOPSIS

`calendar` [-]

DESCRIPTION

`calendar` consults the file `calendar` in the current directory and prints out lines containing today's or tomorrow's date anywhere in the line. On weekends, "tomorrow" extends through Monday.

When a - command-line argument is present, `calendar` searches for the file `calendar` in each user's home directory, and sends any positive results to the user by `mail` (see *mail*(1)). Normally this is done daily in the early morning hours under the control of `cron` (see *cron*(1M)). When invoked by `cron`, `calendar` reads the first line in the `calendar` file to determine the user's environment.

Language-dependent information such as spelling and date format (described below) are determined by the user-specified **LANG** statement in the `calendar` file. This statement should be of the form LANG=*language* where *language* is a valid language name (see *lang*(5)). If this line is not in the `calendar` file, the action described in the EXTERNAL INFLUENCES Environment Variable section is taken.

`calendar` is concerned with two fields: month and day. A month field can be expressed in three different formats: a string representing the name of the month (either fully spelled out or abbreviated), a numeric month, or an asterisk (representing any month). If the month is expressed as a string representing the name of the month, the first character can be either uppercase or lowercase; other characters must be lowercase. The spelling of a month name should match the string returned by calling `langinfo()` (see *langinfo*(3C)). The day field is a numeric value for the day of the month.

Month-Day Formats

If the month field is a string, it can be followed by zero or more blanks. If the month field is numeric, it must be followed by either a slash (/) or a hyphen (-). If the month field is an asterisk (*), it must be followed by a slash (/). The day field can be followed immediately by a blank or non-digit character.

Day-Month Formats

The day field is expressed as a numeral. What follows the day field is determined by the format of the month. If the month field is a string, the day field must be followed by zero or one dot (.) followed by zero or more blanks. If the month field is a numeral, the day field must be followed by either a slash (/) or a hyphen (-). If the month field is an asterisk, the day field must be followed by a slash (/).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings when no **LANG** statement is specified in the `calendar` file.

LANG determines the language in which messages are displayed.

If **LC_TIME** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `calendar` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

The following `calendar` file illustrates several formats recognized by *calendar*:

```
LANG=american
Friday, May 29th: group coffee meeting
meeting with Boss on June 3.
3/30/87 - quarter end review
4-26 Management council meeting at 1:00 pm
It is first of the month ( */1 ); status report due.
```

In the following `calendar` file, dates are expressed according to European English usage:

```
LANG=english
On 20 Jan. code review
```

```
Jim's birthday is on the 3。 February
30/3/87 - quarter end review
26-4 Management council meeting at 1:00 pm
It is first of the month ( 1/* ); status report due.
```

WARNINGS

To get reminder service, either your calendar must be public information or you must run `calendar` from your personal `crontab` file, independent of any `calendar` - run systemwide. Note that if you run `calendar` yourself, the calendar file need not reside in your home directory.

`calendar`'s extended idea of "tomorrow" does not account for holidays.

AUTHOR

`calendar` was developed by AT&T and HP.

FILES

`calendar`	
`/tmp/cal*`	
`/usr/lib/calprog`	to figure out today's and tomorrow's dates
`/usr/lib/crontab`	
`/etc/passwd`	

SEE ALSO

cron(1M), langinfo(3C), mail(1), environ(5).

STANDARDS CONFORMANCE

calendar: SVID2, XPG2, XPG3

NAME

cat - concatenate, copy, and print files

SYNOPSIS

`cat` [`-su`] [`-v` [`-t`] [`-e`]] *file* ...

DESCRIPTION

`cat` reads each *file* in sequence and writes it on the standard output. Thus:

```
cat file
```

prints *file* on the default standard output device;

```
cat file1 file2 >file3
```

concatenates *file1* and *file2,* and places the result in *file3*.

If `-` is appears as a *file* argument, `cat` uses standard input. To combine standard input and other files, use a mixture of `-` and *file* arguments.

Options

`cat` recognizes the following options:

`-s` Silent option. `cat` suppresses error messages about non-existent files, identical input and output, and write errors. Normally, input and output files cannot have identical names unless the file is a special file.

`-u` Do not buffer output (handle character-by-character). Normally, output is buffered.

`-v` Cause non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. Control characters are printed using the form `^`*X* (Ctrl-*X*), and the DEL character (octal 0177) is printed as `^?`. All other non-printing characters are printed as **M-***x,* where *x* is the character specified by the seven low order bits. This option is influenced by the **LANG** environment variable and its corresponding code set.

When the `-v` option is used, the following options are also available:

`-e` Print a `$` character at the end of each line (prior to the new-line). `-v` must be used with the `-e` option or `-e` is ignored.

`-t` Print each tab character as `^I`. `-v` must be used with the `-t` option or `-t` is ignored.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text and filenames as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `cat` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occurred.

EXAMPLES

To create a zero-length file, use any of the following:

```
cat /dev/null >file
cp /dev/null file
touch file
```

SEE ALSO

cp(1), pg(1), pr(1), rmnl(1), ssp(1).

WARNINGS

Command formats such as

overwrites the data in *file1* before the concatenation begins, thus destroying the file. Therefore, be careful when using shell special characters.

STANDARDS CONFORMANCE

`cat`: SVID2, XPG2, XPG3, POSIX.2

NAME

cb - C program beautifier, formatter

SYNOPSIS

`cb` [`-s`] [`-j`] [`-l` *length*] [*file* ...]

DESCRIPTION

`cb` reads C programs, either from *files* or from the standard input, and writes them on the standard output with spacing and indentation that displays the structure of the code. Under default options, `cb` preserves all user new-lines.

Options

`cb` recognizes the following options:

`-s` Converts the code to the canonical style of Kernighan and Ritchie in *The C Programming Language*.

`-j` Causes split lines to be put back together.

`-l length`
Causes `cb` to split lines that are longer than *length*. If the position indicated by *length* is in the middle of an identifier, an operator, a comment, or a string literal, `cb` keeps the entire token on the same line.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of comments and string literals as single and/or multi-byte characters.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting `cb` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

SEE ALSO

cc(1).

The C Programming Language by B. W. Kernighan and D. M. Ritchie.

WARNINGS

Hidden punctuation in preprocessor statements causes indentation errors. An example of hidden punctuation is the curly brace used in a macro definition :

```
#define DO_FOREVER while(1) {
#define END_FOREVER }
```

NAME

cc, c89 - C compiler

SYNOPSIS

`cc` [*options*] *files*
`c89` [*options*] *files*

DESCRIPTION

`cc` is the HP-UX C compiler. `c89` is the HP-UX POSIX-conformant C compiler. Both accept several types of arguments as *files*:

- Arguments whose names end with `.c` are understood to be C source files. Each is compiled and the resulting object file is left in a file having the corresponding basename, but suffixed with `.o` instead of `.c`. However, if a single C file is compiled and linked, all in one step, the `.o` file is deleted.
- Similarly, arguments whose names end with `.s` are understood to be assembly source files and are assembled, producing a `.o` file for each `.s` file.
- Arguments whose names end with `.i` are assumed to be the output of `cpp` (see the `-P` option below). They are compiled without again invoking `cpp` (see *cpp*(1)). Each object file is left in a file having the corresponding basename, but suffixed `.o` instead of `.i`.
- Arguments of the form `-l`*x* cause the linker to search the library `lib`*x*`.sl` or `lib`*x*`.a` in an attempt to resolve currently unresolved external references. Because a library is searched when its name is encountered, placement of a `-l` is significant. If a file contains an unresolved external reference, the library containing the definition must be placed *after* the file on the command line. See *ld*(1) for further details.
- All other arguments, such as those whose names end with `.o` or `.a`, are taken to be relocatable object files that are to be included in the link operation.

Arguments and options can be passed to the compiler through the `CCOPTS` environment variable as well as on the command line. The compiler reads the value of `CCOPTS` and divides these options into two sets; those options which appear before a vertical bar (|), and those options which appear after the vertical bar. The first set of options are placed before any of the command-line parameters to `cc`; the second set of options are placed after the command-line parameters to `cc`. If the vertical bar is not present, all options are placed before the command-line parameters. For example (in *sh*(1) notation),

```
CCOPTS="-v | -lmalloc"
export CCOPTS
cc -g prog.c
```

is equivalent to

```
cc -v -g prog.c -lmalloc
```

When set, the TMPDIR environment variable specifies a directory to be used by the compiler for temporary files, overriding the default directories `/tmp` and `/usr/tmp`.

Options

Note that in the following list, the `cc` and `c89` options `-A` , `-G` , `-g` , `-O` , `-p` , `-v` , `-y` , `+z` , and `+Z` are not supported by the C compiler provided as part of the standard HP-UX operating system. They are supported by the C compiler sold as an optional separate product.

The following option is recognized only by `cc`:

`-A`*mode* Specify the compilation standard to be used by the compiler. *mode* can be one of the following letters:

- `c` Compile in a mode compatible with HP-UX releases prior to 7.0. (See *The C Programming Language*, First Edition by Kernighan and Ritchie). This option is currently the default. The default may change in future releases.
- `a` Compile under ANSI mode (ANSI programming language C standard ANS X3.159-1989). When compiling under ANSI mode, header files define only those names specified by the standard. To get the same name space as in compatibility mode (`-Ac`), define the symbol _HPUX_SOURCE.

The following options are recognized by both `cc` and `c89`:

`-c`	Suppress the link edit phase of the compilation, and force an object (`.o`) file to be produced for each `.c` file even if only one program is compiled. Object files produced from C programs must be linked before being executed.
`-C`	Prevent the preprocessor from stripping C-style comments (see *cpp*(1) for details).
`-D`*name=def* `-D`*name*	Define *name* to the preprocessor, as if by '#define'. See *cpp*(1) for details.
`-E`	Run only `cpp` on the named C or assembly files, and send the result to the standard output.
`-g`	Cause the compiler to generate additional information needed by the symbolic debugger. This option is incompatible with optimization.
`-G`	Prepare object files for profiling with `gprof` (see *gprof*(1)).
`-I`*dir*	Change the algorithm used by the preprocessor for finding include files to also search in directory *dir*. See *cpp*(1) for details.
`-l`*x*	Refer to the fourth bullet item at the beginning of the DESCRIPTION section.
`-L` *dir*	Change the algorithm used by the linker to search for `lib`*x*`.sl` or `lib`*x*`.a`. The `-L` option causes `cc` to search in *dir* before searching in the default locations. See *ld*(1) for details.
`-n`	Cause the output file from the linker to be marked as shareable. For details and system defaults, see *ld*(1).
`-N`	Cause the output file from the linker to be marked as unshareable. For details and system defaults, see *ld*(1).
`-o`*outfile*	Name the output file from the linker *outfile*. The default name is `a.out`.
`-O`	Invoke the optimizer with level 2 optimization. Equivalent to +O2.
`-p`	Arrange for the compiler to produce code that counts the number of times each routine is called. Also, if link editing takes place, replace the standard startoff routine by one that automatically calls `monitor()` at the start (see *monitor*(3C)) and arranges to write out a `mon.out` file at normal termination of execution of the object program. `prof` can then be used to generate an execution profile (see *prof*(1)).
`-P`	Run only `cpp` on the named C files and leave the result on corresponding files suffixed `.i`. The `-P` option is also passed along to `cpp`.
`-q`	Cause the output file from the linker to be marked as demand loadable. For details and system defaults, see *ld*(1).
`-Q`	Cause the output file from the linker to be marked as not demand loadable. For details and system defaults, see *ld*(1).
`-s`	Cause the output of the linker to be stripped of symbol table information. See *strip*(1) for more details. The use of this option prevents the use of a symbolic debugger on the resulting program. See *ld*(1) for more details.
`-S`	Compile the named C files, and leave the assembly language output on corresponding files suffixed `.s`.
`-t`*x*`,`*name*	Substitute subprocess *x* with *name* where *x* is one or more of a set of identifiers indicating the subprocess(es). This option works in two modes: 1) if *x* is a single identifier, *name* represents the full path name of the new subprocess; 2) if *x* is a set of identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path names of the new subprocesses. The *x* can take one or more of the values: `p` Preprocessor (standard suffix is `cpp`)

c	Compiler (standard suffix is ccom)
0	Same as c
a	Assembler (standard suffix is as)
1	Linker (standard suffix is ld)

-U*name* Remove any initial definition of *name* in the preprocessor. See *cpp*(1) for details.

-v Enable verbose mode, which produces a step-by-step description of the compilation process on the standard error.

-w Suppress warning messages.

-W*x***,** *arg1*[**,** *arg2*...]
Pass the argument[s] *argi* to subprocess *x*, where *x* can assume one of the values listed under the **-t** option as well as **d** (driver program). The **-W** option specification allows additional, implementation-specific options to be recognized by the compiler driver. For example,

```
-Wl,-a,archive
```

causes the linker to link with archive libraries instead of with shared libraries. See *ld*(1) for details. For some options, a shorthand notation for this mechanism can be used by placing **+** in front of the option name as in

```
+M
```

which is equivalent to

```
-Wc,-M
```

+M is the Series 300/400 option that causes the compiler to generate calls to the math library instead of generating code for the MC68881 or MC68882 math coprocessor. Options that can be abbreviated using **+** are implementation dependent, and are listed under DEPENDENCIES.

-y Generate additional information needed by static analysis tools, and ensure that the program is linked as required for static analysis. This option is incompatible with optimization.

-Y Enable support of 16-bit characters inside string literals and comments. Note that 8-bit parsing is always supported. See *hpnls*(5) for more details on International Support.

-z Do not bind anything to address zero. This option allows runtime detection of null pointers. See the note on *pointers* below.

-Z Allow dereferencing of null pointers. See the note on *pointers* below. The **-z** and **-Z** are linker options. See *ld*(1) for more details.

+z, +Z Both of these options cause the compiler to generate position independent code (PIC) for use in building shared libraries. The **-G** and **-p** options are ignored if **+z** or **+Z** is used. Normally, **+z** should be used to generate PIC; however, when certain limits are exceeded, **+Z** is required to generate PIC. The **ld** linker issues the error indicating when **+Z** is required. If both **+z** and **+Z** are specified, only the last one encountered applies. For a more complete discussion regarding PIC and these options, see the manual *Programming on HP-UX*.

Any other options encountered generate a warning to standard error.

Other arguments are assumed to be C-compatible object programs, typically produced by an earlier **cc** run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the name **a.out**.

The first edition of *The C Programming Language* by Kernighan and Ritchie and the various addenda to it are intentionally ambiguous in some areas. HP-UX specifies some of these below for compatibility mode (-

Ac) compilations.

`char`	The `char` type is treated as signed by default. It can be declared `unsigned`.
pointers	Accessing the object of a NULL (zero) pointer is technically illegal (see Kernighan and Ritchie), but many systems have permitted it in the past. The following is provided to maximize portability of code. If the hardware is able to return zero for reads of location zero (when accessing at least 8- and 16-bit quantities), it must do so unless the `-z` flag is present. The `-z` flag requests that `SIGSEGV` be generated if an access to location zero is attempted. Writes of location zero may be detected as errors even if reads are not. If the hardware cannot assure that location zero acts as if it was initialized to zero or is locked at zero, the hardware should act as if the `-z` flag is always set.
identifiers	Identifiers are significant up to 255 characters.
types	Certain programs require that a type be a specific number of bits wide. It can be assumed that an `int` can hold at least as much information as a `short`, and that a `long` can hold at least as much information as an `int`. Additionally, either an `int` or a `long` can hold a pointer.

EXTERNAL INFLUENCES

Environment Variables

When the `-Y` option is invoked, `LC_CTYPE` determines the interpretation of string literals and comments as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `cc` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasionally, messages may be produced by the preprocessor, assembler or the link editor.

If any errors occur before `cc` is completed, a non-zero value is returned. Otherwise, zero is returned.

EXAMPLES

The following compiles the C file `prog.c` to create a `prog.o` file, then invokes the `ld` link editor to link `prog.o` and `procedure.o` with all the C startup routines in `/lib/crt0.o` and library routines from the C library `libc.sl` or `libc.a`. The resulting executable program is placed in file `prog`:

```
    cc prog.c procedure.o -o prog
```

WARNINGS

Options not recognized by `cc` are not passed on to the link editor. The option `-W l,`*arg* can be used to pass any such option to the link editor.

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value are to explicitly call `exit()` (see *exit*(2)) or leave the function `main()` with a `return` *expression*`;` construct.

DEPENDENCIES

Series 300/400

Note that in the following list, the `cc` and `c89` options `+e`, `+O`, `+y` are not supported by the C compiler provided as part of the standard HP-UX operating system. They are supported by the C compiler sold as an optional separate product.

The `-z` option is not supported.

The default is to allow null pointer dereferencing; hence using `-Z` has no effect.

The compiler supports the following additional options. The *+opt1* notation can be used as a shorthand notation for some `-W` options.

+bfpa Cause the compiler to generate code that uses the HP98248A or HP98248B floating point accelerator card, if it is installed at run time. If the card is not installed, floating point operations are done on the MC68881 or MC68882 math coprocessor or the MC68040.

+e or **-Wc,-We** Enables HP value added features when compiling in ANSI C mode, **-Aa**. This option is ignored with **-Ac** since these features are already provided. Features enabled:

- $ as an identifier character
- Accept embedded assembly code

+ffpa Cause the compiler to generate code for the HP98248A or HP98248B floating point accelerator card. This code does not run unless the card is installed.

+M Cause the compiler not to generate in-line floating point code for the MC68881, MC68882, or MC68040. Library routines are referenced for **matherr()** capability.

+N*secondary N* Adjust the initial size of internal compiler tables. *secondary* is one of the letters from the set {**abdepstw**}, and *N* is an integer value. *secondary* and *N* are *not* optional. The Series 300/400 compiler automatically expands the tables if they become full. The +N option is supported only for backwards compatibility.

+O*opt* Invoke optimizations selected by *opt*. If *opt* is **1**, only level 1 optimizations are handled. If *opt* is **2**, all optimizations except in-lining are performed. The **-O** option is equivalent to **+O2**. If *opt* is **V**, optimization level 2 is selected, but all global variables and objects dereferenced by global pointers are treated as if they were declared with the keyword **volatile**, meaning that references to the object cannot be optimized away. If *opt* is **3**, all level 2 optimizations are performed and in addition, code for certain functions is generated in-line rather than calling the function. Functions that are in-lined are **strcpy()**, the transcendental functions available on the MC68881 or MC68882 math coprocessor, and certain user-defined functions. For a complete discussion of the various optimization levels, see the *C Programmers Guide*.

+s By default, compilation subprocesses are run concurrently and, in ANSI mode, **cpp** and **ccom** (**cpass1**) are merged into a single subprocess. This results in better compile time performance except when available compilation memory is scarce. Invoking this option executes the processes sequentially and executes **cpp** and **ccom** (**cpass1**) as distinct processes, thereby minimizing memory consumption.

-t*x***,***name* Specify additional subprocess identifiers.

0	First pass of the compiler with level 2 optimization. It is not the same as subprocess **c** (standard suffix is **cpass1** or **cpass1.ansi** if compiling with **-Aa**)
1	Second pass of the compiler with level 2 optimization (standard suffix is **cpass2**)
c	Compiler (standard suffix is **ccom** or **ccom.ansi** if compiling with **-Aa**)
g	Level 2 global optimizer (standard suffix is **c.c1**)
2	Peephole optimizer (standard suffix is **c.c2**)
i	Procedure integrator (standard suffix is **c.c0**)

-v Enables verbose mode in the global optimizer as well.

-W c,-F Perform some function in-lining. The functions that are 'in-lined' are **strcpy()**, and the transcendental functions available on the MC68881 or MC68882 math coprocessor.

-W c,-YE Cause source code lines to be printed on the assembly (**.s**) file as assembly comments, thus showing the correspondence between C source and the resulting assembly code. This option is incompatible with optimization.

-W g,-All Cause the global optimizer to apply all optimizations. By default, the global optimizer does not attempt certain optimizations when the complexity of a function exceeds a certain limit. This option causes the global optimizer to unconditionally apply all optimizations.

+y The default behavior for generating symbolic debugging information (**-g**) and static analysis information (**-y**) is to generate such information only for items referenced in the file being compiled. For example, if a structure is defined in some included header file yet never referenced, no symbolic debugging information or static analysis information is generated for that structure. The **+y** option causes the compiler to generate symbolic debugging information or static analysis information for all items, whether referenced or not. The **+y** option is only valid when used with **-g** or **-y**.

Series 700/800

Note that in the following list of Series 700-and-800-specific **cc** and **c89** options, **-Ae**, **+df**, **+e**, **+f**, **+ES**, **+I**, **-J**, **+m**, **+O**, **+P**, **+pgm**, and **+y** are not supported by the C compiler provided as part of the standard HP-UX operating system. They are supported by the C compiler sold as an optional separate product.

The default is to allow null pointer dereferencing, hence using **-Z** has no effect.

The **-g** option is incompatible with optimization. If both debug and optimization are specified, only the first option encountered takes effect.

The **-y** option is incompatible with optimization. If both static analysis and optimization are specified, only the first option encountered takes effect.

The **-s** option is incompatible with the **-g**, **-G**, **-p**, and **-y** options. If **-s** is specified along with any of the above options, the **-s** option is ignored, regardless of the order in which the options were specified.

Nonsharable, executable files generated with the **-N** option cannot be executed via **exec()** (see *exec*(2)). For details and system defaults, see *ld*(1).

The compiler supports the following additional options. The *+opt1* notation can be used as a shorthand notation for some **-W c** options.

-Wd,-a When processing files which have been written in assembly language, does not assemble with the prefix file which sets up the space and subspace structure required by the linker. Files assembled with this option cannot be linked unless they contain the equivalent information.

-Ae Extended ANSI mode. Same as **-D_HPUX_SOURCE** and **-Aa** and allows the following extensions: **$** characters can appear in identifier names, and enum declarations can include integral type specifiers. Additional extensions may be added to this option in the future.

+DA*architecture* Generate code for the *architecture* specified. *architecture* is required. The default code generated for the Series 800 is **PA_RISC_1.0**. The default code generated for the Series 700 is **PA_RISC_1.1**. The default code generation can be overridden using the **CCOPTS** environment variable or the command line option **+DA**. *architecture* can be either a model number (e.g., **750** for the HP 9000/750 or **870** for the HP 9000/870) or one of the following generic specifications:

- **1.0** Precision Architecture RISC, version 1.0 or higher. This is the default for all Series 800 models.
- **1.1** Precision Architecture RISC, version 1.1. This is the default for all Series 700 models.

The compiler determines the target architecture using the following precedence:

1. Command line specification of **+DA**.
2. Specification of **+DA** in the **CCOPTS** environment variable.
3. The default as mentioned above.

+DS*architecture* Use the instruction scheduler tuned to the *architecture* specified. *architecture* is required. If this option is not used, the compiler uses the instruction scheduler for the architecture on which the program is compiled. The architecture is determined by `uname()` (see *uname*(2)). *architecture* can be either a model number (e.g, `750` for the HP 9000/750 or `870` for the HP 9000/870) or one of the following generic specifications:

`1.0`	Precision Architecture RISC, version 1.0.
`1.1`	Precision Architecture RISC, version 1.1, general scheduling for Series 700 systems.

`+df`*name* Specify profile database file *name* for profile based optimizations. The default is `flow.data` if *name* is not specified. No white space is permitted between `+df` and *name*. Data for more that one application can be kept in the same file. `+df` requires the specification of either `+I` or `+P`. See *ld(1),* `+P`, `+I`, and `+pgm` for more details.

`+e` Enables HP value-added features while compiling in ANSI C mode, **`-Aa`**. This option is ignored with **`-Ac`** because these features are already provided. Features enabled:

- Long pointers
- Integral type specifiers can appear in enum declarations.
- The $ character can appear in identifier names.
- Missing parameters on intrinsic calls

`+ESlit` Place string literals and `const`-qualified data into read-only memory. This may save space in the resulting executable by coalescing identical string literals, and can promote data sharing in a multi-user application.

`+ESsfc` Replace millicode calls with in-line code when performing function pointer comparisons. Care should be taken when using this option and pointers to shared library routines are being compared.

`+f` Inhibit the automatic promotion of float to double when evaluating expressions. This differs from `+r` (see below) in that parameters and function return values *are* promoted. This option is ignored and a warning is produced if ANSI mode is in effect.

`+FP`*string* Specifies how the run time behavior for floating-point operations should be initialized at program start-up. The default is that all behaviors are disabled. See *ld*(1) for specific values of *string*. To dynamically change these settings at run time, refer to *fpgetround*(3M).

`+I` Instrument the application for profile based optimization. See *ld*(1), `+P`, `+df`, and `+pgm` for more details. This option is incompatible with `-G`, `-g`, `+m`, `+o`, `-p`, `-S`, and `-y`.

`-J` Improve run-time performance of standard C routines by altering error condition checking. This option generates in-line assembly for the routines `strcpy()`, `sqrt()`, and `fabs()`, under certain conditions. The `matherr()` function is not called nor is `errno` set on error conditions for the above-mentioned routines (see *matherr*(3M)). This option may also alter the error handling of many routines declared in `<math.h>`. `-J` may in-line or alter the error handling of additional routines in future releases.

`+L` Enable the listing facility and any listing pragmas. A straight listing prints:

- A header on the top of each page
- Line numbers
- The nesting level of each statement
- The postprocessed source file with expanded macros, included files, and no user comments (unless the `-C` option is used).

If the `-Aa` option is used to compile under ANSI C, the listing shows the original source file rather than the postprocessed source file.

`+Lp` Print a listing as described above, but show the postprocessed source file even if one of the ANSI compilation levels is selected. This option is ineffective if the `-y` option is used.

+m Cause the identifier maps to be printed. First, locals by function are listed, then all global identifiers are listed. All other identifiers are then listed by function at the end of the listing. For struct and union members, the address column contains *B*@*b*, where *B* is the byte offset and *b* is the bit offset. Both *B* and *b* are in hexadecimal. This option is incompatible with `+I` and `+P`.

+o Cause the code offsets to be printed in hexadecimal; they are grouped by function at the end of the listing. This option is incompatible with `+I` and `+P`.

+O*opt* Invoke optimizations selected by *opt*. Defined values for *opt* are:

- `0` Perform no optimizations. This is the default.
- `1` Perform optimizations within basic blocks only.
- `2` Perform level 1 and global optimizations. Same as `-O`.
- `3` Perform level 2 as well as interprocedural global optimizations. Also sends `-O` to the linker (see *ld*(1)).
- `E` Same as `-O` but notify the optimizer that floating point traps have been enabled. Prevents the optimizer from performing loop-invariant code motion on floating point operations.
- `m1` Same as `-O` and allow the optimizer to assume no parameters in function calls refer to the same memory.
- `s` Same as `-O` but notify the optimizer to suppress any optimizations which might result in a significant code-size expansion.
- `V` Same as `-O` but assume all global memory references are to be treated as if they were declared with the keyword **volatile**, meaning that references to global objects cannot be optimized away.

+Obb*num* Specify the maximum number of basic blocks allowed in a procedure which is to be optimized at level 2. If the limit is exceeded, a warning is emitted and level 1 optimization is performed for the remainder of the function. The default value for this limit is 500. This option implies `-O`.

+P Optimize the application based on profile data found in the database file **flow.data**, produced by compilation with `+I`. See *ld*(1), `+I`, `+df`, and `+pgm` for more details. This option is incompatible with `-G`, `-g`, `+m`, `+o`, `-p`, `-S`, and `-y`.

+pgm*name* Specify a profile database lookup name within the database file *name*. No white space is permitted between `+pgm` and *name*. `+pgm` requires that either `+I` or `+P` be specified. See also *ld(1)*, `+P`, `+I`, and `+df` for more details.

+r Inhibits the automatic promotion of float to double when evaluating expressions and passing arguments. This option is ignored and a warning produced if the ANSI mode is in effect (see also `+f`).

+R*num* Allow only the first *num* **register** variables to actually have the **register** class. Use this option when the register allocator issues an `out of general registers` message.

+u*num* Allow pointers to access non-natively aligned data. This option alters the way that the compiler accesses dereferenced data. Use of this option may reduce the efficiency of generated code.

- `1` Assume single byte alignment. Dereferences are performed with a series of single-byte loads and stores.
- `2` Dereferences are performed with a series of two-byte loads and stores.
- `4` Dereferences are performed with a series of four-byte loads and stores.

+w*n* Specify the level of the warning messages. The value of *n* can be one of the following values:

- `1` All warnings are issued.

2 Only warnings indicating that code generation might be affected are issued. Equivalent to the compiler default without any `w` opts.
3 No warnings are issued. Equivalent to the `-w` option.

`+Y` Generate static analysis information for all global identifiers not seen in the original source file. This option only has effect if used in conjunction with the `-y` option.

FILES

`file.c`	input file
`file.o`	object file
`a.out`	linked output
`/tmp/ctm*`	default temporary files
`/usr/tmp/ctm*`	default temporary files
`/lib/ccom`	C compiler
`/lib/cpp`	preprocessor
`/lib/cpp.ansi`	preprocessor for ANSI C
`/bin/as`	assembler (see *as*(1))
`/bin/ld`	link editor (see *ld*(1))
`/lib/crt0.o`	runtime startoff
`/lib/mcrt0.o`	startoff for profiling via *prof*(1)
`/lib/gcrt0.o`	startoff for profiling via *gprof*(1)
`/lib/libc.a`	standard C library (archive version), see *HP-UX Reference* Section (3).
`/lib/libc.sl`	standard C library (shared version), see *HP-UX Reference* Section (3).
`/lib/libp/libc.a`	C library for profiled programs (archive version)
`/usr/include`	standard directory for `#include` files
Series 300/400	
`/lib/ccom.ansi`	ANSI C compiler
`/lib/cpass1`	pass 1 of the optimizing compiler
`/lib/cpass1.ansi`	pass 1 of the optimizing ANSI compiler
`/lib/cpass2`	pass 2 of the optimizing compiler
`/lib/c.c0`	procedure in-liner
`/lib/c.c1`	global optimizer
`/lib/c.c2`	peephole optimizer
Series 700/800	
`/lib/icrt0.o`	Startoff for Instrumentation via `+I`
`/usr/lib/nls/$LANG/cc.cat`	C Compiler message catalog
`/usr/lib/uccom`	Stand-alone code generator
`/usr/lib/sched.models`	processor implementation file

SEE ALSO

Program management and analysis tools:

lint(1)	C program checker/verifier
cb(1)	C program beautifier, formatter
cxref(1)	generate C program cross-reference

Profiling and debugging tools:

gprof(1)	display call graph profile data
prof(1)	display profile data
monitor(3C)	prepare execution profile
xdb(1)	C, C++, FORTRAN, and Pascal symbolic debugger
cdb(1)	C, C++, FORTRAN, and Pascal symbolic debugger
adb(1)	absolute debugger

System tools:

as(1)	translate assembly code to machine code
cpp(1)	invoke the the C language preprocessor
ld(1)	invoke the link editor

Miscellaneous:

matherr(3M)	trap math errors
fpgetround(3M)	floating-point mode-control functions
strip(1)	strip symbol and line number information from an object file
crt0(3)	execution startup routine
end(3C)	symbol of the last locations in program
exit(2)	termination of a process

Tutorials and Standards Documents:

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978.

American National Standard for Information Systems - Programming language C, ANS X3.159-1989

STANDARDS CONFORMANCE

`cc`: SVID2, XPG2, XPG3

`c89`: POSIX.2

NAME

cd - change working directory

SYNOPSIS

`cd` [*directory*]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **HOME** is used as the new working directory. If *directory* specifies a complete path starting with **/**, **.**, **. .**, *directory* becomes the new working directory. If neither case applies, `cd` tries to find the designated directory relative to one of the paths specified by the **CDPATH** shell variable. **CDPATH** has the same syntax as, and similar semantics to, the **PATH** shell variable. `cd` must have execute (search) permission in *directory*.

`cd` exists only as a shell built-in command because a new process is created whenever a command is executed, making `cd` useless if written and processed as a normal system command. Moreover, different shells provide different implementations of `cd` as a built-in utility. Features of `cd` as described here may not be supported by all the shells. Refer to individual shell manual entries for differences.

If `cd` is called in a subshell or a separate utility execution environment such as:

find . -type d -exec cd {}; -exec foo {};

(which invokes `foo` on accessible directories)

`cd` does not affect the current directory of the caller's environment. Another usage of `cd` as a stand-alone command is to obtain the exit status of the command.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Change the current working directory to the **HOME** directory from any location in the file system:

```
cd
```

Change to new current working directory `foo` residing in the current directory:

```
cd foo
```

or

```
cd ./foo
```

Change to directory `foobar` residing in the current directory's parent directory:

```
cd ../foobar
```

Change to the directory whose absolute pathname is **`/usr/local/lib/work.files`**:

```
cd /usr/local/lib/work.files
```

Change to the directory **`proj1/schedule/staffing/proposals`** relative to home directory:

```
cd $HOME/proj1/schedule/staffing/proposals
```

VARIABLES

The following environment variables affect the execution of cd:

HOME	The name of the home directory, used when no directory operand is specified.
CDPATH	A colon-separated list of pathnames that refer to directories. If the directory operand does not begin with a slash (**/**) character, and the first component is not dot or dot-dot, `cd` searches for *directory* relative to each directory named in the **CDPATH** variable, in the order listed. The new working directory is set to the first matching directory found. An empty string in place of a directory pathname represents the current directory. If **CDPATH** is not set, it is treated as if it was an empty string.

RETURN VALUE

Upon completion, `cd` exits with one of the following values:

0	The directory was successfully changed.
>0	An error occurred. The working directory remains unchanged.

SEE ALSO

csh(1), pwd(1), ksh(1), sh-posix(1), sh(1), chdir(2).

STANDARDS CONFORMANCE

cd: SVID2, XPG2, XPG3, POSIX.2

NAME

cdb, fdb, pdb - C, C++, FORTRAN, Pascal symbolic debugger

SYNOPSIS

`cdb` [`-d` *dir*] [`-r` *file*] [`-R` *file*] [`-p` *file*] [`-P` *process_ID*] [`-L`] [`-l` *library*] [`-i` *file*] [`-o` *file*] [`-e` *file*] [`-S` *num*] [*objectfile* [*corefile*]]

`fdb` [*cdb options*]

`pdb` [*cdb options*]

DESCRIPTION

`cdb`, `fdb`, and `pdb` are alternate names for a source level debugger for C, C++ HP FORTRAN, and HP Pascal programs. It provides a controlled environment for their execution. Capabilities are similar to **xdb**, but with an older command syntax. The *HP-UX Symbolic Debugger User's Guide* provides a comprehensive description of **xdb**, and includes a section on differences between **xdb** and **cdb**.

objectfile is an executable program file having zero or more of its component modules compiled with the debug option turned on (enabled by the `-g` option of the `cc`, `f77`, `pc`, and `CC` compilers). The support module `/usr/lib/end.o` must be included as the last object file linked, except for libraries included with the `-l` option to `ld` (see *ld*(1)). The support module is included automatically when `ld` is invoked as part of a compile command that uses the `-g` option. The default *objectfile* is `a.out`. Note that by default `ld` links in shared libraries instead of archive libraries.

corefile is a core image from a failed execution of *objectfile*. The default *corefile* is `core`.

Options

`cdb` recognizes the following command-line options:

`-d` *dir*	Specify *dir* as an alternate directory where source files are located.
`-r` *file*	Specify a record *file* which is invoked immediately (for overwrite, not for append).
`-R` *file*	Specify a restore state *file*, which is processed before the `-p` option (if any) and after the `-r` option (if any).
`-p` *file*	Specify a playback *file*, which is invoked immediately.
`-P` *process ID*	Specify the process ID of an existing process that the user wishes to debug.
`-L`	Use the line-oriented interface.
`-l` *library*	Pre-load information about this shared *library*, `-l` **ALL** means always pre-load shared library information.
`-i` *file*	Redirect standard input to the child process from the designated file or character device.
`-o` *file*	Redirect standard output from the child process to the designated file or character device.
`-e` *file*	Redirect standard error from the child process to the designated file or character device.
`-s`	Enable debugging of shared libraries.
`-S` *num*	Set the size of the string cache to *num* bytes (default is 1024, which is also the minimum).

At startup, `cdb` executes commands from the file `.cdbrc` (`.fdbrc` for FORTRAN and `.pdbrc` for Pascal) if it exists in the user's home directory as specified by the environment variable **HOME**.

ENVIRONMENT VARIABLES

Display

`TERM`	Specifies the terminal type. There is no default.
`LINES`	Specifies the window height in lines of text. Default is 24 if not otherwise determinable.
`COLUMNS`	Specifies the window width in text columns. default is 80 if not otherwise determinable.

Command Line Editing

`CDBHIST`	Specifies the history file. Default is `$HOME/.cdbhist.`
`HISTSIZE`	Specifies the actual number of commands allowed in the history file. Default is 128.

`CDBEDIT` This variable specifies the editing mode (`vi`, `emacs`, or `gmacs`). Default is to match the environment variable `VISUAL` or `EDITOR`; otherwise, there is no default.

Native Language Support

LANG Determines the local language equivalent of `y` (for yes/no queries). **LANG** also determines the locale in which messages are displayed. Default is "C".

`LC_CTYPE` Determines the interpretation of text as single- and/or multi-byte characters and their printability when reading or writing character and string data. If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as the default.

International Code Set Support

Single- and multi-byte character code sets are supported.

LOCATION SYNTAX

line A *number* that refers to a particular line in a file.

location A particular *line* in a file and its corresponding address in the user's program. *location* has the following general forms:

> *line*
> **#label**
> *file*[**:***line*]
> [*file***:**]*proc*[**:***proc*[...]][**:***line* | **#***label*]
> [*class*]**::***proc*[**:***line* | **#** *label*]

To reference code addresses symbolically, use:

> *proc***#***line*
> [[*class*]**::**]*proc***#***line*

VARIABLE IDENTIFIERS

Variables are referenced exactly as they are named in your source file(s). Case sensitivity is controlled by the `Z` command.

To determine the value of some variable *var*, various methods can be used, depending on where and what it is:

var Search for *var* first as a local in the current procedure (or the most recent instance of the current procedure), next as a member of that procedure's class, or finally as a global.

*class***::***var* Search *class* for variable.

*proc***.***var*
[[*class*]**::**]*proc***.**[*class***::**]*var*
Search for *var* in the current or most recent instance of *proc*. A leading **::** indicates a global.

*proc***.***depth***.***var*
[[*class*]**::**]*proc***.***depth***.**[*class***::**]*var*
Use the instance of *proc* that is at depth *depth* (exactly), instead of the current or most recent instance.

:*var*
::*var* Search for a global (not local) variable named *var*.

. *dot* is shorthand for the last thing viewed.

SPECIAL VARIABLES

Special variables are names for things that are not normally directly accessible. Special variables include:

$*var* The debugger has room in its own address space for several user-created special variables of type `long`.

`$pc`, `$fp`, `$sp`,
`$d0`, etc. These are the names of the program counter, frame pointer, stack pointer, CPU general registers, etc.

`$result`	This is used to reference the return value from the last command-line procedure call. It can also be referenced as `$short` and `$long`.
`$signal`	This lets you see and modify the current child process signal number.
`$lang`	This lets you see and modify the current language. Possible values are: `C`, `C++`, `FORTRAN`, `Pascal`, `default`.
`$print`	Alters the behavior of the `print` command when printing character data. Values that can be assigned are `ascii`, `native`, and `raw`.
`$line`	This lets you see and modify the current source line number.
`$malloc`	This lets you see the current amount of memory (bytes) allocated at run-time for use by the debugger itself.
`$cBad`	This lets you see and modify the number of machine instructions the debugger will step while in a non-debuggable procedure before setting an up-level breakpoint and free-running to it.
`$pagelines`	This lets you set the number of lines per "page" of debugger output. The prompt `--More--` occurs between pages. Values of zero or less turn off paging.
`$fpa`	If this is set to a non-zero value, any sequence of machine instructions which effectively constitute a single floating-point-accelerator instruction are treated as a single instruction for machine-level single-stepping and display.
`$fpa_reg`	If `$fpa` is set to a non-zero value, `$fpa_reg` indicates which address register is used in floating-point-accelerator instruction sequences. `0` corresponds to register a0, `1` to `a1`, etc. The default value is 2.
`$cplusplus`	This is interpreted as a set of flags to control behavior of certain C++ capabilities.

`bit 0`	Set means print full base class information at each occurrence.
`bit 1`	Set means `bpc` sets breakpoints on member functions of base classes, also.
`bit 2`	Set means `bi` sets breakpoints on member functions of base classes, also.

The default for all bits is clear. Some commands take a `-c` or `-C` argument which causes the action to be as if the appropriate bit of `$cplusplus` was clear (`-c`) or set (`-C`).

COMMANDS

The debugger has a large number of commands for viewing and manipulating the program being debugged. They are explained below and are grouped by functional similarity.

Window Mode Commands

These commands control what is displayed in the source window. The source window by default comes up in source mode for viewing source code. If assembly language instructions are needed, the disassembly mode can be selected. Registers are also shown in this mode. If both assembly instructions and source code are needed, the split-screen mode can be selected. Here are the commands:

`td`	Toggle disassembly mode.
`ts`	Toggle split screen mode.
`gr`	Display the general registers when the debugger is in disassembly (non-split-screen) mode.
`fr`	Display the floating point registers when the debugger is in disassembly (non-split-screen) mode.
`+r`	Scroll the floating-point register display forward four lines.
`-r`	Scroll the floating-point register display back four lines.
`ws` *size*	Set the size of the source viewing window.
`u`	Update the screen to reflect the current location.

`U`	Clear and redraw the screen.

Path Map Commands

Path maps can be used to redirect portions of a source tree to different directories. Here are the commands:

`apm` *old_path* [*new_path*]	Add a path map to the list of path maps.
`lpm`	List path maps. The list is numbered for use with the `dpm` command.
`dpm` [*number* | `*`]	Delete path map. Default *number* is 1 (most recent path map). An `*` deletes all path maps.
`dir` *directory*	Add *directory* to the list of alternate directory search paths for source files.

File Viewing Commands

These commands may change the current viewing position, but they do not affect the next statement to be executed in the child process, if any. Here are the commands:

`e`	Show the current file, procedure, line number, and source line.
`e` *location*	View the source at the specified location.
[*depth*] `E`	Similar to `e`, but sets viewing location to the current location in *proc* on the stack at *depth* (not necessarily the first executable line in the procedure).
`L`	This is a synonym for `0E`.
line	View the source line number *line* in the current file.
[*line*] `p` [*count*]	View one (or *count*) lines starting at the current line (or line number *line*). With the line oriented interface, if multiple lines are printed, the current line is marked with a `=` in the left-most column.
`+` [*lines*]	Move to *lines* (default one) lines after the current line.
`-` [*lines*]	Move to *lines* (default one) lines before the current line.
[*line*] `w` [*size*]	For the line oriented interface, print a window of text containing *size* (default 11) lines centered around the current line (or *line*). The target line is marked with a `=` in the left-most column if multiple lines are printed.
[*line*] `W` [*size*]	Same as `w`, but *size* defaults to 21 lines.
`+w` [*size*] `+W` [*size*]	View a window of text of given or default *size*, beginning at the end of the previous window if the previous command was a window command; otherwise at the current line.
`-w` [*size*] `-W` [*size*]	View a window of text of given or default *size*, ending at the beginning of previous window if the previous command was a window command; otherwise at the current line.
`/` [*string*]	Search forward through the current file for *string*, starting at the line after the current line.
`?` [*string*]	Search backward for *string*, starting with the line before the current line.
`n`	Repeat the previous `/` or `?` command using the same *string* as the last search, starting at the current location being viewed.
`N`	Same as `n`, but the search goes in the opposite direction from that specified by the previous `/` or `?` command.

Display Formats

The display formats tell the debugger's data viewing commands how output should be formatted. A *format* is of the form [*count*]*formchar* [*size*]. For example, `p abc\4x2` prints, starting at the location of `abc`, four two-byte numbers in hexadecimal.

Formats that print numbers use lowercase characters to represent `integer` data types and uppercase to represent `long` data types. For example, `O` prints in `long` octal.

The following formats are available:

n	Print in the "normal" format, based on the type.
(d|D)	Print in decimal (as `integer` or `long`).
(u|U)	Print in unsigned decimal (as `integer` or `long`).
(o|O)	Print in octal (as `integer` or `long`).
(x|X)	Print in hexadecimal (as `integer` or `long`).
(z|Z)	Print in binary (as `integer` or `long`).
(b|B)	Print a byte in decimal (either way).
c	Print a character.
C	Print a wide-character.
(e|E)	Print in "e" floating point notation (as `float`, `double`, or `long double`).
(f|F)	Print in `f` floating point notation (as `float`, `double`, or `long double`).
(g|G)	Print in `g` floating-point notation (as `float`, `double`, or `long double`).
i	Print a disassembled machine instruction.
a	Print a string using *expr* as the address of the first byte.
w	Print a wide-character string using *expr* as the address of the first element.
W	Print a wide-character string using *expr* as the address of a pointer to the first element.
s	Print a string using *expr* as the address of a pointer to the first byte.
t	Show the type of *expr* (usually a variable or procedure name).
T	This is identical to the `t` format except for C++ classes and struct objects where base class and struct type information will also be displayed.
p	Print the name of the procedure containing address *expr*.
r	Print the template of an object (C++).
R	Print the template of an object with base classes displayed (C++).
S	Do a formatted dump of a structure.
k	This is identical to the `S` format.
K	This is identical to the `S` format except for C++ class and struct objects where base class and struct data will also be displayed.

There are some shorthand notations for *size*:

b	1 byte (`char`).
s	2 bytes (`short`).
l	4 bytes (`long`).
D	8 bytes (double). Can only be used with floating-point formats.
L	16 bytes (long double). Can only be used with floating-point formats.

Data Viewing and Modification Commands

expr	If *expr* does not resemble anything else (such as a command), it is handled as if you had typed *expr*`/n` (print expression in normal format), unless followed by `;` or `}`, in which case nothing is printed.
expr / *format*	Print the contents (value) of *expr* using *format*.
expr ? *format*	Print the address of *expr* using *format*.
^[[/]*format*]	Back up to the preceding memory location (based on the size of the last thing displayed).

class`::`	Print the values of all static data members of *class*.
`l` [*proc*[`.`*depth*]] `l` [[[*class*]`::`][*proc*[`.`*depth*]]]	List all parameters and local variables of the current procedure (or of *proc*, if given, at the specified *depth*, if any).
`la`	List all assertions.
`lb`	List all breakpoints.
`ld`	List all directories (where to search for files).
`lsl`	List all shared libraries known to the debugger.
`lz`	List all zignals (signal actions).
`lc` [*string*]	List all (or matching) common blocks in the current procedure (FORTRAN).
`lf` [*string*]	List all (or matching) files (source files used to build *objectfile*).
`lg` [*string*]	List all (or matching) global variables.
`ll` [*string*][`@`*library*]	List all (or matching) labels.
`lm` [*string*]	List all (or matching) macros.
`lp` [[*class*]`::`][*string*]	List all (or matching) procedure names.
`lr` [*string*]	List all (or matching) registers.
`ls` [*string*]	List all (or matching) special variables (except registers).
`lx`	List exception stop-on-throw and -catch state. (C++)
`lcl` [*string*]	List all (or matching) classes. (C++)
`lct` [*string*]	List all (or matching) class templates. (C++)
`ltf` [*string*]	List all (or matching) template functions. (C++)
`lft` [*string*]	List all (or matching) function templates. (C++)
`lo` [[*class*]`::`][*string*]	List all (or matching) overloaded functions. (C++)
`mm` [*string*]	Show a memory map of all currently loaded shared libraries and the main program.

Stack Viewing Commands

[*depth*] `t`	Trace the stack for the first *depth* (default 20) levels.
[*depth*]`T`	The same as `t`, but local variables are also displayed using `/n` format (except that all arrays and pointers are shown as addresses, and structures as first word only).
`up` [*offset*]	Move up (decreasing depth) *offset* levels in the stack. The default value of *offset* is 1.
`down` [*offset*]	Move down (increasing depth) *offset* levels in the stack. The default value of *offset* is 1.
`top`	Move to the top of the stack (this is the same as `0 E`).

Job Control Commands

The parent (debugger) and child (*objectfile*) processes take turns running. The debugger is only active while the child process is stopped due to a signal, includes hitting a breakpoint, or terminated for whatever reason.

Here are the job control commands:

`r` [*arguments*]	Run a new child process with the given or previous argument list, if any.
`R`	Run a new child process with no argument list.
`k`	Terminate (kill) the current child process, if any.
[*count*] `c` [*line*]	Continue after a breakpoint or a signal, ignoring the signal, if any. If *count* is given, the current breakpoint, if any, has its *count* set to that value. If *line* is given, a temporary breakpoint is set at that line number, with a *count* of –1.
[*count*] `C` [*line*]	Continue just like `c`, but allow the signal (if any) to be received. This is fatal to the child process if it does not catch or ignore the signal.
[*count*] `s`	Single step 1 (or *count*) statements.
[*count*] `S`	Single step like `s`, but treat procedure calls as single statements (do not step "into" them).
[*count*] `j`	Single step 1 (or *count*) machine instructions.
[*count*] `J`	Single step like `j`, but treat procedure calls as single instructions (do not step "into" them).

The `s`, `S`, `j`, and `J` commands pass the current signal (like `C`). Set `$signal = 0` if necessary, to prevent this.

Breakpoint Commands

The debugger provides a number of commands for setting and deleting breakpoints. Associated with any breakpoint are three attributes:

location	A particular *line* in a file and its corresponding address in the user's program, if executable code exists for that line.
count	The number of times the breakpoint is encountered prior to recognition. Negative *count*s are temporary and positive *count*s are permanent.
commands	Actions to be taken upon recognition of a breakpoint before waiting for command input. This is a list of debugger commands separated by `;` and enclosed in `{ }`.

Each breakpoint can be individually activated or suspended, and there is an overall breakpoint mode which can be toggled. If any breakpoint is added or activated, or if all breakpoints are suspended, the global mode is toggled automatically.

Here are the breakpoint commands:

`lb`
`B` List all breakpoints.

[*line*] `b` [*commands*]
`b` [*location*] [*commands*]
Set a permanent breakpoint at the specified *line* (in the current procedure) or the specified *location*. Default is the current location.

[*number*] `d` Delete breakpoint *number*, or at the current location.

`D [b]` Delete all breakpoints (including "procedure" breakpoints).

`bi` *expr.proc* [\count] [*commands*]
After evaluating *expr* to what must be a class instance, set an "instance" breakpoint at the first executable line of *proc* for the instance's class.

`bi` [`-c`|`-C`] *expr* [*commands*]
After evaluating *expr* to what must be a class instance, set "instance" breakpoints at the first executable line of all member functions of the instance's class. The `-c` option indicates only members of the designated class. The `-C` option indicates members of base classes as well as members of the designated class.

`bpc` [`-c`|`-C`] *class* [*commands*]
Set "class" breakpoints at the first executable line of all member functions of *class*. See the previous command for information on `-c` and `-C`.

`bpo` [[*class*]`::`]*proc* [*commands*]
Set "overload" breakpoints at the first executable line of all overloaded functions with name *proc* (which can be qualified by a *class*).

`bp` [*commands*]
Set permanent breakpoints at the beginning (first executable line) of every debuggable procedure.

`bpx` [*commands*]
Set permanent breakpoints at the exit (final executable statement) of every debuggable procedure.

`bpt` [*commands*]
Set permanent breakpoints at the entry and exit (first and final executable statement) of every debuggable procedure. The given commands are associated with the entry breakpoint, and default to `Q;2t;c`.

`D p` Delete all "procedure" breakpoints.

`Dpx` Delete all "procedure exit" breakpoints.

`Dpt` Delete all "procedure trace" breakpoints.

`abc` *commands*
Define a global breakpoint command list to be executed whenever any breakpoint is hit (normal, instance, class, overload, procedure, procedure exit, or procedure trace).

`dbc` Delete the global breakpoint command.

For the following commands, if the second character is uppercase, for example, `bU` instead of `bu`, the breakpoint is temporary (*count* is –1), not permanent (*count* is 1).

[*depth*] `bb` [*commands*]
[*depth*] `bB` [*commands*]
: Set a breakpoint at the beginning (first executable line) of the procedure at the given stack *depth*. Default is current procedure.

[*depth*] `bx` [*commands*]
[*depth*] `bX` [*commands*]
: Set a breakpoint at the exit (last executable line) of the procedure at the given stack *depth*. Default is current procedure.

[*depth*] `bu` [*commands*]
[*depth*] `bU` [*commands*]
: Set an up-level breakpoint. Default *depth* is 1.

[*depth*] `bt` [*proc*] [*commands*]
[*depth*] `bT` [*proc*] [*commands*]
: Trace the current procedure (or procedure at *depth*, or *proc*). This command sets breakpoints at both the entrance and exit of a procedure. By default, the entry breakpoint *commands* are `Q;2t;c`, which shows the top two procedures on the stack and continues. The exit breakpoint command list is always `Q;L;c` (print the current location and continue).

address `ba` [*commands*]
address `bA` [*commands*]
: Set a breakpoint at the given code address.

`txc` Toggle the exception stop-on-catch state.

`txt` Toggle the exception stop-on-throw state.

`xcc` [*commands*]
: Define the stop-on-catch command-list.

`xtc` [*commands*]
: Define the stop-on-throw command-list.

`sb` [*num*] Suspend breakpoint number *num*, or at the current location.

`sb *` Suspend all breakpoints.

`ab` [*num*] Activate breakpoint number *num*, or at the current location.

`ab *` Activate all breakpoints.

`tb` Toggle the overall breakpoint mode between active and suspended.

Auxiliary Breakpoint Commands

The following commands are not strictly part of the breakpoint group, but are used almost exclusively in command-list arguments to breakpoints or assertions.

`if` *expr* `{`*commands*`}` [`{`*commands*`}`]
: If *expr* evaluates to a non-zero value, the first group of commands (the first `{}` block) is executed, otherwise it (and the following `{`, if any) is skipped.

`Q` If the Quiet command appears as the first command in a breakpoint command list, the normal announcement of `breakpoint at` *address* is not made.

"*any string you like*"
: Print the given string.

Assertion Control Commands

Assertions are command lists that are executed before every instruction. If there is an active assertion, the program is single stepped at the machine-instruction level and runs very slowly.

Each assertion can be individually activated or suspended, and there is an overall assertions mode which can be toggled. If any assertion is added or activated or if all assertions become suspended, the global mode is toggled automatically.

Here are the assertion control commands:

`a` *commands*	Create a new assertion with the given command list, which is not parsed until it is executed.
expr `a(a` | `d` | `s)`	Modify the assertion numbered *expr:* activate it, delete it, or suspend it.
`A`	Toggle the overall assertions mode between *active* and *suspended*.
`D a`	Delete all assertions.
[*mode*] `x`	Force an exit from assertions *mode* immediately (default or *mode* is non-zero) or at the end of the command list (*mode* non-zero).

Signal Control Commands

These commands are used to modify and list the contents of the "zignal" (signal) handling table. Here are the signal control commands:

`z` [*signal*] [`i`][`r`][`s`][`Q`]	Toggles flags (`i`gnore, `r`eport, or `s`top) for signals (`Q`uietly).
`lz`	Lists the current handling of all signals.

Record and Playback Commands

These commands allow the recording of debugger sessions in a recordfile and the playing back of those sessions. Here are the record and playback commands:

>*file*	Set or change the recordfile to *file* and turn recording on.
>>*file*	This is the same as >*file*, but appends to *file* instead of overwriting.
>@*file* >>@*file*	Set or change the record-all file to *file*, for overwriting or appending.
>(t | f | c)	Turn recording on (`t`) or off (`f`), or close the recording file (`c`).
>@(t | f | c)	Turn record-all on, off, or close the record-all file.
>	Tell the current recording status (same as >>).
>@	Tell the current record-all status (same as >>@).
<*file*	Start playback from *file*.
<<*file*	Start playback from *file*, using the single-step feature of playback.

Save State Command

`ss` *file*

Save the current set of breakpoints, macros and assertions in *file* for later use with the `-R` command-line option.

Macro Definition Commands

`def` *name* [*replacement-text*]	Define *name* as a macro whose value is *replacement-text*.
`undef` *name*	Remove the macro definition from *name* so that *name* no longer exists as a replacement string macro.
`tm`	Toggle the state of the macro substitution mechanism between active and suspended.

Miscellaneous Commands

sm	Suspend the "more" (pagination) facility of the debugger output.
am	Activate the "more" facility to paginate the debugger output.
<carriage-return> ~	Repeat the last command, if possible, with an appropriate increment, if any.
`!` [*command-line*]	Invoke a shell program.
`#` [*text*]	Flag *text* as a comment to be echoed to the command window.
`f` ["*printf-style-format*"]	Set the address printing format using *printf*(3S) format specifications (*not* debugger format

styles). The default is **%10.8lx** .

g (*line* | **#***label* | **+**[*lines*] | **-**[*lines*])
: Go to an address in the procedure on the stack at *depth* zero (not necessarily the current procedure).

h [*topic*]
help [*topic*]
: Print commands/syntaxes related to this *topic* using *more*(1). Use **h help** for a list of topics.

I
: Print information (inquire) about the state of the debugger and various toggles.

M
: Print the current text (*objectfile*) and core (*corefile*) address maps.

tM
: Toggle the address mapping of *corefile* between the initial map and the modifiable mapping pair which the user can set with the **Mc** command.

M(**t** | **c**) [*expr* [**;** *expr* [...]]]
: Set the **t**ext (*objectfile*) or the modifiable **c**ore (*corefile*) address map.

q
: Quit the debugger.

Z
: Toggle case sensitivity in searches.

ADOPTING AN EXISTING PROCESS

The symbolic debugger (**xdb**) command line option **-P** *process_ID* allows for the debugging of a free-running process. To adopt a process, the effective user IDs of the debugger and the process to be adopted must match, or the effective user ID of the debugger must be *root*. When a process is adopted, it halts and the debugger displays where the program is halted, at which point the program can be debugged. If the user quits the debugger without killing the process, the debugger removes all breakpoints from the process and allows it to continue running. If a program is designed to be adopted by the debugger when in a certain state (such as an error condition), it is important that the program do something such as enter an infinite loop, rather than calling **sleep()** (see *sleep*(3C)). A sleeping program cannot be adopted correctly by the debugger, although a suspended process (i.e., blocked on a read) can be.

When using the **-s** option with **xdb** to debug shared libraries in an adopted process, prepare the *executable_file* by executing:

pxdb -s on *executable_file*

Once the file is prepared for debugging, run the *executable_file* in the background and adopt it:

xdb -s -P *process_ID executable_file*

The syntax for this use of the **pxdb** command is:

pxdb -s [**on** | **enable**] *file*
: Enables shared library debugging of the adopted process by setting private data switches within *file*.

pxdb -s [**off** | **disable**] *file*
: Disables shared library debugging of the adopted process by clearing private data switches within *file*.

pxdb -s [**status**] *file*
: Reports whether: shared-library debugging is enabled or disabled, symbolic-debugging information is present, or symbolic-debug information has already been preprocessed. *file* is not changed when the **status** option is specified. If all three conditions are true, an exit value of 0 is returned; otherwise 1.

Note that for the **on** or **off** option, *file* must be writable by the user.

WARNINGS

The debugger does not terminate on an interrupt (**SIGINT**); it jumps to its main loop and awaits another command. However, this does not imply that sending the debugger an interrupt is harmless. It can result in internal tables being left in an inconsistent state that could produce incorrect behavior.

Code that is not compiled debuggable or does not have a corresponding source file is dealt with in a half-hearted manner. The debugger shows **unknown** for unknown file and procedure names, cannot show code locations or interpret parameter lists, etc. However, the linker symbol table provides procedure names for most procedures, even if they are not debuggable.

On some systems, if the debugger is run on a shared *objectfile* you cannot set breakpoints (may only apply if someone else is also executing the program). This may be indicated by the error `Bad access` when you attempt to start a child process. If another user starts running *objectfile* while you are debugging, you may have some interesting interactions.

The debugger will most likely be unusable on systems that have been booted from something other than `/hp-ux` (e.g. `SYSBCKUP` was booted instead).

The debugger has no knowledge about or control over child processes forked in turn by the process being debugged. Programs being debugged should not execute a different program via `exec()` without a `fork()` (see *exec*(2) and *fork*(2)).

Child process output may be (and usually is) buffered. Hence it may not appear immediately after you step through an output statement such as `printf()`. It may not appear at all if you kill the process.

If the *address* given to a `ba` command is not a code address in the child process, strange results or errors may ensue.

Single stepping floating-point instructions may show delayed results for operations that are actually emulated via exception traps (e.g. `fsin` on the Series 300/400 MC68040 processor). Actual results will not be apparent until the next floating-point operation is performed.

Debugging dynamically loaded code is inherently difficult, since no symbols within it are known to the debugger.

If you set the address printing format to something `printf()` doesn't like, you might get an error (usually memory fault) each time you try to print an address, until you fix the format with another `f` command.

Do not use the `z` command to manipulate the `SIGTRAP` signal. This signal is used by the debugger to synchronize with and control the traced process, and unpredictable results may occur if it is otherwise manipulated. A corrolary to this is that applications that make use of the `SIGTRAP` signal will at best be difficult to debug.

If you single step or run with assertions through a call to `longjmp()` (see *setjmp*(3C)), the child process will probably take off free-running as the debugger sets but never hits an up-level breakpoint.

Do not modify any file while the debugger has it open. If you do, the debugger gets confused and may display garbage.

Although the debugger tries to do things reasonably, it is possible to confuse the recording mechanism. Be careful about trying to play back from a file currently open for recording, or vice versa; strange things can happen.

The output of some program generators such as `yacc` have compiler line number directives in them that can confuse the debugger. It expects source line entries in the symbol table to appear in sorted order.

AUTHOR

`cdb` was developed by HP and Third Eye Software.

FILES

`a.out`	Default *objectfile* to debug.
`core`	Default *corefile* to debug.
`/usr/lib/cdb.help`	Text file listed by the `help` command.
`/usr/lib/cdb.help.nro`	Unformatted text file used to generate cdb.help.
`/usr/lib/end.o`	Auxiliary object file (support module) to link with all debuggable programs.
`/usr/lib/nls/$LANG/cdb.cat`	The cdb message catalog.
`/usr/lib/nls/$LANG/pxdb.cat`	The pxdb message catalog.
`$HOME/.cdbrc`	The cdb startup command file.
`$HOME/.fdbrc`	The fdb startup command file.
`$HOME/.pdbrc`	The pdb startup command file.

SEE ALSO

adb(1), cc(1), echo(1), fc(1), ksh(1), ld(1), more(1), pc(1), creat(2), exec(2), fork(2), open(2), ptrace(2), ecvt(3C), multibyte(3C), printf(3S), setjmp(3C), system(3S), a.out(4), core(4), user(4), lang(5), signal(5).

HP-UX Symbolic Debugger (xdb) User's Guide

HP-UX Symbolic Debugger (xdb) Quick Reference

NAME

cdc - change the delta commentary of an SCCS delta

SYNOPSIS

`cdc -r`*SID* [`-m`[*mrlist*]] [`-y`[*comment*]] *files*

DESCRIPTION

`cdc` changes the **delta commentary**, for the *SID* specified by the `-r` option, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the *delta*(1) command (`-m` and `-y` options).

If a directory is named, `cdc` behaves as if each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read (see WARNINGS); each line of the standard input is taken to be the name of an SCCS file to be processed.

Options

Arguments to `cdc`, which can appear in any order, consist of *option* arguments and file names.

All the described *option* arguments apply independently to each named file:

`-r`*SID* Used to specify the *S*CCS *ID*entification (SID) string of a delta for which the delta commentary is to be changed.

`-m`[*mrlist*] If the SCCS file has the `v` option set (see *admin*(1)), a list of MR numbers to be added and/or deleted in the delta commentary of the *SID* specified by the `-r` option *may* be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of *delta*(1). To delete an MR, precede the MR number with the character `!` (see EXAMPLES). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs, is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If `-m` is not used and the standard input is a terminal, the prompt `MRs?` is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The `MRs?` prompt always precedes the `comments?` prompt (see `-y` option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MRs list.

Note that if the `v` option has a value (see *admin*(1)), it is treated as the name of a program (or shell procedure) that validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, `cdc` terminates and the delta commentary remains unchanged.

`-y`[*comment*] Arbitrary text used to replace the *comment* or *comment*s already existing for the delta specified by the `-r` option. Previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If `-y` is not specified and the standard input is a terminal, the prompt `comments?` is issued on the standard output before standard input is read; if standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in the *SCCS* tutorial (see SEE ALSO below). Simply stated, they are either:

- If you made the delta, you can change its delta commentary, or
- If you own the file and directory, you can modify the delta commentary.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

Use *help*(1) for explanations.

EXAMPLES

Add `b178-12345` and `b179-00001` to the MR list, remove `b177-54321` from the MR list, and add the comment `trouble` to delta 1.6 of `s.file`:

```
cdc -r1.6 -m~b178-12345 !b177-54321 b179-00001~ -ytrouble s.file
```

The following does the same thing:

```
cdc -r1.6 s.file
MRs? !b177-54321 b178-12345 b179-00001
comments? trouble
```

WARNINGS

If SCCS file names are supplied to the `cdc` command via the standard input (- on the command line), the `-m` and `-y` options must also be used.

FILES

x-file	(see *delta*(1))
z-file	(see *delta*(1))

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
SCCS User's Guide, in *Programming on HP-UX*. rcsfile(4), acl(5), rcsintro(5).

NAME

cflow - generate C flow graph

SYNOPSIS

cflow [**-r**] [**-ix**] [**-i_**] [**-d***num*] [**-Y_**] files

DESCRIPTION

cflow analyzes a collection of C, YACC, LEX, assembler, and object files, and attempts to build a graph charting the external references. Files suffixed in **.y**, **.l**, **.c**, and **.i** are processed by *yacc*(1), *lex*(1), and *cpp*(1) as appropriate (**.i** files are bypassed), then run through the first pass of *lint*(1). (The **-I**, **-D**, and **-U** options of *cpp*(1) are also understood.) Files suffixed with **.s** are assembled, and information is extracted (as in **.o** files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Next comes the name of the global (normally only a function not defined as an external or beginning with an underscore — see below for the **-i** inclusion option), a colon, and its definition. For information extracted from C source, the definition consists of an abstract type declaration (such as **char ***), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition can be found. For undefined references, only **< >** is printed.

As an example, given the following in **file.c**:

```
int     i;

main()
{
        f();
        g();
        f();
}

f()
{
        i = h();
}
```

the command

```
cflow -ix file.c
```

produces the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the **-e** option of *pr*(1) can be used to compress the tab expansion to something less than every eight spaces.

The following options are interpreted by *cflow*:

-r Reverse the *caller***:***callee* relationship producing an inverted listing showing the callers of each function. The listing is sorted in ascending collation order by callee (see Environment Variables below).

-ix Include external and static data symbols. The default is to include only functions in the flowgraph.

-i_ Include names that begin with an underscore. The default is to exclude these functions (and data if **-ix** is used).

`-dnum` The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a non-positive integer are rejected.

`-Y` Enable support of multi-byte characters inside string literals and comments. Note that 8-bit parsing is always supported. See *hpnls*(5) for more details on international code set support.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating order output by the `-r` option.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, *cflow* behaves as if all internationalization variables are set to "C". See *environ*(5).

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (such as `cpp`).

WARNINGS

Files produced by `lex` and `yacc` cause reordering of line number declarations which can confuse `cflow`. To get proper results, feed `cflow` the `yacc` or `lex` input.

SEE ALSO

as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

STANDARDS CONFORMANCE

`cflow`: SVID2, XPG2, XPG3

NAME

chacl - add, modify, delete, copy, or summarize access control lists (ACLs) of files

SYNOPSIS

`chacl` *acl file ...*
`chacl -r` *acl file ...*
`chacl -d` *aclpatt file ...*
`chacl -f` *fromfile tofile ...*
`chacl -[ z | Z | F ]` *file ...*

DESCRIPTION

`chacl` extends the capabilities of *chmod*(1), by enabling the user to grant or restrict file access to additional specific users and/or groups. Traditional file access permissions, set when a file is created, grant or restrict access to the file's owner, group, and other users. `chacl` enables a user to designate up to thirteen additional sets of permissions (called optional access control list (ACL) entries) which are stored in the access control list of the file.

To use *chacl*, the owner (or superuser) constructs an *acl*, a set of *(user.group, mode)* mappings to associate with one or more files. A specific user and group can be referred to by either name or number; any user (*u*), group (*g*), or both can be referred to with a `%` symbol, representing no specific user or group. The person executing the command can refer to the file's owner or group using the `@` symbol.

Read, write, and execute/search (**rwx**) *modes* are identical to those used by *chmod*; symbolic operators (*op*) add (+), remove (-), or set (=) access rights. The entire *acl* should be quoted if it contains whitespace or special characters. Although two variants for constructing the *acl* are available (and fully explained in *acl*(5)), the following syntax is suggested:

> *entry* [, *entry*] ...

where the syntax for an *entry* is

> *u.g op mode*[*op mode*] ...

By default, `chacl` modifies existing ACLs. It adds ACL entries or modifies access rights in existing ACL entries. If *acl* contains an ACL entry already associated with a file, the entry's mode bits are changed to the new value given, or are modified by the specified operators. If the file's ACL does not already contain the specified entry, that ACL entry is added. `chacl` can also remove all access to files. Giving it a null *acl* argument means either "no access" (when using the `-r` option) or "no changes."

For a summary of the syntax, run `chacl` without arguments.

If *file* is specified as `-`, `chacl` operates on the file open as standard input.

Options

`chacl` recognizes the following options:

`-r` Replace old ACLs with the given ACL. All optional ACL entries are first deleted from the specified files's ACLs, their base permissions are set to zero, and the new ACL is applied. If *acl* does not contain an entry for the owner (*u*`.%`), the group (*%.g*), or the other (`%.%`) users of a file, that base ACL entry's mode is set to zero (no access). The command affects all of the file's ACL entries, but does not change the file's owner or group ID.

In *chmod*(1), the "modify" and "replace" operations are distinguished by the syntax (string or octal value). There is no corollary for ACLs because they have a variable number of entries. Hence `chacl` modifies specific entries by default, and optionally replaces all entries.

`-d` Delete the specified entries from the ACLs on all specified files. The *aclpatt* argument can be an exact ACL or an ACL pattern (see *acl*(5)). `chacl -d` updates each file's ACL only if entries are deleted from it.

If you attempt to delete a base ACL entry from any file, the entry remains but its access mode is set to zero (no access). If you attempt to delete a non-existent ACL entry from a file (that is, if an ACL entry pattern matches no ACL entry), `chacl` informs you of the error, continues, and eventually returns non-zero.

-f *fromfile* Copy the ACL from *fromfile* to the specified *tofile*, transferring ownership, if necessary (see *acl*(5), *chown*(2), or *chownacl*(3C)). *fromfile* can be **-** to represent standard input.

This option implies the **-r** option. If the owner and group of *fromfile* are identical to those of *tofile*, **chacl -f** is identical to:

```
chacl -r 'lsacl fromfile' tofile ...
```

To copy an ACL without transferring ownership, the above command is suggested instead of **chacl -f**.

-z Delete ("zap") all optional entries in the specified file's ACLs, leaving only base entries.

-Z Delete ("zap") all optional entries in the specified file's ACLs, and set the access modes in all base entries to zero (no access). This is identical to replacing the old ACL with a null ACL:

```
chacl -r '' file ...
```

or using *chmod*(1), which deletes optional entries as a side effect:

chmod 0 *file* ...

-F Incorporate ("fold") optional ACL entries into base ACL entries. The base ACL entry's permission bits are altered, if necessary, to reflect the caller's effective access rights to the file; all optional entries, if any, are deleted.

For ordinary users, only the access mode of the owner base ACL entry can be altered. Unlike **getaccess**, the write bit is not turned off for a file on a read-only file system or a shared-text program being executed (see *getaccess*(1)).

For super-users, only the execute mode bit in the owner base ACL entry might be changed, only if the file is not an regular file or if an execute bit is not already set in a base ACL entry mode, but is set in an optional ACL entry mode.

acl also can be obtained from a string in a file:

chacl 'cat file' *files* ...

Using **@** in *acl* to represent "file owner or group" can cause **chacl** to run more slowly because it must reparse the ACL for each file (except with the **-d** option).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **chacl** behaves as if all internationalization variables are set to "C". See *environ*(5).

RETURN VALUE

If **chacl** succeeds, it returns a value of zero.

If **chacl** encounters an error before it changes any file's ACL, it prints an error message to standard error and returns 1. Such errors include invalid invocation, invalid syntax of *acl* (*aclpatt*), a given user name or group name is unknown, or inability to get an ACL from *fromfile* with the **-f** option.

If **chacl** cannot execute the requested operation, it prints an error message to standard error, continues, and later returns 2. This includes cases when a file does not exist, a file's ACL cannot be altered, more ACL entries would result than are allowed, or an attempt is made to delete a non-existing ACL entry.

EXAMPLES

The following command adds read access for user **jpc** in any group, and removes write access for any user in the files's groups, for files **x** and **y**.

```
chacl "jpc.%+r, %.@-w" x y
```

This command replaces the ACL on the file open as standard input and on file **test** with one which only allows the file owner read and write access.

```
chacl -r '(@.%,rw-)' - test
```

Delete from file `myfile` the specific access rights, if any, for user 165 in group 13. Note that this is different from adding an ACL entry that restricts access for that user and group. The user's resulting access rights depend on the entries remaining in the ACL. The command also deletes all entries for user `jpc` that have a read bit turned on:

```
chacl -d '165.13, jpc.*+r' myfile
```

Copy the ACL from `oldfile` to `slow/hare` and `fast/tortoise`.

```
chacl -f oldfile slow/hare fast/tortoise
```

Delete the optional ACL entries, if any, on the file open as standard input.

```
chacl -z -
```

Deny all access to all files in the current directory whose names start with `a`, `b`, or `c`:

```
chacl -Z [a-c]*
```

Turn a file's ACL into a summary (for the caller) in the base entries (file access permission bits):

```
chacl -F fun.stuff
```

WARNINGS

An ACL string cannot contain more than 16 unique entries, even though converting @ symbols to user or group names and combining redundant entries might result in fewer than 16 entries for some files.

DEPENDENCIES

NFS

Only the `-F` option is supported on remote files.

AUTHOR

`chacl` was developed by HP.

SEE ALSO

chmod(1), getaccess(1), lsacl(1), getacl(2), setacl(2), acl(5), glossary(9).

NAME

chatr - change program's internal attributes

SYNOPSIS

`chatr` [`-n`][`-q`][`-s`] *file* ...

DESCRIPTION

`chatr`, by default, prints each *file*'s magic number and file attributes to the standard output.

Options

With one or more optional arguments, `chatr` performs the following operations:

`-n`	Change *file* from demand-loaded to shared.
`-q`	Change *file* from shared to demand-loaded.
`-s`	Perform its operation silently.

The term **shared** applies to the magic number `SHARE_MAGIC` while the term **demand-loaded** applies to the magic number `DEMAND_MAGIC`. Consult *How HP-UX Works: Concepts for the System Administrator* to see how these magic numbers are interpreted.

Upon completion, `chatr` prints the file's old and new values to standard output unless `-s` is specified.

RETURN VALUE

`chatr` returns zero on success. If the command line contents is syntactically incorrect, or one or more of the specified files cannot be acted upon, `chatr` returns the number of files whose attributes could not be modified. If no files are specified, `chatr` returns decimal 255.

DIAGNOSTICS

The error messages produced by `chatr` are self-explanatory.

EXAMPLES

Change `a.out` to demand-loaded

```
chatr -q a.out
```

700/800 only: Change binding mode of program file that uses shared libraries to immediate and nonfatal. Also enable usage of `SHLIB_PATH` environment variable:

```
chatr -B immediate -B nonfatal +s enable a.out
```

700/800 only: Disallow run-time path lookup for the shared library `/lib/libc.sl` that the shared library `libfoo.sl` depends on:

```
chatr +l /lib/libc.sl libfoo.sl
```

DEPENDENCIES

Series 300/400

The following additional option is supported:

`-C` *cachespec* Specify caching mode for the data and/or stack segment. This option has a mandatory argument, *cachespec*, which specifies whether a particular segment uses either write-through caching or copyback caching. *cachespec* consists of one or more of the following arguments, concatenated together:

`d`	Use write-through caching for the data segment.
`D`	Use copyback caching for the data segment.
`s`	Use write-through caching for the stack segment.
`S`	Use copyback caching for the stack segment.

If no option is specified for a particular segment, the caching mode for that segment remains the same as before.

In general, copyback caching should be used because it provides better performance. However, applications that write object code into either the data or stack segment, and then execute the code, do not work with copyback caching, unless the application is modified to flush the cache. Use of `chatr` to change the caching mode to write-through should be restricted to those cases where it would be difficult or impossible to change the source code to use `cachectl()` because simply changing the caching mode does not solve the problem of stale entries in the instruction cache (see *cachectl*(3C)).

Systems that are based on an MC68020 or MC68030 microprocessor do not have a copy-back cache, so the caching mode is always write-through on those systems (i.e., *cachespec* is ignored).

Series 700/800

The following additional options are supported:

`-l` *library* Indicate that the specified shared library is subject to run-time path lookup if directory path lists are provided (see `+s` and `+b`).

`-B` *bind* Select run-time binding behavior of a program using shared libraries. One of the major binding modes `immediate` or `deferred` must be specified. One or more of the binding modifiers `nonfatal` or `restricted` can also be specified, each with a separate option. See the *Programming on HP-UX* manual for a description of binding modes.

`+b` *flag* Control whether the directory path list stored when the program was built can be used to locate shared libraries needed by the program. The two flag values, `enable` and `disable`, respectively enable and disable use of the path list. See the `+s` option.

`+l` *library* Indicate that the specified shared library is not subject to run-time path lookup if directory path lists are provided (see `+s` and `+b`).

`+s` *flag* Control whether the directory path list specified with the `SHLIB_PATH` environment variable can be used to locate shared libraries needed by the program. The two flag values, `enable` and `disable`, respectively enable and disable use of the environment variable. If both `+s` and `+b` are used, their relative order on the command line indicates which path list will be searched first. See the `+b` option.

AUTHOR

`chatr` was developed by HP.

SEE ALSO

ld(1), cachectl(3C), a.out(4), magic(4).

How HP-UX Works: Concepts for the System Administrator.

Programming on HP-UX.

NAME

checknr - check nroff/troff files

SYNOPSIS

`checknr` [`-s`][`-f`][`-a`.*x1*.*y1*.*x2*.*y2* ... .*xn*.*yn*][`-c`.*x1*.*x2*.*x3*...c .*xn*][*file* ...]

DESCRIPTION

`checknr` searches a list of `nroff` or `troff` input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, `checknr` searches the standard input. `checknr` looks for the following:

- Font changes using `\fx` ... `\fP`.
- Size changes using `\sx` ... `\s0`.
- Macros that come in *open* ... *close* forms, such as the `.TS` and `.TE` macros, which must appear in matched pairs.

`checknr` knows about the `ms` and `me` macro packages.

Options

`checknr` recognizes the following options:

`-a` Define additional macro pairs in the list. `-a` is followed by groups of six characters, each group defining a pair of macros. Each six characters consist of a period, the first macro name, another period, and the second macro name. For example, to define the pairs `.BS` and `.ES`, and `.XS` and `.XE`, use:

```
-a.BS.ES.XS.XE
```

No spaces are allowed between the option and its arguments.

`-c` Define commands that `checknr` would otherwise interpret as undefined.

`-f` Ignore `\f`*x* font changes.

`-s` Ignore `\s`*x* size changes.

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

DIAGNOSTICS

`checknr` complains about unmatched delimiters, unrecognized commands, and bad command syntax.

EXAMPLES

Check file `sorting` for errors that involve mismatched opening and closing delimiters and unknown commands, but disregard errors caused by font changes:

```
checknr -f sorting
```

WARNINGS

`checknr` is designed for use on documents prepared with the intent of using `checknr`, much the same as `lint` is used. It expects a certain document writing style for `\f`... and `\s`... commands, in which each `\f`*x* is terminated with `\fP` and each `\s`*x* is terminated with `\s0`. Although text files format properly when the next font or point size is coded directly instead of using `\fP` or `\s0`, such techniques produce complaints from *checknr*. If files are to be examined by *checknr*, the `\fP` and `\s0` delimiting conventions should be used.

`-a` cannot be used to define single-character macro names.

`checknr` does not recognize certain reasonable constructs such as conditionals.

AUTHOR

`checknr` was developed by the University of California, Berkeley.

SEE ALSO

checkeq(1), lint(1), nroff(1).

NAME

chfn - change finger entry

SYNOPSIS

`chfn` [*loginname*]

DESCRIPTION

`chfn` is used to change information about users. The information consists of the user's "real life" name, location, office phone number, and home phone number, and is used by `finger` and other programs (see *finger*(1)). `chfn` prompts the user for each field. Included in the prompt is a default value, which is enclosed in brackets. The default value is accepted simply by pressing the Return key. To enter a blank field, type the word `none`. Below is a sample run:

```
Name [Tracy Simmons]:
Location (Ex: 47U-P5) []:
42L-P1
Office Phone (Ex: 1632) []:
1863
Home Phone (Ex: 9875432) [5551546]:
none
```

`chfn` allows phone numbers to be entered with or without hyphens.

Run `finger` after running `chfn` to make sure the information was processed correctly.

The optional argument *loginname* is used to change another person's `finger` information. This can only be done by users with appropriate privileges.

WARNINGS

Encoding of office and extension information is installation-dependent.

For historical reasons, the user's name, etc., are stored in the `passwd` file. This is an inappropriate place to store the information.

Because two users may try to write the `passwd` file at once, a synchronization method was developed. On rare occasions, `chfn` prints a message that the password file is busy. When this occurs, `chfn` sleeps for a short time, then tries to write to the `passwd` file again.

AUTHOR

`chfn` was developed by the University of California, Berkeley.

FILES

`/etc/passwd`
`/etc/ptmp`

SEE ALSO

finger(1), passwd(4).

NAME

chksnmpd - check connectivity with the SNMP agent

SYNOPSIS

chksnmpd [**-x**] [**-v**]

DESCRIPTION

chksnmpd performs a loop-back, SNMP request to the SNMP agent to get a MIB value. The request is sent to either the **snmpd** or the **snmpd.ea** (see *snmpd*(1M) and *snmpd.ea*(1M)). If **chksnmpd** receives a valid response, it reports a "Success" message; otherwise, it reports a "Failure" message.

If there is a community name entered in **/etc/community**, **chksnmpd** uses that community name; otherwise, **chksnmpd** uses the **public** default community name.

chksnmpd checks connectivity with the local SNMP agent executing on the same system that the **chksnmpd** command is executing. **chksnmpd** cannot check connectivity with an SNMP agent running on a remote system.

Options

chksnmpd recognizes the following options:

-x	Hexdump the packets sent to the SNMP agent and the packets received from the SNMP agent.
-v	Display the results of the response.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages appear. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **chksnmpd** behaves as if all internationalization variables are set to "C." See *environ*(5).

International Code Set Support

Supports single-byte character code sets.

AUTHOR

chksnmpd was developed by HP.

FILES

/etc/snmpd.conf

SEE ALSO

snmpd(1M), snmpd.ea(1M), snmpd.conf(4).

RFC 1155, RFC 1157, RFC 1212, RFC 1213.

NAME

chmod - change file mode

SYNOPSIS

`chmod [-A] [-R]` *mode_list file* ...

Obsolescent form:

`chmod [-A] [-R]` *numeric_mode file* ...

DESCRIPTION

`chmod` changes the permissions of any named *file* according to the symbolic *mode_list* and/or the absolute *numeric_mode*.

A symbolic *mode_list* is a comma-separated list of elements, each of the form:

[*who*] *op* [*permission*] [, ...]

interpreted as follows (see EXAMPLES):

who One or a combination of two or more of the following:

`u`	Modify permissions for user.
`g`	Modify permissions for group.
`o`	Modify permissions for others.
`a`	Modify permissions for all users (`a` is equivalent to `ugo`).

If *who* is not specified, *permission* is modified according to the file mode creation mask that `chmod` inherits (see *umask*(1)); *permission* is changed only if the corresponding bit or bits in the creation mask are clear.

op Must be specifed, and can be one of the following:

`+`	Add *permission* to the existing file mode bits.
`-`	Delete *permission* from the existing file mode bits.
`=`	Set *permission* to specified value (discard existing mode bits).

permission Any combination of the following letters:

- `r` Add or delete read permission for *who*.
- `w` Add or delete write permission for *who*.
- `x` Add or delete execute permission for *who*.
- `s` Set owner or group ID for *who*. Useful only if `u` or `g` is explicitly or implicitly present in *who*.
- `H` Hide a directory (change into a context-dependent file). See *cdf*(4).
- `t` Set the save-text (sticky) bit. Useful only if `u` is explicitly or implicitly present in *who*.
- `X` Conditionally add or delete execute or search permission as follows:
  - If *file* is a directory, add or delete search permission to the existing file mode for *who*.
  - Same as `x` if *file* is a directory, or the current file permissions include execute permission for at least one of user, group, or other.
  - Do nothing if *file* is not a directory and no execute permissions are set in the current *file* mode.
- `u` Copy existing `u` permissions.
- `g` Copy existing `g` permissions.
- `o` Copy existing `o` permissions.

Omitting *permission* is useful only when used with `=` to delete all permissions.

Multiple symbolic modes separated by commas can be given. Operations are performed in the order specified, and can override preceding operations specified in the same command line.

Obsolescent Form:

Absolute permissions can be set by specifing a *numeric_mode*, an octal number constructed from the logical OR of the following mode bits:

Miscellaneous mode bits:

```
 ────────────────── 4000  set user ID on execution (file)
|                      or hide directory (see cdf(4))
|   ─────────────── 2000  set group ID on execution
|  |   ──────────── 1000  sticky bit; see chmod(2)
|  |  |
s  s  t
```

Permission mode bits:

```
 ──────────────────── 0400  read by owner
| ─────────────────── 0200  write by owner
| |  ──────────────── 0100  execute (search in directory) by owner
| | |  ────────────── 0040  read by group
| | | |  ──────────── 0020  write by group
| | | | |  ────────── 0010  execute/search by group
| | | | | |  ──────── 0004  read by others
| | | | | | |  ────── 0002  write by others
| | | | | | | |  ─── 0001  execute/search by others
| | | | | | | | |
r w x r w x r w x
```

Options

-A Preserve any optional access control list (ACL) entries associated with the file. (By default, in conformance with the IEEE Standard POSIX 1003.1-1988, optional ACL entries are deleted.) For information about access control lists, see *acl*(5).

-R Recursively change the file mode bits. For each *file* operand that names a directory, **chmod** alters the file mode bits of the named directory and all files and subdirectories in the file hierarchy below it.

Only the owner of a file (or the user with the appropriate privileges) can change its mode. Only a user having appropriate privileges can set (or retain, if previously set) the sticky bit of a regular file. In order to set the group ID on execution bit, the group of the file must correspond to your current group ID.

When using **chmod** on a symbolic link, the mode of the file referred to by the link is changed.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

Upon completion, **chmod** returns one of the following values:

0	Successful completion.
>0	Error condition occured.

EXAMPLES

Deny write permission to others:

`chmod o-w` *file*

Make a file executable by everybody:

`chmod a+x` *file*

Assign read and execute permission to everybody, and set the set-user-ID bit:

`chmod a=rx,u+s` *file*

Assign read and write permission to the file owner, and read permission to everybody else:

`chmod u=rw,go=r` *file*

or

`chmod 644` *file* (obsolescent form)

Traverse a directory subtree making all regular files readable by user and group only, and all executables and directories executable (searchable) by everyone:

`chmod -R ug+r,o-r,a+X` *pathname*

Note that an inadvertent change of the miscellaneous mode bits of a directory to 4000 or greater, or use of the *permission* letter `H` (hide directory), "hides" a directory (turning it into a context dependent file). Such a directory will quite probably not have your context and therefore will be hidden from you.

If a file seems to be missing from a directory, use `ls -H` to determine the cause. Recovery is accomplished by the addition of a `+` to the *dirname*:

`chmod u-H` *dirname*`+`

WARNINGS

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

DEPENDENCIES

NFS

The `-A` option is not supported for networked files.

HP Clustered Environment

The absolute *numeric_mode* of 4000 also serves to hide a directory. For symbolic *mode_list* elements, the *permission* letter `H` is used to hide a directory (see *cdf*(4)). Root directories cannot be made hidden.

AUTHOR

`chmod` was developed by AT&T and HP.

SEE ALSO

chacl(1), find(1), ls(1), setprivgrp(1M), chmod(2), cdf(4), acl(5).

STANDARDS CONFORMANCE

`chmod`: SVID2, XPG2, XPG3, POSIX.2

NAME

chown, chgrp - change file owner or group

SYNOPSIS

`chown` [`-R`] *owner*[`:`*group*] *file* ...

`chgrp` [`-R`] *group file* ...

DESCRIPTION

`chown` changes the owner ID of one or more *file*s to *owner* and optionally the group ID of one or more *file*s to *group*. The owner can be either a decimal user ID or a login name found in the password file. The group can be either a decimal group ID or a group name found in the group file.

`chgrp` changes the group ID of *file*s to *group*.

In order to change the owner or group, you must own the file or have appropriate privileges. If either command is invoked on a regular file by other than the super-user, the set-user-ID and set-group-ID bits of the file mode (04000 and 02000 respectively) are cleared. Note that a given user's or group's ability to use this command can be restricted by `setprivgrp` (see *setprivgrp*(1M)).

When using `chown` or `chgrp` on symbolic links, the owner or group of the symbolic link is changed.

Access Control Lists (ACLs)

Users can permit or deny specific individuals and groups to access a file by setting optional ACL entries in the file's access control list (see *acl*(5)). When using `chown` in conjunction with ACLs, if the new owner and/or group of a file does not have an optional ACL entry corresponding to *u*`.%` and/or `%.`*g* in the file's access control list, the file's access permission bits remain unchanged. However, if the new owner and/or group is already designated by an optional ACL entry of *u*`.%` and/or `%.`*g* in the file's ACL, `chown` sets the corresponding file access permission bits (and the corresponding base ACL entries) to the permissions contained in that entry.

Options

`-R` Recursively change the owner or group. For each *file* operand that names a directory, the owner or group of the directory and all files and subdirectories in the file hierarchy below it are changed.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`chown` and `chgrp` return the following values:

0	Successful completion.
>0	Error condition occured.

EXAMPLES

The following command changes the owner of the file `jokes` to `sandi`:

```
chown sandi jokes
```

To execute this command, you must be either the owner of `jokes` or a user who has appropriate privileges.

The following command searches the directory `design_notes` and changes each file in that directory to owner `mark` and group `users`:

```
chown -R mark:users design_notes
```

WARNINGS

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

FILES

```
/etc/group
/etc/passwd
```

SEE ALSO

chmod(1), setprivgrp(1M), chown(2), group(4), passwd(4), acl(5).

STANDARDS CONFORMANCE

`chown`: SVID2, XPG2, XPG3, POSIX.2

`chgrp`: SVID2, XPG2, XPG3, POSIX.2

NAME

chsh - change default login shell

SYNOPSIS

chsh *name* [*shell*]

DESCRIPTION

chsh is a command similar to **passwd**, but is used to change the login shell field of the password file rather than the password entry (see *passwd*(1)). *shell*, if given, must be an absolute path name. If the file **/etc/shells** exists, the new login shell must be listed in that file. Otherwise one of the shells listed in the *getusershell*(3C) manual entry can be specified as the shell. If no shell is specified, the shell reverts to the default login shell **/bin/sh**.

NETWORKING FEATURES

NFS

File **/etc/passwd** can be implemented as a Network Information Service database. **chsh** can change **/etc/passwd** on the local system only.

AUTHOR

chsh was developed by HP and the University of California, Berkeley.

FILES

/etc/shells

SEE ALSO

csh(1), ksh(1), pam(1), passwd(1), sh(1), getusershell(3C), passwd(4), shells(4).

NAME

ci - check in RCS revisions

SYNOPSIS

`ci` [*options*] *file* ...

DESCRIPTION

`ci` stores new revisions into RCS files. Each file name ending in `,v` is treated as an RCS file; all others are assumed to be working files. `ci` deposits the contents of each working file into the corresponding RCS file (see *rcsintro*(5)).

If the RCS file does not exist, `ci` creates it and deposits the contents of the working file as the initial revision. The default number is "1.1". The access list is initialized to empty. Instead of the log message, `ci` requests descriptive text (see the `-t` option below).

An RCS file created by `ci` inherits the read and execute permissions from the working file. If the RCS file exists, `ci` preserves its read and execute permissions. `ci` always turns off all write permissions of RCS files.

The caller of the command must have read/write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself. A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. `ci` always creates a new RCS file and unlinks the old one; therefore links to RCS files are useless.

For `ci` to work, the user's login must be in the access list unless the access list is empty, the user is the owner of the file, or the user is super-user.

Normally, `ci` checks whether the revision to be deposited is different from the preceding one. If it is not different, `ci` either aborts the deposit (if `-q` is given) or asks whether to abort (if `-q` is omitted). A deposit can be forced with the `-f` option.

For each revision deposited, `ci` prompts for a log message. The log message should summarize the change and must be terminated with a line containing a single "." or a control-D. If several files are being checked in, `ci` asks whether or not to reuse the log message from the previous file. If the standard input is not a terminal, `ci` suppresses the prompt and uses the same log message for all files (see `-m` option below.

The number of the deposited revision can be given with any of the options `-r`, `-f`, `-k`, `-l`, `-u`, or `-q` (see `-r` option below).

To add a new revision to an existing branch, the head revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file, unless locking is set to `strict` (see *rcs*(1)). A lock held by someone else can be broken with the `rcs` command (see *rcs*(1)).

Options

`-f`[*rev*]	Forces a deposit. The new revision is deposited even if it is not different from the preceding one.
`-k`[*rev*]	Searches the working file for keyword values to determine its revision number, creation date, author, and state (see *co*(1)), and assigns these values to the deposited revision, rather than computing them locally. A revision number given with a command option overrides the number in the working file. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the `-k` option at these sites to preserve its original number, date, author, and state.
`-l`[*rev*]	Works like `-r`, except it performs an additional `co -l` for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the check-in.
`-m"`*msg*`"`	Uses the string *msg* as the log message for all revisions checked in.
`-n"`*name*`"`	Assigns the symbolic name *name* to the checked-in revision. `ci` prints an error message if *name* is already assigned to another number.
`-N"`*name*`"`	Same as `-n`, except that it overrides a previous assignment of *name*.
`-q`[*rev*]	Quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited unless `-f` is given.

`-r`[*rev*] Assigns the revision number *rev* to the checked-in revision, releases the corresponding lock, and deletes the working file. This is the default.

If *rev* is omitted, `ci` derives the new revision number from the caller's last lock. If the caller has locked the head revision of a branch, the new revision is added to the head of that branch and a new revision number is assigned to the new revision. The new revision number is obtained by incrementing the head revision number. If the caller locked a non-head revision, a new branch is started at the locked revision, and the number of the locked revision is incremented. The default initial branch and level numbers are 1. If the caller holds no lock, but is the owner of the file and locking is not set to *strict*, the revision is added to the head of the trunk.

If *rev* indicates a revision number, it must be higher than the latest one on the branch to which *rev* belongs, or must start a new branch.

If *rev* indicates a branch instead of a revision, the new revision is added to the head of that branch. The level number is obtained by incrementing the head revision number of that branch. If *rev* indicates a non-existing branch, that branch is created with the initial revision numbered *rev*`.1`.

NOTE: On the trunk, revisions can be added to the head, but not inserted.

`-s`"*state*" Sets the state of the checked-in revision to the identifier *state*. The default is **Exp**.

`-t`[*txtfile*] Writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, `ci` prompts the user for text from standard input that is terminated with a line containing a single `.` or Ctrl-D. Otherwise, the descriptive text is copied from the file *txtfile*. During initialization, descriptive text is requested even if `-t` is not given. The prompt is suppressed if standard input is not a terminal.

`-u`[*rev*] Similar to `-l`, except that the deposited revision is not locked. This is useful if one wants to process (e.g., compile) the revision immediately after check in.

Access Control Lists (ACLs)

Optional ACL entries should not be added to RCS files, because they might be deleted.

DIAGNOSTICS

For each revision, `ci` prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status always refers to the last file checked in, and is 0 if the operation was successful, 1 if unsuccessful.

EXAMPLES

If the current directory contains a subdirectory `RCS` with an RCS file `io.c,v`, all of the following commands deposit the latest revision from `io.c` into `RCS/io.c,v`:

```
ci io.c
ci RCS/io.c,v
ci io.c,v
ci io.c RCS/io.c,v
ci io.c io.c,v
ci RCS/io.c,v io.c
ci io.c,v io.c
```

WARNINGS

The names of RCS files are generated by appending `,v` to the end of the working file name. If the resulting RCS file name is too long for the file system on which the RCS file should reside, `ci` terminates with an error message.

The log message cannot exceed 2046 bytes.

A file with approximately 240 revisions may cause a hash table overflow. `ci` cannot add another revision to the file until some of the old revisions have been removed. Use the `rcs -o` (obsolete) command option to remove old revisions.

RCS is designed to be used with TEXT files only. Attempting to use RCS with non-text (binary) files results in data corruption.

AUTHOR

`ci` was developed by Walter F. Tichy.

SEE ALSO

co(1), ident(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4), acl(5), rcsintro(5).

NAME

cksum - print file checksum and sizes

SYNOPSIS

`cksum` [*file* ...]

DESCRIPTION

`cksum` calculates and prints to standard output a checksum for each named file, and also prints the number of octets in each file.

`cksum` uses a portable algorithm based on a 32-bit Cyclic Redundancy Check. This algorithm finds a broader spectrum of errors than the 16-bit algorithms used by `sum` (see *sum*(1)). The CRC is the sum of the following expressions, where x is each byte of the file.

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + x^{0}$$

The results of the calculation are truncated to a 32-bit value. The number of bytes in the file is also printed.

Standard input is used if no file names are given.

`cksum` is typically used to verify data integrity when copying files between systems.

EXIT STATUS

Upon completion, `cksum` returns one of the following values:

0	All files were processed succesfully.
>0	One or more files could not be read or some other error occurred.

If an inaccessible file is encountered, `cksum` continues processing any remaining files, but the final exit status is affected.

SEE ALSO

sum(1), wc(1), pdf(4).

STANDARDS CONFORMANCE

`cksum`: POSIX.2

NAME
clear - clear terminal screen

SYNOPSIS
`clear`

DESCRIPTION
`clear` clears the terminal screen if it is possible to do so. It reads the `TERM` environment variable for the terminal type, then reads the appropriate `terminfo` database to determine how to clear the screen.

FILES
`/usr/lib/terminfo/?/*`
terminal database files

AUTHOR
`clear` was developed by the University of California, Berkeley.

SEE ALSO
terminfo(4).

NAME

cmp - compare two files

SYNOPSIS

`cmp [-l] [-s]` *file1 file2*

DESCRIPTION

cmp compares two files (if *file1* or *file2* is `-`, the standard input is used). Under default options, **cmp** makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

cmp recognizes the following options:

`-l` Print the byte number (decimal) and the differing bytes (octal) for each difference (byte numbering begins at 1 rather than 0).

`-s` Print nothing for differing files; return codes only.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **cmp** behaves as if all internationalization variables are set to "C". See *environ(5).*

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

cmp returns the following exit values:

0	Files are identical.
1	Files are not identical.
2	Inaccessible or missing argument.

SEE ALSO

comm(1), diff(1).

STANDARDS CONFORMANCE

cmp: SVID2, XPG2, XPG3, POSIX.2

NAME

cnodes - display information about specified cluster nodes

SYNOPSIS

`cnodes [-almnrsxAC1] [`*name* ...`]`

DESCRIPTION

`cnodes` displays information about the cluster nodes described in the `/etc/clusterconf` file for clustered systems. When no argument is given, all currently active cluster nodes in the cluster are listed. When one or more *name*s are specified, information about each named cluster node is displayed.

There are two major listing formats. The format chosen depends on whether the output is going to a terminal, and can also be controlled by command-line options. The default format for a terminal is to list cluster node names in multi-column format. If the standard output is not a terminal, the default format is to list one cluster node per line.

In order to determine output formats for multi-column output, `cnodes` uses the environment variable `COLUMNS` to determine the number of character positions available on the output line. If this variable is not set, the `terminfo` database is used to determine the number of columns, based on the environment variable `TERM` (see *terminfo*(4)). If this information cannot be obtained, 80 columns is assumed.

Options

`cnodes` recognizes the following options:

- `-a` List all specified cluster nodes, whether or not the nodes are clustered. If no *name* is specified, `cnodes` lists all entries in the `/etc/clusterconf` file. Cluster nodes not currently clustered are displayed with an asterisk (*) following the cluster node name.
- `-l` (ell) List cluster nodes in long format, giving cluster node name, cnode ID, the cnode's swap server, and whether or not the cnode is a cluster server. If the output is going to a terminal, a header is also displayed.
- `-m` List information about the local cluster node only. If `-m` is specified, any *name* arguments are ignored.
- `-n` List cluster node IDs instead of cluster node names.
- `-r` Display information about the cluster server only. If `-r` is specified, any *name* arguments are ignored.
- `-s` Suppress all output. This option is useful for testing the exit value.
- `-x` Do not display information about the local cluster node.
- `-A` Same as `-a`, except cluster nodes currently not clustered do not have an asterisk appended to their names.
- `-C` Force multi-column output.
- `-1` (one) Force single column output.

EXTERNAL INFLUENCES

Environment Variables

`LANG` determines the language in which messages are displayed.

If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `cnodes` behaves as if all internationalization variables are set to "C". See *environ*(5).

RETURN VALUE

`cnodes` exits with a value of `0` if the local machine is a member of a cluster; `1` if not.

DIAGNOSTICS

`cnodes` writes `Machine is not clustered` to the standard error output if the specified machine is not currently a member of a cluster and neither the `-a` nor the `-s` option was specified.

`Machine does not exist` is written to the standard error output if the specified machine is not listed in `/etc/clusterconf` and `-s` was not specified.

AUTHOR

`cnodes` was developed by HP.

FILES

`/etc/clusterconf`

SEE ALSO

ccck(1M), cnodeid(2), cnodes(2), terminfo(4), clusterconf(4).

NAME

co - check out RCS revisions

SYNOPSIS

`co` [*options*] *file* ...

DESCRIPTION

`co` retrieves revisions from RCS files. Each file name ending in `,v` is taken to be an RCS file. All other files are assumed to be working files. `co` retrieves a revision from each RCS file and stores it in the corresponding working file (see also *rcsintro*(5)).

Revisions of an RCS file can be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later checked in must normally be locked. Locking a revision currently locked by another user fails (a lock can be broken with the `rcs` command, but poses inherent risks when independent changes are being made simultaneously (see *rcs*(1)). `co` with locking requires the caller to be on the access list of the RCS file unless: he is the owner of the file, a user with appropriate privileges, or the access list is empty. `co` without locking is not subject to access list restrictions.

A revision is selected by number, check-in date/time, author, or state. If none of these options are specified, the latest revision on the trunk is retrieved. When the options are applied in combination, the latest revision that satisfies all of them is retrieved. The options for date/time, author, and state retrieve a revision on the selected branch. The selected branch is either derived from the revision number (if given), or is the highest branch on the trunk. A revision number can be attached to the options `-l`, `-p`, `-q`, or `-r`.

The caller of the command must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory that contains the RCS file.

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless the file is checked out unlocked and locking is set to `strict` (see *rcs*(1)).

If a file with the name of the working file exists already and has write permission, `co` aborts the check out if `-q` is given, or asks whether to abort if `-q` is not given. If the existing working file is not writable, it is deleted before the check out.

A number of temporary files are created. A semaphore file is created in the directory of the RCS file to prevent simultaneous update.

A `co` command applied to an RCS file with no revisions creates a zero-length file. `co` always performs keyword substitution (see below).

Options

`-l`[*rev*] Locks the checked out revision for the caller. If omitted, the checked out revision is not locked. See option `-r` for handling of the revision number *rev*.

`-p`[*rev*] Prints the retrieved revision on the standard output rather than storing it in the working file. This option is useful when `co` is part of a pipe.

`-q`[*rev*] Quiet mode; diagnostics are not printed.

`-d`*date* Retrieves the latest revision on the selected branch whose check in date/time is less than or equal to *date*. The date and time may be given in free format and are converted to local time. Examples of formats for *date*:

Tue-PDT, 1981, 4pm Jul 21	(free format)
Fri April 16 15:52:25 EST 1982	(output of *ctime*(3C))
86/4/21 10:30am	(format: yy/mm/dd hh:mm:ss)

Most fields in the date and time can be defaulted. `co` determines the defaults in the order year, month, day, hour, minute, and second (from most- to least-significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date `20, 10:30` defaults to 10:30:00 of the 20th of the current month and current year. Date/time fields can be delimited by spaces or commas. If spaces are used, the string must be surrounded by double quotes.

-r[*rev*] Retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. *rev* is composed of one or more numeric or symbolic fields separated by `.`. The numeric equivalent of a symbolic field is specified with the `ci -n` and `rcs -n` commands (see *ci*(1) and *rcs*(1)).

-s*state* Retrieves the latest revision on the selected branch whose state is set to *state*.

-w[*login*]
Retrieves the latest revision on the selected branch that was checked in by the user with login name *login*. If the argument *login* is omitted, the caller's login is assumed.

-j*joinlist* Generates a new revision that is the result of the joining of the revisions on *joinlist*. *joinlist* is a comma-separated list of pairs of the form rev2:rev3, where *rev2* and *rev3* are (symbolic or numeric) revision numbers. For the initial pair, *rev1* denotes the revision selected by the options `-l`, ..., `-w`. For all other pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, `co` joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is particularly useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1* < *rev2* < *rev3* on the same branch, joining generates a new revision that is similar to *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap with changes from *rev2* to *rev3*, `co` prints a warning and includes the overlapping sections, delimited as follows:

```
<<<<<<<
rev1
=======
rev3
>>>>>>>
```

For the initial pair, *rev2* can be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. If the `-l` option is present, the initial *rev1* is locked.

Keyword Substitution

Strings of the form *$keyword$* and *$keyword:...$* embedded in the text are replaced with strings of the form *$keyword: value $*, where *keyword* and *value* are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form *$keyword$*. On check out, `co` replaces these strings with strings of the form *$keyword: value $*. If a revision containing strings of the latter form is checked back in, the value fields are replaced during the next checkout. Thus, the keyword values are automatically updated on checkout.

Keywords and their corresponding values:

`$Author$`	The login name of the user who checked in the revision.
`$Date$`	The date and time the revision was checked in.
`$Header$`	A standard header containing the RCS file name, the revision number, the date, the author, and the state.
`$Locker$`	The login name of the user who locked the revision (empty if not locked).
`$Log$`	The log message supplied during checkin, preceded by a header containing the RCS file name, the revision number, the author, and the date. Existing log messages are *not* replaced. Instead, the new log message is inserted after `$Log:...$`. This is useful for accumulating a complete change log in a source file.
`$Revision$`	The revision number assigned to the revision.
`$Source$`	The full pathname of the RCS file.

`$State$` The state assigned to the revision with `rcs -s` or `ci -s`.

Access Control Lists (ACLs)

Optional ACL entries should not be added to RCS files because they might be deleted.

DIAGNOSTICS

The RCS file name, the working file name, and the revision number retrieved are written to the diagnostic output. The exit status always refers to the last file checked out, and is 0 if the operation was successful, 1 if unsuccessful.

EXAMPLES

Suppose the current directory contains a subdirectory named `RCS` with an RCS file named `io.c,v`. Each of the following commands retrieves the latest revision from `RCS/io.c,v` and stores it into `io.c`:

```
co  io.c
co  RCS/io.c,v
co  io.c,v
co  io.c  RCS/io.c,v
co  io.c  io.c,v
co  RCS/io.c,v  io.c
co  io.c,v  io.c
```

WARNINGS

The `co` command generates the working file name by removing the `,v` from the end of the RCS file name. If the given RCS file name is too long for the file system on which the RCS file should reside, `co` terminates with an error message.

There is no way to suppress the expansion of keywords, except by writing them differently. In `nroff` and `troff`, this is done by embedding the null-character `\&` into the keyword.

The `-d` option gets confused in some circumstances, and accepts no date before 1970.

The `-j` option does not work for files containing lines consisting of a single `.`.

RCS is designed to be used with *text* files only. Attempting to use RCS with non-text (binary) files results in data corruption.

AUTHOR

`co` was developed by Walter F. Tichy.

SEE ALSO

ci(1), ident(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4), acl(5), rcsintro(5).

NAME

col - filter reverse line-feeds and backspaces

SYNOPSIS

`col [-blfxp]`

DESCRIPTION

`col` reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code `ESC-7`), and by forward and reverse half-line feeds (`ESC-9` and `ESC-8`). `col` is particularly useful for filtering multi-column output made with the `nroff .rt` command, and output resulting from use of the `tbl` preprocessor (see *nroff*(1) and *tbl*(1)).

If the `-b` option is given, `col` assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read is output.

If the `-l` option is given, `col` assumes the output device is a line printer (rather than a character printer) and removes backspaces in favor of multiply overstruck full lines. It generates the minimum number of print operations necessary to generate the required number of overstrikes. (All but the last print operation on a line are separated by carriage returns (\r); the last print operation is terminated by a newline (\n).)

Although `col` accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the `-f` (fine) option; in this case, the output from `col` may contain forward half-line feeds (ESC-9), but will still never contain either kind of reverse line motion.

Unless the `-x` option is given, `col` converts white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters SO (\016) and SI (\017) are assumed by `col` to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output SI and SO characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, SI , SO , and VT , (\013), and ESC followed by `7`, `8`, or `9`. The VT character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, `col` ignores any unrecognized escape sequences found in its input; the `-p` option can be used to cause `col` to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `col` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

`col` is used most often with `nroff` and `tbl`. A common usage is:

```
tbl filename | nroff -man | col | more -s
```

(very similar to the usual *man*(1) command). This command allows vertical bars and outer boxes to be printed for tables. The file is run through the `tbl` proprocessor, and the output is then piped through `nroff`, formatting the output using the `-man` macros. The formatted output is then piped through `col`, which sets up the vertical bars and aligns the columns in the file. The file is finally piped through the `more` command, which prints the output to the screen with underlining and highlighting substituted for italic and bold typefaces. The `-s` option deletes excess space from the output so that multiple blank lines are not printed to the screen.

SEE ALSO

nroff(1), tbl(1), ul(1), man(5).

NOTES

The input format accepted by `col` matches the output produced by `nroff` with either the `-T37` or `-Tlp` options. Use `-T37` (and the `-f` option of `col`) if the ultimate disposition of the output of `col` is a device that can interpret half-line motions, and `-Tlp` otherwise.

BUGS

Cannot back up more than 128 lines. Cannot back up across page boundaries.

There is a maximum limit for the number of characters, including backspaces and overstrikes, on a line. The maximum limit is at least 800 characters.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

STANDARDS CONFORMANCE

`col`: SVID2, XPG2, XPG3

NAME

comb - combine SCCS deltas

SYNOPSIS

`comb` [`-p`*sid*] [`-c`*list*] [`-o`] [`-s`] *file* ...

DESCRIPTION

`comb` generates a shell procedure (see *sh*(1)) which, when run, reconstructs the given SCCS files. The reconstructed files are usually smaller than the original files. Arguments can be specified in any order, but all options apply to all named SCCS files. If a directory is named, `comb` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

Options

`comb` recognizes the following options. Each is explained as if only one named file is to be processed, but the effects of any option apply independently to each named file.

-p*SID* The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

`-c`*list* A *list* of deltas to be preserved (see *get*(1) for the syntax of a *list*). All other deltas are discarded.

`-o` For each `get -e` generated, this option causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the `-o` option can decrease the size of the reconstructed SCCS file. It can also alter the shape of the delta tree of the original file.

`-s` This option causes `comb` to generate a shell procedure which, when run, produces a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

$$100 \times (\text{original} - \text{combined}) / \text{original}$$

It is recommended that this option be used before any SCCS files are actually combined to determine exactly how much space is saved by the combining process.

If no options are specified, `comb` preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

Use *help*(1) for explanations.

EXAMPLES

The command:

```
comb -c1.1,1.3,1.6 s.document > save_file
```

creates a shell script named `save_file`, which if executed, creates a new `s.document` using only the deltas `1.1`, `1.3`,and `1.6` from the old `s.document`. The script overwrites the old `s.document`; thus, it might be wise to copy the original elsewhere. Here is an example of typical technique:

```
cp s.document s.save
comb -c1.1,1.3,1.6 s.document > save_file
sh save_file
```

WARNINGS

`comb` may rearrange the shape of the tree of deltas. Combining files may or may not save space; in fact, it is possible for the reconstructed file to actually be larger than the original.

FILES

`s.COMB?????`	Temporary file
`comb?????`	Temporary file

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sh(1), sccsfile(4).

Source Code Control System User Guide in *Programming on HP-UX*.

NAME

comm - select or reject lines common to two sorted files

SYNOPSIS

`comm [-[123]]` *file1 file2*

DESCRIPTION

comm reads *file1* and *file2*, which should be ordered in increasing collating sequence (see *sort*(1) and Environment Variables below), and produces a three-column output:

Column 1:	Lines that appear only in *file1*,
Column 2:	Lines that appear only in *file2*,
Column 3:	Lines that appear in both files.

If `-` is used for *file1* or *file2*, the standard input is used.

Options 1, 2, or 3 suppress printing of the corresponding column. Thus **`comm -12`** prints only the lines common to the two files; **`comm -23`** prints only lines in the first file but not in the second; **`comm -123`** does nothing useful.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence `comm` expects from the input files.

`LANG` determines the language in which messages are displayed.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `comm` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

EXAMPLES

The following examples assume that **`file1`** and **`file2`** have been ordered in the collating sequence defined by the `LC_COLLATE` or `LANG` environment variable.

Print all lines common to **`file1`** and **`file2`** (in other words, print column 3):

```
comm -12 file1 file2
```

Print all lines that appear in **`file1`** but not in **`file2`** (in other words, print column 1):

```
comm -23 file1 file2
```

Print all lines that appear in **`file2`** but not in **`file1`** (in other words, print column 2):

```
comm -13 file1 file2
```

SEE ALSO

cmp(1), diff(1), sdiff(1), sort(1), uniq(1).

STANDARDS CONFORMANCE

`comm`: SVID2, XPG2, XPG3, POSIX.2

NAME

command - execute a simple command

SYNOPSIS

`command` *command_name* [*argument* ...]

DESCRIPTION

`command` enables the shell to treat the arguments as a simple command, supressing the shell function lookup.

If *command_name* is not the name of the function, the effect of `command` is the same as omitting *command*.

OPERANDS

`command` recognizes the following operands:

command_name	The name of a HP-UX command or a shell built-in command.
argument	One or more strings to be interpreted as arguments to *command_name*.

The `command` command is necessary to allow functions that have the same name as a command to call the command (instead of a recursive call to the function).

Nothing in the description of `command` is intended to imply that the command line is parsed any differently than any other simple command. For example,

```
command a | b ; c
```

is not parsed in any special way that causes `|` or `;` to be treated other than a pipe operator or semicolon or that prevents function lookup on *b* or *c*.

EXTERNAL INFLUENCE

Environment Variables

PATH determines the search path used during the command search.

RETURN VALUE

`command` exits with one of the following values:

- If `command` fails:

126	The utility specified by the `command_name` is found but not executable.
127	An error occurred in the *command* utility or the utility specified by `command_name` is not found.

- If `command` does not fail:

  The exit status of `command` is the same as that of the simple command specified by the arguments:

  command_name [*argument* ...]

EXAMPLES

Create a version of the `cd` command that always prints the name of the new working directory whenever it is used:

```
cd() {
        command "$@" >/dev/null
        pwd
}
```

Circumvent the redefined `cd` command above, and change directories without printing the name of the new working directory:

```
command cd
```

SEE ALSO

sh-posix(1), getconf(1), confstr(2).

STANDARDS CONFORMANCE

`command`: SVID2, XPG2, XPG3, proposed POSIX.2 FIPS (June 1990)

NAME

compact, uncompact, ccat - compact and uncompact files, and cat them

SYNOPSIS

`compact` [*name* ...]
`uncompact` [*name* ...]
`ccat` [*file* ...]

DESCRIPTION

`compact` compresses the named files using an adaptive Huffman code. If no file names are given, standard input is compacted and sent to the standard output. `compact` operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to attach a decoding tree in front of the compressed file because the encoder and the decoder start in the same state and stay synchronized. Furthermore, `compact` and `uncompact` can operate as filters. In particular,

```
... | compact | uncompact | ...
```

operates as a (very slow) no-op.

When an argument *file* is given, it is compacted, the resulting file is placed in *file*`.C`, and *file* is unlinked. The first two bytes of the compacted file code the fact that the file is compacted. These bytes are used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical file size reduction (in percent) through compression are: Text, 38%; Pascal Source, 43%; C Source, 36%; and Binary, 19%.

`uncompact` restores the original file from a file compressed by `compact`. If no file names are specified, standard input is uncompacted and sent to the standard output.

`ccat` cats the original file from a file compressed by `compact`, without uncompressing the file.

Access Control Lists (ACLs)

On systems that implement access control lists, when a new file is created with the effective user and group ID of the caller, the original file's ACL is copied to the new file after being altered to reflect any change in ownership (see *acl*(5)).

WARNINGS

On short-filename systems, the last segment of the file name must contain 12 or fewer characters to allow space for the appended `.C`.

DEPENDENCIES

NFS

Access control list entries of networked files are summarized (as returned in `st_mode` by `stat()`), but not copied to the new file (see *stat*(2)).

FILES

`*.C`	compacted file created by compact, removed by uncompact

SEE ALSO

compress(1), pack(1), acl(5).
Gallager, Robert G., "Variations on a Theme of Huffman," *I.E.E.E. Transactions on Information Theory,* vol. IT-24, no. 6, November 1978, pp. 668 - 674.

AUTHOR

`compact` was developed by Colin L. Mc Master.

NAME

compress, uncompress, zcat, compressdir, uncompressdir - compress and expand data

SYNOPSIS

Compress Files

`compress` [`-d`] [`-f`] [`-v`] [`-c`] [`-V`] [`-b` *maxbits*] [*file* ...]
`uncompress` [`-f`] [`-v`] [`-c`] [`-V`] [*file* ...]
`zcat` [`-V`] [*file* ...]

Compress Entire Directory Subtrees

`compressdir` [*options*] [*directory* ...]
`uncompressdir` [*options*] [*directory* ...]

DESCRIPTION

The following commands compress and uncompress files and directory subtrees as indicated:

`compress`	Reduce the size of the named *file*s using adaptive Lempel-Ziv coding. If reduction is possible, each *file* is replaced by a new file of the same name with the suffix `.Z` added to indicate that it is a compressed file. Original ownership, modes, access and modification times are preserved. If no *file* is specified, standard input is compressed to the standard output.
`uncompress`	Restore compressed *file*s to original form. Resulting files have original filename, ownership, and permissions, and the `.Z` filename suffix is removed.
`zcat`	Restore compressed *file*s to original form and send result to standard output.
`compressdir`	Front-end processor. Recursively descend each specified *directory* subtree and use `compress` to compress each file in *directory*. Existing files are replaced by a compressed file having the same name plus the suffix `.Z`, provided the resulting file is smaller than the original. If no directories are specified, compression is applied to all files starting with the current directory. *options* can include any valid `compress` command options (they are passed through to `compress`). To force compression of all files, even when the result is larger than the original file, use the `-f` option.
`uncompressdir`	Opposite of `compressdir`. Restore compressed files to their original form. *options* can include any valid `uncompress` command options (they are passed through to `uncompress`).

The amount of compression obtained depends on the size of the input, the maximum number of bits (*maxbits*) per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50-60 percent. Compression is generally much better than that achieved by Huffman coding (as used in `pack`), or adaptive Huffman coding (`compact`), and takes less time to compute.

Options

These commands recognize the following options in the combinations shown above in SYNOPSIS:

`-d`	Decompress *file*. `compress -d` is equivalent to `uncompress`.
`-f`	Force compression of *file*. This is useful for compressing an entire directory, even if some of the files do not actually shrink. If `-f` is not given and `compress` is run in the foreground, the user is prompted as to whether an existing file should be overwritten.
`-v`	Print a message describing the percentage of reduction for each file compressed.
`-c`	Force `compress` and `uncompress` to write to the standard output; no files are changed. The nondestructive behavior of *zcat* is identical to that of `uncompress -c`.
`-V`	Print the current version and compile options onto the standard error.
`-b` *maxbits*	Specify the maximum number of bits the `compress` algorithm will use. The default is 16 and the range can be any integer between 9 and 16.

`compress` uses the modified Lempel-Ziv algorithm popularized in "A Technique for High Performance Data Compression," Terry A. Welch, *IEEE Computer,* vol. 17, no. 6 (June 1984), pages 8-19. Common substrings in the file are first replaced by 9-bit codes 257 and up. When code 512 is reached, the algorithm switches to 10-bit codes and continues to use more bits until the limit specified by the `-b` flag is reached (default 16).

After the *maxbits* limit is attained, `compress` periodically checks the compression ratio. If it is increasing, `compress` continues to use the existing code dictionary. However, if the compression ratio is decreasing, `compress` discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next "block" of the file.

Note that the `-b` flag is omitted for `uncompress` since the *maxbits* parameter specified during compression is encoded within the output, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is attempted.

Access Control Lists

`compress` retains a file's access control list when compressing and expanding data.

RETURN VALUE

These commands return the following values upon completion:

0	Completed successfully.
2	Last file is larger after (attempted) compression.
1	An error occurred.

DIAGNOSTICS

`Usage: compress [-dfvcV] [-b maxbits] [file ...]`
: Invalid options were specified on the command line.

`Missing maxbits`
: *maxbits* must follow `-b`.

file`: not in compressed format`
: The file specified to `uncompress` has not been compressed.

file`: compressed with` *xx* `bits, can only handle` *yy* `bits`
: *file* was compressed by a program that could deal with a higher value of *maxbits* than the compress code on this machine. Recompress the file with a lower value of *maxbits*.

file`: already has .Z suffix -- no change`
: The file is assumed to be already compressed. Rename the file and try again.

file`: filename too long to tack on .Z`
: The output file name, which is the source file name with a `.Z` extension, is too long for the file system on which the source file resides. Make the source file name shorter and try again.

`file already exists; do you wish to overwrite (y or n)?`
: Respond `y` if you want the output to replace the existing file; `n` if not.

`uncompress: corrupt input`
: A `SIGSEGV` violation was detected which usually means that the input file has been corrupted.

`Compression:` *xx.xx*`%`
: Percentage of the input saved by compression. (Relevant only for `-v`.)

`-- not a regular file: unchanged`
: When the input file is not a regular file (a directory for example), it is left unaltered.

`-- has` *xx* `other links: unchanged`
: The input file has links and has been left unchanged. See *ln*(1) for more information.

`-- file unchanged`
: No savings is achieved by compression. The input remains unaltered.

EXAMPLES

Compress the file named `zenith` and print compression information to the terminal:

```
compress -v zenith
```

The terminal display shows a line resembling the following:

```
zenith: Compression: 23.55% -- replaced with zenith.Z
```

indicating that the compressed file is 23.55% smaller than the original, or

```
zenith: Compression: -12.04% -- file unchanged
```

indicating that an additional 12.04% space must be used to compress the file.

To undo the compression, type:

```
uncompress zenith.Z
```

or

```
compress -d zenith.Z
```

This restores file `zenith.Z` to its original uncompressed form and name.

WARNINGS

Although compressed files are compatible between machines with large memory, `-b12` should be used for file transfer to architectures with a small process data space (64K bytes or less).

NFS

Access control lists of networked files are summarized (as returned in `st_mode` by `stat()`, but not copied to the new file (see *stat*(2)).

AUTHOR

`compress` was developed by Joseph M. Orost, Kenneth E. Turkowski, Spencer W. Thomas, and James A. Woods.

FILES

`*.Z`	Compressed file created by `compress` and removed by `uncompress`.

SEE ALSO

compact(1), pack(1), acl(5).

NAME

cp - copy files and directory subtrees

SYNOPSIS

cp [**-f** | **-i**] [**-p**] *file1 new_file*
cp [**-f** | **-i**] [**-p**] *file1* [*file2* ...] *dest_directory*
cp [**-f** | **-i**] [**-p**] [**-R** | **-r**] *directory1* [*directory2* ...] *dest_directory*

DESCRIPTION

cp copies:

- *file1* to new or existing *new_file*,
- *file1* to existing *dest_directory*,
- *file1*, *file2*, ... to existing *dest_directory*,
- directory subtree *directory1*, to new or existing *dest_directory*. or
- multiple directory subtrees *directory1*, *directory2*, ... to new or existing *dest_directory*.

cp fails if *file1* and *new_file* are the same (be cautious when using shell metacharacters). When destination is a directory, one or more files are copied into that directory. If two or more files are copied, the destination must be a directory. When copying a single file to a new file, if *new_file* exists, its contents are destroyed.

If the access permissions of the destination *dest_directory* or existing destination file *new_file* forbid writing, **cp** aborts and produces an error message "cannot create *file*".

To copy one or more directory subtrees to another directory, the **-r** option is required. The **-r** option is ignored if used when copying a file to another file or files to a directory.

If *new_file* is a link to an existing file with other links, **cp** overwrites the existing file and retains all links. If copying a file to an existing file, **cp** does not change existing file access permission bits, owner, or group.

When copying files to a directory or to a new file that does not already exist, **cp** creates a new file with the same file miscellaneous bits as *file1*, except that the sticky bit is not set unless you are the user with appropriate privilege (see *chmod*(2)). The owner and group of the new file or files are those of the user. The last modification time of *new_file* (and last access time, if *new_file* did not exist) and the last access time of the source *file1* are set to the time the copy was made.

Options

-i (interactive copy) Cause **cp** to write a prompt to standard error and wait for a response before copying a file that would overwrite an existing file. If the response from the standard input is affirmative, the file is copied if permissions allow the copy. If the **-i** (interactive) and **-f** (forced-copy) options are both specified, the **-i** option is ignored.

-f Force existing destination pathnames to be removed before copying, without prompting for confirmation. This option has the effect of destroying and replacing any existing file whose name and directory location conflicts with the name and location of the new file created by the copy operation.

-p (preserve permissions) Causes **cp** to preserve in the copy as many of the modification time, access time, file mode, user ID, and group ID as allowed by permissions.

-r (recursive subtree copy) Cause **cp** to copy the subtree rooted at each source directory to *dest_directory*. If *dest_directory* exists, it must be a directory, in which case **cp** creates a directory within *dest_directory* with the same name as *file1* and copies the subtree rooted at *file1* to *dest_directory*/*file1*. An error occurs if *dest_directory*/*file1* already exists. If *dest_directory* does not exist, **cp** creates it and copies the subtree rooted at *file1* to *dest_directory*. Note that **cp -r** cannot merge subtrees.

Only normal files and directories are copied. Character special devices, block special devices, network special files, named pipes, symbolic links, and sockets are not copied, and a warning is printed stating that the file was skipped. *dest_directory* should not reside within *directory1*, nor should *directory1* have a cyclic directory structure, since in both cases **cp** attempts to copy an infinite amount of data.

-R (recursive subtree copy) The **-R** option is identical to the **-r** option with the exception that directories copied by the **-R** option are created with read, write, and search permission for the owner. User and group permissions remain unchanged.

Access Control Lists (ACLs)

If *new_file* is a new file, or if a new file is created in *dest_directory*, it inherits the access control list of the original *file1*, *file2*, etc., altered to reflect any difference in ownership between the two files (see *acl*(5)).

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters.

`LANG` and `LC_CTYPE` determine the local language equivalent of y (for yes/no queries).

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `cp` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

The following command moves the directory *sourcedir* and its contents to a new location (*targetdir*) in the file system. Since `cp` creates the new directory, the destination directory *targetdir* should not already exist.

```
cp -r sourcedir targetdir && rm -rf sourcedir
```

The `-r` option copies the subtree (files and subdirectories) in directory *sourcedir* to directory *targetdir*. The double ampersand (**&&**) causes a conditional action. If the operation on the left side of the **&&** is successful, the right side is executed (and removes the old directory). If the operation on the left of the **&&** is not successful, the old directory is not removed.

This example is equivalent to:

```
mv sourcedir targetdir
```

To copy all files and directory subtrees in the current directory to an existing *targetdir*, use:

```
cp -r * targetdir
```

To copy all files and directory subtrees in *sourcedir* to *targetdir*, use:

```
cp -r sourcedir/* targetdir
```

Note that directory pathnames can precede both *sourcedir* and *targetdir*.

To create a zero-length file, use any of the following:

```
cat /dev/null >file
cp /dev/null file
touch file
```

DEPENDENCIES

NFS

Access control lists of networked files are summarized (as returned in `st_mode` by `stat()`), but not copied to the new file. When using `mv` or `ln` on such files, a `+` is not printed after the mode value when asking for permission to overwrite a file.

AUTHOR

`cp` was developed by AT&T, the University of California, Berkeley, and HP.

SEE ALSO

cpio(1), ln(1), mv(1), rm(1), link(1M), lstat(2), readlink(2), stat(2), symlink(2), symlink(4), acl(5).

STANDARDS CONFORMANCE

`cp`: SVID2, XPG2, XPG3, POSIX.2

NAME

cpio - copy file archives in and out

SYNOPSIS

`cpio -o [aABcxvCh]`

`cpio -i [BdcrtuxvmfPsSb6RU]` [*patterns*]

`cpio -p [aduxvlmrU]` *directory*

DESCRIPTION

`cpio -o` (copy out) Read the standard input to obtain a list of path names, and copy those files to the standard output together with path name and status information. Output is padded to a 512-byte boundary.

`cpio -i` (copy in) Extract files from the standard input which is assumed to be the product of a previous `cpio -o`. Only files with names that match *patterns*, according to the rules of Pattern Matching Notation (see *regexp*(5)), are selected. In addition, a leading `!` within a pattern indicates that only those names should be selected that do *not* match the remainder of the pattern. Multiple *patterns* can be specified. If no *patterns* are specified, the default for *patterns* is `*` (select all files). Extracted files are conditionally created and copied into the current directory tree, as determined by the options described below. The permissions of the files match the permissions of the original files when the archive was created by `cpio -o` unless the `-U` option is used. File owner and group are that of the current user unless the user has appropriate privileges, in which case `cpio` retains the owner and group of the files of the previous `cpio -o`.

`cpio -p` (pass) Read the standard input to obtain a list of path names of files which are then conditionally created and copied into the destination *directory* tree as determined by the options described below. Destination path names are interpreted relative to the named *directory*.

Options

`cpio` recognizes the following options in addition to `-i`, `-o`, and `-p`:

`a` Reset access times of input files after they are copied.

`A` Suppress warning messages regarding optional access control list entries. *cpio*(1) does not backup optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file that has optional access control list entries.

`B` Block input/output at 5 120 bytes to the record (does not apply to the `cpio -p` option). This option is meaningful only with data directed to or from devices that support variable-length records such as magnetic tape.

`d` Create directories as needed.

`c` Write or read header information in ASCII character form for portability.

`r` Rename files interactively. If the user types a null line, the file is skipped.

`t` Print only a table of contents of the input. No files are created, read, or copied.

`u` Copy unconditionally (normally, an older file does not replace a newer file with the same name).

`x` Save or restore device special files. Since `mknod()` is used to recreate these files on a restore, `-ix` and `-px` can only be used by users with appropriate privileges (see *mknod*(2)). This option is intended for intrasystem (backup) use only. Restoring device files onto a different system can be very dangerous.

`v` Verbose: cause a list of file names to be printed. When used with the `t` option, the table of contents looks like the output of an `ls -l` command (see *ls*(1)).

`l` Whenever possible, link files rather than copying them. This option does not destroy existing files. Usable only with the `-p` option.

`m` Retain previous file modification time. This option does not affect directories that are being copied.

`f`	Copy in all files except those in *patterns*.
`P`	Read a file written on a PDP-11 or VAX system (with byte swapping) that did not use the `-c` option. Only useful with `-i` (copy in). Files copied in this mode are not changed. Non-ASCII files are likely to need further processing to be readable. This processing often requires knowledge of file contents, and thus cannot always be done by this program. (PDP-11 and VAX are registered trademarks of Digital Equipment Corporation). The `-s`, `-S`, and `-b` options below can be used when swapping all the bytes on the tape (rather than just the headers) is appropriate. In general, text is best processed with `-P` and binary data with one of the other options.
`s`	Swap all bytes of the file. Use only with the `-i` option.
`S`	Swap all half-words in the file. Use only with the `-i` option.
`b`	Swap both bytes and half-words. Use only with the `-i` option.
`6`	Process a UNIX Sixth-Edition-format file. Only useful with `-i` (copy in).
`R`	Resynchronize automatically when `cpio` goes "Out of phase," (see DIAGNOSTICS).
`C`	Have `cpio` checkpoint itself at the start of each volume. If `cpio` is writing to a streaming tape drive with immediate-report mode enabled and a write error occurs, it normally aborts and exits with return code **2**. With this option specified, `cpio` instead automatically restarts itself from the checkpoint and rewrites the current volume. Alternatively, if `cpio` is not writing to such a device and a write error occurs, `cpio` normally continues with the next volume. With this option specified, however, the user can choose to either ignore the error or rewrite the current volume.
`h`	Follow symbolic links as though they were normal files or directories. Normally, `cpio` archives the link.
`U`	Use the process's file-mode creation mask (see *umask*(2)) to modify the mode of files created, in the same manner as *creat*(2).

Note that `cpio` archives created using a raw device file must be read using a raw device file.

When the end of the tape is reached, `cpio` prompts the user for a new special file and continues.

If you want to pass one or more metacharacters to `cpio` without the shell expanding them, be sure to precede each of them with a backslash (\).

Device files written with the `-ox` option (such as `/dev/tty03`) do not transport to other implementations of HP-UX.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in pattern matching notation.

LC_TIME determines the format and content of date and time strings output when listing the contents of an archive with the `-v` option.

LANG determines the language in which messages are displayed.

If **LC_COLLATE**, **LC_CTYPE**, or **LC_TIME** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `cpio` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

The diagnostic message `Out of phase` indicates that `cpio` could not successfully read its particular "magic number" in the header. Without the R option specified, `cpio` fails and returns an exit code of 2. With the R option, `cpio` attempts to resync automatically (resyncing means that `cpio` tries to find the next good header in the archive and continues processing from there). If `cpio` tries to resynchronize from being "Out of phase", it returns an exit code of 1. If resynchronization fails, try changing header mode (`-c` option) or byte swapping the header (`-P` or `-s` options).

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/rmt/0m

cd olddir
find .  -depth -print | cpio -pd newdir
```

The trivial case `find . -depth -print | cpio -oB >/dev/rmt/0m` can be handled more efficiently by:

```
find . -cpio /dev/rmt/0m
```

WARNINGS

Do not redirect the output of `cpio` to a named `cpio` archive file residing in the same directory as the original files belonging to that `cpio` archive. This can cause loss of data.

`cpio` strips any leading `./` characters in the list of filenames piped to it.

Path names are restricted to **PATH_MAX** characters (see `<limits.h>` and *limits*(5)). If there are too many unique linked files, the program runs out of memory to keep track of them. Thereafter, linking information is lost. Only users with appropriate privileges can copy special files.

`cpio` tapes written on HP machines with the `-ox[c]` options can sometimes mislead (non-HP) versions of `cpio` that do not support the `-x` option. If a non-HP(or non-AT&T) version of `cpio` happens to be modified so that (HP) `cpio` recognizes it as a device special file, a spurious device file might be created.

If `/dev/tty` is not accessible, `cpio` issues a complaint and exits.

The `-pd` option does not create the directory typed on the command line.

The `-idr` option does not make empty directories.

The `-plu` option does not link files to existing files.

POSIX defines a file named **TRAILER!!!** as an end-of-archive marker. Consequently, if a file of that name is contained in a group of files being written by `cpio -o`, the file is interpreted as end-of-archive, and no remaining files are copied. Recommended practice is to avoid naming files anything that resembles an end-of-archive file name.

To create a POSIX-conforming cpio archive, the `-c` option must be used. To read a POSIX-conforming cpio archive, the `-c` option must be used and the `-b`, `-s`, `-S`, and `-6` options should not be used. If the user does not have appropriate privileges, the `-U` option must also be used to get POSIX-conforming behavior when reading an archive. Users with appropriate privileges should not use this option to get POSIX-conforming behavior.

Using Cartridge Tape Drives:

The use of `cpio` with cartridge tape units requires additional comments. For an explanation of the constraints on cartridge tapes, see *ct*(7).

Using `cpio` to write directly to a cartridge tape unit can severely damage the tape drive in a short amount of time, and is therefore strongly discouraged. The recommended method of writing to the cartridge tape unit is to use *tcio*(1) in conjunction with `cpio` (note that `-B` must not be used when *tcio*(1) is used) because *tcio*(1) buffers data into larger pieces suitable for cartridge tapes. The `-B` option must be used when writing directly (that is, without using *tcio*(1)) to a CS/80 cartridge tape unit.

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

SEE ALSO

ar(1), find(1), tar(1), tcio(1), cpio(4), acl(5), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

`cpio`: SVID2, XPG2, XPG3

NAME

cpp - the C language preprocessor

SYNOPSIS

`/lib/cpp` [*option...*] [*ifile* [*ofile*]]

DESCRIPTION

is the C language preprocessor which is invoked as the first pass of any C compilation using the `cc` command (see *cpp*(1)). Its purpose is to process `include` and conditional compilation instructions, and macros. Thus the output of `cpp` is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, `cpp` and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of `cpp` in other than this framework is not suggested. The preferred way to invoke `cpp` is through the `cc` command, since the functionality of `cpp` may someday be moved elsewhere. See *m4*(1) for a general macro processor.

`cpp` optionally accepts two file names as arguments. *ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not specified.

The following options to `cpp` are recognized:

-P Preprocess the input without producing the line-control information used by the next pass of the C compiler.

-C By default, `cpp` strips C-style comments. If the `-C` option is specified, all comments (except those found on `cpp` directive lines) are passed along.

-U*name* Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	`unix`	`__unix`
hardware:		
	`hp9000s800`	`hp9000s500`
	`hp9000s300`	`hp9000s200`
	`hp9000ipc`	
	`__hp9000s800`	`__hp9000s300`
	`hppa`	`__hppa`
	`_PA_RISC1_0`	`_PA_RISC1_1`
	`_WSIO`	`_SIO`
UNIX systems variant:	`PWB _PWB hpux`	`__hpux _HPUX_SOURCE`
lint(1)	`lint __lint`	

In addition, all symbols that begin with an underscore and either an uppercase letter or another underscore are reserved. Other symbols may be defined by the `CCOPTS` variable or other command-line options to the C compiler at compile time (see *cc*(1)). All HP-UX systems have the symbols `PWB`, `hpux`, `unix`, `_PWB`, `__hpux`, and `__unix` defined. Each system defines at least one hardware variant, as appropriate. The lint symbols are defined when *lint*(1) is running. See DEPENDENCIES.

-A Remove all predefined symbols that begin with a letter and `_HPUX_SOURCE`. The user is expected to define `_POSIX_SOURCE` or `_XOPEN_SOURCE` when using this option.

-D*name*
-D*name*=*def* Define *name* as if by a `#define` directive. If no =*def* is given, *name* is defined as 1. The `-D` option has lower precedence than the `-U` option. Thus, if the same name is used in both a `-U` option and a `-D` option, the name is undefined regardless of the order of the options.

-T HP-UX no longer restricts preprocessor symbols to eight characters. The `-T` option forces `cpp` to use only the first eight characters for distinguishing different preprocessor names. This behavior is the same as preprocessors on some other systems with respect to the length of names, and is included for backward compatability.

-I*dir* Change the algorithm for searching for **#include** files whose names do not begin with **/** to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in double quotes (**""**) are searched for first in the directory of the file containing the **#include** line, then in directories named in **-I** options in left-to-right order, and last in directories on a standard list. For **#include** files whose names are enclosed in **<>**, the directory of the file containing the **#include** line is not searched. However, directory *dir* is still searched.

-H*nnn* Change the internal macro definition table to be *nnn* bytes in size. The macro symbol table will be increased proportionately. The default table size is at least 128 000 bytes. This option serves to eliminate "too many defines" and "too much defining" errors.

Two special names are understood by **cpp**. The name **__LINE__** is defined as the current line number (as a decimal integer) as counted by **cpp**, and **__FILE__** is defined as the current file name (as a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

All **cpp** directives start with lines begun by **#**. Any number of blanks and tabs are allowed between the **#** and the directive. The directives are:

#define *name token-string*
Replace subsequent instances of *name* with *token-string*. (*token-string* can be null).

#define *name*(*arg*, ..., *arg*) *token-string*
Notice that there can be no space between *name* and the **(**. Replace subsequent instances of *name* followed by a **(**, a list of comma-separated set of tokens, and a **)** by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to expand at the beginning of newly created *token-string*.

#undef *name* Cause the definition of *name* (if any) to be forgotten from now on.

#include "*filename*"
#include <*filename*>
Include at this point the contents of *filename* (which is then run through **cpp**). See the **-I** option above for more detail.

#line *integer-constant "filename"*
Causes **cpp** to generate line-control information for the next pass of the C compiler. *integer-constant* is the line number of the next line and *filename* is the file where it comes from. If *"filename"* is not given (in double quotes), the current file name is unchanged.

#endif <**text**>
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. Any *text* occurring on the same line as the **#endif** is ignored and thus may be used to mark matching **#if**-**#endif** pairs. This makes it easier, when reading the source, to match **#if**, **#ifdef**, and **#ifndef** directives with their associated **#endif** directive.

#ifdef *name* The lines following appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name*
The lines following do not appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*
Lines following appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary **-**, **!**, and **~** operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined** which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the use of **#ifdef** and **#ifndef** in an **#if** directive. Only these operators, integer constants, and names which are known by **cpp** should be used in *constant-*

expression. In particular, the `sizeof` operator is not available.

`#else` Reverses the notion of the test directive which matches this directive. Thus, if lines previous to this directive are ignored, the following lines appear in the output, and vice versa.

The test directives and the possible `#else` directives can be nested. `cpp` supports names up to 255 characters in length.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of comments and string literals as single and/or multi-byte characters.

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `cpp` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

Error messages produced by `cpp` are intended to be self-explanatory The line number and filename where the error occurred are printed along with the diagnostic.

DEPENDENCIES

Series 300

In the hardware *name* definition associated with predefined symbols (see `-U` option), two hardware variants are defined instead of one. Both `hp9000s200` and `hp9000s300` are present, and they are treated synonymously because of the similarity between the two series.

Series 700

The symbols `hp9000s700` and `__hp9000s700` are not reserved symbols recognized by the `-U` option. They are supplied to `cpp` either automatically by the compiler, or by use of a compiler option. For example, on a Series 700 system, executing:

```
cc -v t.c
```

produces:

```
/lib/cpp  t.c  /tmp/ctmAAAa29220  -D__hp9000s700  -D__hp9000s800
-D__hppa...
```

(Also see *cc*(1)), `-D` option.)

FILES

`/usr/include`	standard directory for `#include` files

SEE ALSO

cc(1), m4(1).

NOTES

When new-line characters were found in argument lists for macros to be expanded, previous versions of `cpp` put out the new-lines as they were found and expanded. The current version of `cpp` replaces these new-lines with blanks to alleviate problems that the previous versions had when this occurred.

STANDARDS CONFORMANCE

`cpp`: SVID2, XPG2

NAME

crontab - user crontab file

SYNOPSIS

```
crontab [file ]
crontab -r
crontab -l
```

DESCRIPTION

`crontab` copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontab files (see *cron*(1M)). The `-r` option removes a user's crontab from the crontab directory. `crontab -l` lists the crontab file for the invoking user.

Users are permitted to use `crontab` if their names appear in the file `/usr/lib/cron/cron.allow`. If that file does not exist, the file `/usr/lib/cron/cron.deny` is checked to determine if the user should be denied access to `crontab`. If neither file exists, only root is allowed to submit a job. If only `cron.deny` exists and is empty, global usage is permitted. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0-59),
hour (0-23),
day of the month (1-31),
month of the year (1-12),
day of the week (0-6 with 0=Sunday).

Each of these patterns can be either an asterisk (meaning all legal values), or a list of elements separated by commas. An element is either a number, or two numbers separated by a hyphen (meaning an inclusive range). Note that the specification of days can be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, `0 0 1,15 * 1` runs a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to `*` (for example, `0 0 * * 1` runs a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by `\`) is translated to a new-line character. Only the first line (up to a `%` or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an initial argument of `sh`. Users who desire to have their `.profile` executed must explicitly do so in the crontab file. `cron` supplies a default environment for every shell, defining **HOME**, **LOGNAME**, **SHELL=**`/bin/sh`, and **PATH=**`:/bin:/usr/bin`.

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

WARNINGS

Be sure to redirect the standard output and standard error from commands. If this is not done, any generated output or errors are mailed to the user.

FILES

`/usr/lib/cron`	main cron directory
`/usr/lib/cron/cron.allow`	list of allowed users
`/usr/lib/cron/cron.deny`	list of denied users
`/usr/spool/cron/crontabs`	directory containing the crontab files
`/usr/lib/cron/log`	accounting information

SEE ALSO

sh(1), cron(1M), queuedefs(4).

STANDARDS CONFORMANCE

crontab: SVID2, XPG2, XPG3

NAME

crypt - encode/decode files

SYNOPSIS

`crypt` [*password*]

DESCRIPTION

`crypt` reads from the standard input and writes on the standard output. *password* is a key that selects a particular transformation. If no *password* is given, `crypt` demands a key from the terminal and turns off printing while the key is being typed in. `crypt` encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

The latter command decrypts the file and prints the clear version.

Files encrypted by `crypt` are compatible with those treated by the `ed` editor in encryption mode (see *ed*(1)).

Security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or clear text can become visible must be minimized.

`crypt` implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive; i.e., to take a substantial fraction of a second to compute. However, if keys are restricted to, for example, three lowercase letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the `crypt` command, it is potentially visible to users executing the `ps` or a derivative (see *ps*(1)). The choice of keys and key security are the most vulnerable aspect of `crypt`.

EXAMPLES

The following example demonstrates the use of `crypt` to edit a file that the user wants to keep strictly confidential:

```
$ crypt <plans >plans.x
key: violet
$ rm plans
      ...
$ vi -x plans.x
key: violet
      ...
:wq
$
      ...
$ crypt <plans.x | pr
key: violet
```

Note that the `-x` option is the encryption mode of `vi`, and prompts the user for the same key with which the file was encrypted.

WARNINGS

If output is piped to `nroff` and the encryption key is *not* given on the command line, `crypt` can leave terminal modes in a strange state (see *nroff*(1) and *stty*(1)).

If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the the first of the original files is decrypted correctly.

FILES

`/dev/tty`	for typed key

SEE ALSO

ed(1), makekey(1), stty(1).

NAME

csh - a shell (command interpreter) with C-like syntax

SYNOPSIS

`csh` [`-cefinstvxTVX`] [*command_file*] [*argument_list* ...]

DESCRIPTION

`csh` is a command language interpreter that incorporates a command history buffer, C-like syntax, and job control facilities.

Command Options

Command options are interpreted as follows:

`-c`	Read commands from the (single) following argument which must be present. Any remaining arguments are placed in `argv`.
`-e`	C shell exits if any invoked command terminates abnormally or yields a non-zero exit status.
`-f`	Suppress execution of the `.cshrc` file in your home directory, thus speeding up shell start-up time.
`-i`	Force `csh` to respond interactively when called from a device other than a computer terminal (such as another computer). `csh` normally responds non-interactively. If `csh` is called from a computer terminal, it always responds interactively, regardless of which options are selected.
`-n`	Parse but do *not* execute commands. This is useful for checking syntax in shell scripts. All substitutions are performed (history, command, alias, etc.).
`-s`	Take command input from the standard input.
`-t`	Read and execute a single line of input.
`-v`	Set the `verbose` shell variable, causing command input to be echoed to the standard output device after history substitutions are made.
`-x`	Set the `echo` shell variable, causing all commands to be echoed to the standard error immediately before execution.
`-T`	Disable the tenex features which use the ESC key for command/file name completion and CTRL-D for listing available files (see the *CSH UTILITIES* section below)
`-V`	Set the `verbose` variable before `.cshrc` is executed so that all `.cshrc` commands are also echoed to the standard output.
`-X`	Set the `echo` variable before `.cshrc` is executed so that all `.cshrc` commands are also echoed to the standard output.

After processing the command options, if arguments remain in the argument list, and the `-c`, `-i`, `-s`, or `-t` options were not specified, the first remaining argument is taken as the name of a file of commands to be executed.

COMMANDS

A simple command is a sequence of words, the first of which specifies the command to be executed. A sequence of simple commands separated by vertical bar (`|`) characters forms a pipeline. The output of each command in a pipeline becomes the input for the next command in the pipeline. Sequences of pipelines can be separated by semicolons (`;`) which causes them to be executed sequentially. A sequence of pipelines can be executed in background mode by adding an ampersand character (`&`) after the last entry.

Any pipeline can be placed in parentheses to form a simple command which, in turn, can be a component of another pipeline. Pipelines can also be separated by `||` or `&&` indicating, as in the C language, that the second pipeline is to be executed only if the first fails or succeeds, respectively.

Jobs

`csh` associates a **job** with each pipeline and keeps a table of current jobs (printed by the `jobs` command) and assigns them small integer numbers. When a job is started asynchronously using `&`, the shell prints a line resembling:

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and want to do something else, you can type the currently defined *suspend* character (see *termio*(7)) which sends a stop signal to the current job. `csh` then normally indicates that the job has been 'Stopped', and prints another prompt. You can then manipulate the state of this job, putting it in the background with the `bg` command, run some other commands, and then eventually bring the job back into the foreground with the foreground command `fg`. A *suspend* takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is a delayed *suspend* character which does not generate a stop signal until a program attempts to *read*(2) it. This can usefully be typed ahead when you have prepared some commands for a job which you want to stop after it has read them.

A job being run in the background stops if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command `stty tostop` (see *stty*(1)). If you set this tty option, background jobs stop when they try to produce output, just as they do when they try to read input. Keyboard signals and line-hangup signals from the terminal interface are not sent to background jobs on such systems. This means that background jobs are immune to the effects of logging out or typing the interrupt, quit, suspend, and delayed suspend characters (see *termio*(7)).

There are several ways to refer to jobs in the shell. The character `%` introduces a job name. If you wish to refer to job number 1, you can name it as `%1`. Just naming a job brings it to the foreground; thus `%1` is a synonym for `fg %1` , bringing job 1 back into the foreground. Similarly, typing `%1 &` resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them if these prefixes are unambiguous; thus `%ex` normally restarts a suspended *ex*(1) job, if there is only one suspended job whose name begins with the string `ex`. It is also possible to say `%?string` which specifies a job whose text contains *string*, if there is only one such job.

`csh` maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a `+` and the previous job with a `-`. The abbreviation `%+` refers to the current job and `%-` refers to the previous job. For close analogy with the syntax of the **history** mechanism (described below), `%%` is also a synonym for the current job.

`csh` learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before printing a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable `notify`, `csh` notifies you immediately of changes in status of background jobs. There is also a `csh` built-in command called `notify` which marks a single process so that any status change is immediately reported. By default, `notify` marks the current process. Simply type `notify` after starting a background job to mark it.

If you try to leave the shell while jobs are stopped, `csh` sends the warning message: `You have stopped jobs.` Use the `jobs` command to see what they are. If you do this or immediately try to exit again, `csh` does not warn you a second time, and the suspended jobs are terminated (see *exit*(2)).

Built-In Commands

Built-in commands are executed within the shell without spawning a new process. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell. The built-in commands are:

`alias`
`alias` *name*
`alias` *name wordlist*
: The first form prints all aliases. The second form prints the alias for *name*. The third form assigns the specified *wordlist* as the alias of *name*. Command and file name substitution are performed on *wordlist*. *name* cannot be `alias` or `unalias`.

`bg [%`*job* ...`]`
: Put the current (*job* not specified) or specified jobs into the background, continuing them if they were stopped.

`break` Causes execution to resume after the `end` of the nearest enclosing `foreach` or `while`. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a **switch**, resuming after the **endsw**.

case *label* **:**
A label in a **switch** statement as discussed below.

cd
cd *directory_name*
chdir
chdir *directory_name*
Change the shell's current working directory to *directory_name*. If not specified, *directory_name* defaults to your home directory.
If *directory_name* is not found as a subdirectory of the current working directory (and does not begin with **/**, **./**, or **../**), each component of the variable *cdpath* is checked to see if it has a subdirectory *directory_name*. Finally, if all else fails, **csh** treats *directory_name* as a shell variable. If its value begins with **/**, this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing **while** or **foreach**. The rest of the commands on the current line are executed.

default:
Labels the default case in a **switch** statement. The default should come after all other **case** labels.

dirs Prints the directory stack; the top of the stack is at the left; the first directory in the stack is the current directory.

echo *wordlist*
echo -n *wordlist*
The specified words are written to the shell's standard output, separated by spaces, and terminated with a new-line unless the **-n** option is specified.

else
end
endif
endsw See the descriptions of the **foreach**, **if**, **switch**, and **while** statements below.

eval *arguments* ...
(Same behavior as *sh*(1).) *arguments* are read as input to the shell and the resulting command(s) executed. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions.

exec *command*
The specified command is executed in place of the current shell.

exit
exit **(***expression***)**
csh exits either with the value of the **status** variable (first form) or with the value of the specified *expression* (second form).

fg [**%***job* ...]
Brings the current (*job* not specified) or specified jobs into the foreground, continuing them if they were stopped.

foreach *name* **(***wordlist***)**
. . .
end The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching **end** are executed. (Both **foreach** and **end** must appear alone on separate lines.)

The built-in command **continue** can be used to continue the loop prematurely; the built-in command **break** to terminate it prematurely. When this command is read from the terminal, the loop is read once, prompting with **?** before any statements in the loop are executed. If you make a mistake while typing in a loop at the terminal, use the erase or line-kill character as appropriate to recover.

`glob` *wordlist*
: Like `echo` but no `\` escapes are recognized and words are delimited by null characters in the output. Useful in programs that use the shell to perform file name expansion on a list of words.

`goto` *word*
: The specified *word* is file name and command expanded to yield a string of the form `label`. The shell rewinds its input as much as possible and searches for a line of the form `label:` possibly preceded by blanks or tabs. Execution continues after the specified line.

`hashstat`
: Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **execs**). An **exec** is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component that does not begin with a `/`.

`history [-h][-r][`*n*`]`
: Displays the history event list. If *n* is given, only the *n* most recent events are printed. The `-r` option reverses the order of printout to be most recent first rather than oldest first. The `-h` option prints the history list without leading numbers for producing files suitable for the `source` command.

`if (`*expression*`)` *command*
: If *expression* evaluates true, the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the `if` command. *command* must be a simple command; not a pipeline, a command list, a parenthesized command list, or an aliased command. Input/output redirection occurs even if *expression* is false, meaning that *command* is *not* executed (this is a bug).

`if (`*expression1*`) then`
  . . .
`else if (`*expression2*`) then`
  . . .
`else`
  . . .
`endif` If *expression1* is true, all commands down to the first `else` are executed; otherwise if *expression2* is true, all commands from the first `else` down to the second `else` are executed, etc. Any number of `else-if` pairs are possible, but only one `endif` is needed. The `else` part is likewise optional. (The words `else` and `endif` must appear at the beginning of input lines. The `if` must appear alone on its input line or after an `else`.)

`jobs [-l]`
: Lists active jobs. The `-l` option lists process IDs in addition to the usual information.

`kill %` *job*
`kill -`*sig* `%`*job* ...
`kill` *pid*
`kill -`*sig pid* . . .
`kill -l`
: Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in `/usr/include/signal.h`, stripped of the `SIG` prefix (see *signal*(2)). The signal names are listed by `kill -l`. There is no default, so `kill` used alone does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), the job or process is sent a CONT (continue) signal as well.

`login` Terminates a login shell, replacing it with an instance of `/bin/login`. This is one way to log off, included for compatibility with *sh*(1).

`logout`
: Terminates a login shell. Especially useful if *ignoreeof* is set. A similar function, **bye**, which works for sessions that are not login shells, is provided for historical reasons. Its use is not recommended because it is not part of the standard BSD `csh` and may not be supported in future releases.

`newgrp`
: Changes the group identification of the caller; for details see *newgrp*(1). A new shell is executed by `newgrp` so that the current shell environment is lost.

`nice`
`nice` *+number*
`nice` *command*
`nice` *+number command*
: The first form sets the *nice* (run command priority) for this shell to 4 (the default). The second form sets the priority to the given *number*. The final two forms run *command* at priority 4 and *number* respectively. The user with appropriate privileges can raise the priority by specifying negative niceness using `nice` *-number* ... *command* is always executed in a sub-shell, and restrictions placed on commands in simple `if` statements apply.

`nohup` [*command*]
: Without an argument, `nohup` can be used in shell scripts to cause hangups to be ignored for the remainder of the script. With an argument, causes the specified *command* to be run with hangups ignored. All processes executed in the background with `&` are effectively `nohup`ed as described under Jobs in the *COMMANDS* section.

`notify` [*job* ...]
: Causes the shell to notify the user asynchronously when the status of the current (*job* not specified) or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

`onintr` [-] [*label*]
: Controls the action of the shell on interrupts. With no arguments, *onintr* restores the default action of the shell on interrupts, which action is to terminate shell scripts or return to the terminal command input level. If `-` is specified, all interrupts are ignored. If a *label* is given, the shell executes a `goto` *label* when an interrupt is received or a child process terminates because it was interrupted.

  If the shell is running in the background and interrupts are being ignored, *onintr* has no effect; interrupts continue to be ignored by the shell and all invoked commands.

`popd` [*+n*]
: Pops the directory stack, returning to the new top directory. With an argument, discards the *n* th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top. A synonym for `popd`, called `rd`, is provided for historical reasons. Its use is not recommended because it is not part of the standard BSD `csh` and may not be supported in future releases.

`pushd` [*name*] [*+n*]
: With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (using *cd*) and pushes the old current working directory (as in *csw*) onto the directory stack. With a numeric argument, *pushd* rotates the *n* th argument of the directory stack around to be the top element and changes to that directory. The members of the directory stack are numbered from the top starting at 0. A synonym for *pushd,* called *gd,* is provided for historical reasons. Its use is not recommended since it is not part of the standard BSD `csh` and may not be supported in future releases.

`rehash`
: Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories or if a systems programmer changes the contents of one of the system directories.

`repeat` *count command*
: The specified *command* (which is subject to the same restrictions as the *command* in the one-line `if` statement above) is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

`set`
`set` *name*
`set` *name*=*word*
`set` *name*[*index*]=*word*
`set` *name*= (*wordlist*)

The first form of **set** shows the value of all shell variables. Variables whose value is other than a single word print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index*th component of *name* to *word*; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and file-name expanded.

These arguments can be repeated to set multiple values in a single *set* command. Note, however, that variable expansion happens for all arguments before any setting occurs.

`setenv` *name value*

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variables, **USER**, **TERM**, and **PATH**, are automatically imported to and exported from the **csh** variables *user*, *term*, and *path*; there is no need to use **setenv** for these.

`shift` [*variable*]

If no argument is given, the members of **argv** are shifted to the left, discarding **argv[1]**. An error occurs if **argv** is not set or has less than two strings assigned to it. When *variable* is specified, *shift* performs the same function on the specified *variable*.

`source [-h]` *name*

csh reads commands from *name*. **source** commands can be nested, but if nested too deeply the shell may run out of file descriptors. An error in a **source** at any level terminates all nested **source** commands. Normally, input during **source** commands is not placed on the history list. The **-h** option can be used to place commands in the history list without being executing them.

`stop [%`*job* `... ]`

Stops the current (no argument) or specified jobs executing in the background.

`suspend`

Causes **csh** to stop as if it had been sent a *suspend* signal. Since **csh** normally ignores *suspend* signals, this is the only way to suspend the shell. This command gives an error message if attempted from a login shell.

`switch` (*string*)
`case` *str1* **:**
  . . .
breaksw
  . . .
default:
  . . .
breaksw
endsw

Each **case** label (*str1*) is successively matched against the specified *string* which is first command and file name expanded. The form of the **case** labels is the Pattern Matching Notation with the exception that non-matching lists in bracket expressions are not supported (see *regexp*(5)). If none of the labels match before a **default** label is found, the execution begins after the **default** label. Each **case** label and the **default** label must appear at the beginning of a line. The **breaksw** command causes execution to continue after the *endsw*. Otherwise, control may fall through **case** labels and **default** labels as in C. If no label matches and there is no default, execution continues after the **endsw**.

`time` [*command*]

When *command* is not specified, a summary of time used by this shell and its children is printed. If specified, the simple *command* is timed and a time summary as described under the **time** variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

`umask` [*value*]
: The current file creation mask is displayed (*value* not specified) or set to the specified *value*. The mask is given in octal. Common values for the mask are `002`, which gives all permissions to the owner and group and read and execute permissions to all others, or `022`, which gives all permissions to the owner, and only read and execute permission to the group and all others.

`unalias` *pattern*
: All aliases whose names match the specified *pattern* are discarded. Thus, all aliases are removed by `unalias *`. No error occurs if *pattern* does not match an existing alias.

`unhash`
: Use of the internal hash table to speed location of executed programs is disabled.

`unset` *pattern*
: All variables whose names match the specified *pattern* are removed. Thus, all variables are removed by `unset *`; this has noticeably undesirable side-effects. No error occurs if *pattern* matches nothing.

`unsetenv` *pattern*
: Removes all variables whose names match the specified *pattern* from the environment. See also the `setenv` command above and *printenv*(1).

`wait`
: Waits for all background jobs to terminate. If the shell is interactive, an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

`while` (*expression*)
`. . .`
`end`
: While the specified *expression* evaluates non-zero, the commands between the `while` and the matching `end` are evaluated. `break` and `continue` can be used to terminate or continue the loop prematurely. (The `while` and `end` must appear alone on their input lines.) If the input is a terminal (i.e., not a script), prompting occurs the first time through the loop as for the `foreach` statement.

`%`*job*
: Brings the specified job into the foreground.

`%`*job* `&`
: Continues the specified job in the background.

`@`
`@` *name*=*expression*
`@` *name*[*index*]=*expression*
: The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expression*. If the expression contains `<`, `>`, `&`, or `|`, at least this part of the expression must be placed within parentheses. The third form assigns the value of *expression* to the *index*th argument of *name*. Both *name* and its *index*th component must already exist.

: The operators `*=`, `+=`, etc., are available as in C. White space can optionally separate the *name* from the assignment operator. However, spaces are mandatory in separating components of *expression* which would otherwise be single words.

: Special postfix `++` and `--` operators increment and decrement *name*, respectively (e.g., `@ i++`).

Non-Built-In Command Execution

When a command to be executed is not a built-in command, `csh` attempts to execute the command via *exec*(2). Each word in the variable *path* names a directory in which the shell attempts to find the command (if the command does not begin with `/`). If neither `-c` nor `-t` is given, the shell hashes the names in these directories into an internal table so that an *exec* is attempted only in those directories where the command might possibly reside. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via `unhash`), or if `-c` or `-t` was given, or if any directory component of *path* does not begin with a `/`, the shell concatenates the directory name and the given command name to form a path name of a file which it then attempts to execute.

Commands placed inside parentheses are always executed in a subshell. Thus

```
(cd ; pwd)
```

prints the *home* directory then returns to the current directory upon completion, whereas:

```
cd ; pwd
```

remains in the *home* directory upon completion.

When commands are placed inside parentheses, it is usually to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary file, it is assumed to be a script file, which is a file of data for an interpreter that is executed as a separate process.

`csh` first attempts to load and execute the script file (see *exec*(2)). If the first two characters of the script file are **#!**, *exec*(2) expects an interpreter path name to follow and attempts to execute the specified interpreter as a separate process to read the entire script file.

If no **#! interpreter** is named, and there is an *alias* for the shell, the words of the *alias* are inserted at the beginning of the argument list to form the shell command. The first word of the *alias* should be the full path name of the command to be used. Note that this is a special, late-occurring case of *alias* substitution, which inserts words into the argument list without modification.

If no **#! interpreter** is named and there is no shell *alias*, but the first character of the file is **#**, the interpreter named by the **$shell** variable is executed (note that this normally would be **/bin/csh**, unless the user has reset **$shell**). If **$shell** is not set, **/bin/csh** is executed.

If no **!# interpreter** is named, and there is no shell alias, and the first character of the file is not **#**, **/bin/sh** is executed to interpret the script file.

History Substitutions

History substitutions enable you to repeat commands, use words from previous commands as portions of new commands, repeat arguments of a previous command in the current command, and fix spelling or typing mistakes in an earlier command.

History substitutions begin with an exclamation point (**!**). Substitutions can begin anywhere in the input stream, but *cannot* be nested. The exclamation point can be preceded by a backslash to cancel its special meaning. For convenience, an exclamation point is passed to the parser unchanged when it is followed by a blank, tab, newline, equal sign, or left parenthesis. Any input line that contains history substitution is echoed on the terminal before it is executed for verification.

Commands input from the terminal that consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The number of previous commands saved is controlled by the **history** variable. The previous command is always saved, regardless of its value. Commands are numbered sequentially from 1.

You can refer to previous events by event number (such as **!10** for event 10), relative event location (such as **!-2** for the second previous event), full or partial command name (such as **!d** for the last event using a command with initial character **d**), and string expression (such as **!?mic?** referring to an event containing the characters **mic**).

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, **!!** is a re-do; it refers to the previous command.

To select words from a command, use a colon (**:**) and a designator for the desired words after the event specification. The words of an input line are numbered from zero. The basic word designators are:

0	First word (i.e., the command name itself).
n	*n*th word.
^	First argument. (This is equivalent to **1**.)
$	Last word.
a-*b*	Range of words from *a* through *b*. Special cases are **-***y*, an abbreviation for "word 0 through word *y*"; and *x***-**, which means "word *x* up to, but not including, word **$**".

- `*` Range from the second word through the last word.
- `%` Used with a search sequence to substitute the immediately preceding matching word.

The colon separating the command specification from the word designator can be omitted if the argument selector begins with a **^**, **$**, *****, **-**, or **%**.

After word designator can be followed by a sequence of modifiers, each preceded by a colon. The following modifiers are defined:

- `h` Use only the first component of a path name by removing all following components.
- `r` Use the root file name by removing any trailing suffix (.xxx).
- `e` Use the file name's trailing suffix (`.`*xxx*) by removing the root name.
- `s` */l/r*
  substitute the value of *r* for the value *l* in the indicated command.
- `t` Use only the final file name of a path name by removing all leading path name components.
- `&` Repeat the previous substitution.
- `p` Print the new command but do not execute it.
- `q` Quote the substituted words, preventing further substitutions.
- `x` Like `q`, but break into words at blanks, tabs and newlines.
- `g` Use a global command as a prefix to another modifier to cause the specified change to be made globally. All words in the command are changed, one change per word, and each string enclosed in single quotes (`'`) or double quotes (`"`) is treated as a single word.

Unless preceded by a `g`, the modification is applied only to the first modifiable word. An error results if a substitution is attempted and cannot be completed (i.e., if you ask for a substitution of `!11` on a history buffer containing only 10 commands).

The left hand side of substitutions are strings; not regular expressions in the sense of HP-UX editors. Any character can be used as the delimiter in place of a slash (/). Use a backslash to quote a delimiter character if it is used in the *l* or *r* string. The character `&` in the right-hand side is replaced by the text from the left. A `\` also quotes `&`. A null *l* string uses the previous string either from an *l* or from a contextual scan string *s* in `!?`*s*`?`. The trailing delimiter in the substitution can be omitted if a new-line character follows immediately, as may the trailing `?` in a contextual scan.

A history reference can be given without an event specification (as in `!$`). In this case, the reference is to the previous command unless a previous history reference occurred on the same line, in which case this form repeats the previous reference. Thus

```
!?foo?^ !$
```

gives the first and last arguments from the command matching `?foo?`.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a circumflex (^). This is equivalent to `!:s^`, providing a convenient shorthand for substitutions on the text of the previous line. Thus `^lb^lib` fixes the spelling of `lib` in the previous command.

Finally, a history substitution can be enclosed within curly braces `{ }` if necessary to insulate it from the characters which follow. Thus, after

```
ls -ld ~paul
```

one could execute `!{l}a` to do

```
ls -ld ~paula
```

while `!la` would look for a command starting with `la`.

Quoting with Single and Double Quotes

The quotation of strings by single quotes (') and double quotes (") can be used to prevent all or some of the remaining substitutions. Strings enclosed in single quotes are protected from any further interpretation. Strings enclosed in double quotes are still variable- and command-expanded as described below.

In both cases the resulting text becomes (all or part of) a single word. Only in one special case (see *Command Substitution* below) does a double-quoted string yield parts of more than one word; single-quoted strings never do.

Alias Substitution

`csh` maintains a list of aliases that can be established, displayed, and modified by the `alias` and `unalias` commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, the text which is the alias for that command is reread with the history mechanism available as if that command was the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, the argument list is left unchanged.

Thus, if the alias for `ls` is `ls -l`, the command `ls /usr` maps to `ls -l /usr`, leaving the argument list undisturbed. Similarly, if the alias for `lookup` was `grep !^ /etc/passwd`, `lookup bill` maps to `grep bill /etc/passwd .`

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the re-formed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus:

```
alias print 'pr \!* | lp'
```

makes a command that uses *pr*(1) to print its arguments on the line printer.

Expressions

Some of the built-in commands take expressions in which the operators are similar to those of C, with the same precedence. These expressions appear in the `@`, `exit`, `if`, and `while` commands. The following operators are available (shown in order of increasing precedence):

```
|| && | ^ & == != =~ !~ <= >= < > << >> + - * / % ! ~ ( )
```

The following list shows the grouping of these operators. The precedence decreases from top to bottom in the list:

```
* / %
+ -
<< >>
<= >= < >
== != =~ !~
```

The operators `==`, `!=`, `=~`, and `!~` compare their arguments as strings; all others operate on numbers. The operators `=~` and `!~` are similar to `!=` and `==`, except that the right-hand side is a *pattern* (containing `*`s, `?`s, and instances of `[...]`) against which the left hand operand is matched. This reduces the need for use of the `switch` statement in shell scripts when all that is really needed is pattern matching.

Strings beginning with `0` are considered octal numbers. Null or missing arguments are considered `0`. The result of all expressions are strings that represent decimal numbers. It is important to note that no two components of an expression can appear in the same word. These components should be surrounded by spaces except when adjacent to components of expressions that are syntactically significant to the parser: `-`, `&`, `|`, `<`, `>`, `(`, and `)`.

Also available in expressions as primitive operands are command executions enclosed in curly braces (`{ }`) and file enquiries of the form *-l filename*, where *l* is one of:

`r`	read access
`w`	write access
`x`	execute access
`e`	existence
`o`	ownership
`z`	zero size
`f`	plain file
`d`	directory

The specified *filename* is command- and file-name expanded then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, all inquiries return false (0). Command executions succeed, returning true, if the command exits with status 0; otherwise they fail, returning false. If more detailed status information is required, the command should be executed outside of an expression and the `status` variable examined.

Control of the Flow

`csh` contains a number of commands that can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip parts of its input and, due to the implementation, restrict the placement of some of the commands.

The `foreach`, `switch`, and `while` statements, as well as the `if-then-else` form of the `if` statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward `goto`s succeed on non-seekable inputs.)

Signal Handling

`csh` normally ignores *quit* signals. Jobs running in background mode are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. `csh`'s handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file `.logout`.

Command Line Parsing

`csh` splits input lines into words at blanks and tabs. The following exceptions (parser metacharacters) are considered separate words:

&	ampersand;
|	vertical bar;
;	semicolon;
<	less-than sign;
>	greater-than sign;
(	left parenthesis;
)	right parenthesis;
&&	double ampersand;
||	double vertical bar;
<<	double less-than sign;
>>	double greater-than sign;
#	comment delimiter

The backslash (\) removes the special meaning of these parser metacharacters. A parser metacharacter preceded by a backslash is interpreted as its ASCII value. A newline character (ASCII 10) preceded by a backslash is equivalent to a blank.

Strings enclosed in single or double quotes form parts of a word. Metacharacters in these strings, including blanks and tabs, do not form separate words. Within pairs of backslashes or quotes, a newline preceded by a backslash gives a true newline character.

When `csh`'s input is not a terminal, the # charactercharacter introduces a comment terminated by a newline.

CSH VARIABLES

`csh` maintains a set of variables. Each variable has a value equal to zero or more strings (words). Variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter. The value of a variable may be displayed and changed by using the `set` and `unset` commands. Some of the variables are Boolean, that is, the shell does not care what their value is, only whether they are set or not.

Some operations treat variables numerically. The at sign (@) command permits numeric calculations to be performed and the result assigned to a variable. The null string is considered to be zero, and any subsequent words of multi-word values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable expansion is performed keyed by the dollar sign ($)`character.` Variable expansion can be prevented by preceding the dollar sign with a backslash character (\) except within double quotes (") where substitution **always** occurs. Variables are never expanded if enclosed in single quotes. Strings quoted by single quotes are interpreted later (see *Command Substitution*) so variable substitution does not occur there until later, if at all. A dollar sign is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together.

Unless enclosed in double quotes or given the **:q** modifier, the results of variable substitution may eventually be command and file name substituted. Within double quotes, a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variable's value separated by blanks. When the **:q** modifier is applied to a substitution, the variable expands to multiple words with each word separated by a blank and quoted to prevent later command or file name substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable that is not set.

$*variable_name*
${*variable_name***}**
: When interpreted, this sequence is replaced by the words of the value of the variable *variable_name*, each separated by a blank. Braces insulate *variable_name* from subsequent characters that would otherwise be interpreted to be part of the variable name itself.
If *variable_name* is not a **csh** variable, but is set in the environment, that value is used. Non-**csh** variables cannot be modified as shown below.

$*variable_name*[*selector*]
${*variable_name*[*selector*]**}**
: This modification selects only some of the words from the value of *variable_name*. The selector is subjected to variable substitution, and can consist of a single number or two numbers separated by a dash. The first word of a variable's value is numbered **1**. If the first number of a range is omitted it defaults to **1**. If the last member of a range is omitted it defaults to the total number of words in the variable (**$#***variable_name*). An asterisk metacharacter used as a selector selects all words.

$#*variable_name*
${#*variable_name***}**
: This form gives the number of words in the variable, and is useful for forms using a [*selector*] option.

$0 This form substitutes the name of the file from which command input is being read. An error occurs if the file name is not known.

$*number*
${*number***}**
: This form is equivalent to an indexed selection from the variable **argv** (**$argv[***number***]**).

$* This is equivalent to selecting all of *argv* (**$argv[*]**).

The modifiers **:h**, **:t**, **:r**, **:q**, and **:x** can be applied to the substitutions above, as can CR :gh , CR :gt , and CR :gr . If curly braces (**{ }**) appear in the command form, the modifiers must appear within the braces. *The current implementation allows only one* **:** *modifier on each* **$d** *expansion*.

The following substitutions cannot be modified with **:** modifiers:

$?*variable_name*
${?*variable_name***}**
: Substitutes the string **1** if *variable_name* is set, **0** if it is not.

$?0 Substitutes **1** if the current input file name is known, **0** if it is not.

$$ Substitutes the (decimal) process number of the (parent) shell.

$< Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Pre-Defined and Environment Variables

The following variables have special meaning to the shell. Of these **autologout**, **argv**, **cwd**, **home**, **path**, **prompt**, **shell**, and **status** are always set by the shell. Except for **cwd** and **status**, this setting occurs only at initialization (initial execution of **csh**). These variables are not modified unless modified explicitly by the user.

csh copies the HP-UX environment variable USER into the shell variable **user**, the environment variable **TERM** into **term**, the environment variable **HOME** into **home**, and **PATH** into **path**. **csh** copies these values back into the environment whenever the **csh** variables are reset.

In a windowed environment, if **csh** detects that the window has changed size, **csh** sets the environment variables **LINES** and **COLUMNS** to match the new window size.

argv This variable is set to the arguments of the **csh** command statement. It is from this variable that positional parameters are substituted; i.e., **$1** is replaced by **$argv[1]**, etc.

cdpath This variable gives a list of alternate directories searched to find subdirectories in *chdir* commands.

cwd This variable contains the absolute path name of the current working directory. Whenever changing directories (using *cd*), this variable is updated.

echo This variable is set by the **-x** command line option. If set, all built-in commands and their arguments are echoed to the standard output device just before being executed. Built-in commands are echoed before command and file name substitution, since these substitutions are then done selectively. For non-built-in commands, all expansions occur before echoing.

hidden If set, **csh** includes unmatched CDFs in expansions (see *cdf*(4)). Also, unmatched CDFs are displayed with a trailing **+** when using the **control-D** command to display matching file names (see the *CSH UTILITIES* section). By default, **hidden** is not set.

history This variable is used to create the command history buffer and to set its size. If this variable is not set, no command history is maintained and history substitutions cannot be made. Very large values of **history** can cause shell memory overflow. Values of 10 or 20 are normal. All commands, executable or not, are saved in the command history buffer.

home This variable contains the absolute path name to your home directory. The variable **home** is initialized from the HP-UX environment. File name expansion of tilde (~) refers to this variable.

ignoreeof If set, **csh** ignores end-of-file characters from input devices that are terminals. **csh** exits normally when it encounters the end-of-file condition (CTRL-D typed as the first character on a command line). Setting *ignoreeof* prevents the current shell from being killed by an accidental (CTRL-D. However, to prevent an infinite loop of EOF input, **csh** terminates if it receives 26 consecutive EOFs.

mail This variable contains a list of the files where **csh** checks for your mail. **csh** periodically (default is 10 minutes) checks this variable before producing a prompt upon command completion. If the variable contains a file name that has been modified since the last check (resulting from mail being put in the file), **csh** prints **You have new mail.**

If the first word of the value of **mail** is numeric, that number specifies a different mail checking interval in seconds.

If multiple mail files are specified, the shell says **New mail in** *file_name*, where *file_name* is the file containing the mail.

noclobber This variable places restrictions on output redirection to ensure that files are not accidentally destroyed, and that commands using append redirection (>>) refer to existing files.

noglob If set, file name expansion is inhibited. This is most useful in shell scripts that are not dealing with file names, or after a list of file names has been obtained and further expansions are not desirable.

`nonomatch`	If set, it is no longer an error for a file name expansion to not match any existing files. If there is no match, the primitive pattern is returned. It is still an error for the primitive pattern to be malformed. For example, `'echo ['` still gives an error.
`notify`	If set, `csh` notifies you immediately (through your standard output device) of background job completions. The default is `unset` (indicate job completions just before printing a prompt).
`path`	Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies your current working directory. If there is no *path* variable, only full path names can be executed. When *path* is not set and when users do not specify full path names, `csh` searches for the command through the directories `.` (current directory), `/bin`, and `/usr/bin`. A `csh` which is given neither the `-c` nor the `-t` option normally hashes the contents of the directories in the `path` variable after reading `.cshrc`, and each time the `path` variable is reset. If new commands are added to these directories while the shell is active, it is necessary to execute `rehash` for `csh` to access these new commands.
`prompt`	This variable lets you select your own prompt character string. The prompt is printed before each command is read from an interactive terminal input. If a `!` appears in the string, it is replaced by the current command history buffer event number unless a preceding `\` is given. The default prompt is the percent sign (%) for users and the # character for the super-user.
`shell`	This variable contains the name of the file in which the `csh` program resides. This variable is used in forking shells to interpret files that have their execute bits set but which are not executable by the system. (See the description of *Non-Built-In Command Execution*).
`status`	This variable contains the status value returned by the last command. If the command terminated abnormally, 0200 is added to the status variable's value. Built-in commands which terminated abnormally return exit status `1`, and all other built-in commands set status to `0`.
`time`	This variable contains a numeric value that controls the automatic timing of commands. If set, `csh` prints, for any command taking more than the specified number of cpu seconds, a line of information to the standard output device giving user, system, and real execution times plus a utilization percentage. The utilization percentage is the ratio of user plus system times to real time. This message is printed after the command finishes execution.
`verbose`	This variable is set by the `-v` command line option. If set, the words of each command are printed on the standard output device after history substitutions have been made.

Command and File name Substitution

The remaining substitutions, command and file name substitution, are applied selectively to the arguments of built-in commands. This means that portions of expressions that are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command Substitution

Command substitution is indicated by a command enclosed in grave accents (` ...`). The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded; this text then replacing the original string. Within double quotes, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

File name Substitution

Each command *word* is processed as a pattern for file name substitution, also known as **globbing**, and replaced with a sorted list of file names which match the pattern. The form of the patterns is the Pattern Matching Notation defined by *regexp*(5) with the following exceptions:

- Non-matching lists in bracket expressions are not supported.
- In a list of words specifying file name substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match.
- The metanotation **a{b,c,d}e** is a shorthand for "abe ace ade". Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus:

  ```
  ~source/s1/{oldls,ls}.c
  ```

  expands to

  ```
  /usr/source/s1/oldls.c /usr/source/s1/ls.c
  ```

  whether or not these files exist, without any chance of error if the home directory for `source` is `/usr/source`. Similarly,

  ```
  ../{memo,*box}
  ```

  might expand to

  ```
  ../memo ../box ../mbox
  ```

  (Note that **memo** was not sorted with the results of matching ***box**.) As a special case, **{**, **}**, and **{ }** are passed undisturbed.

Input/Output

The standard input and standard output of a command can be redirected with the following syntax:

< *name* — Open file *name* (which is first variable, command and file name expanded) as the standard input.

<< *word* — Read the shell input up to a line which is identical to *word*. *word* is not subjected to variable, file name or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, **'**, or **`** appears in *word*, variable and command substitution is performed on the intervening lines, allowing **\\** to quote **$**, **\\** and **`**. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> *name*
>! *name*
>& *name*
>&! *name* — The file *name* is used as standard output. If the file does not exist, it is created; if the file exists, it is truncated, and its previous contents are lost.

If the variable `noclobber` is set, the file must not exist or be a character special file (e.g., a terminal or `/dev/null`) or an error results. This helps prevent accidental destruction of files. In this case the exclamation point (**!**) forms can be used to suppress this check.

The forms involving the ampersand character (**&**) route the standard error into the specified file as well as the standard output. *name* is expanded in the same way as **<** input file names are.

>> *name*
>>& *name*
>>! *name*
>>&! *name* — Uses file *name* as standard output the same as **>**, but appends output to the end of the file. If the variable `noclobber` is set, it is an error for the file not to exist unless one of the **!** forms is given. Otherwise, it is similar to **>**.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands executed from a shell script have no access to the text of the commands by default; rather they receive the original standard input of the shell. The **<<** mechanism should be used to present inline data. This permits shell scripts to function as components of pipelines and allows the shell to block-read its input.

Diagnostic output can be directed through a pipe with the standard output. Simply use the form `|&` rather than `|` by itself.

CSH UTILITIES

File Name Completion

In typing file names as arguments to commands, it is no longer necessary to type a complete name, only a unique abbreviation is necessary. When you want the system to try to match your abbreviation, press the ESC key. The system then completes the file name for you, echoing the full name on your terminal. If the abbreviation does not match an available file name, the terminal's bell is sounded. The file name may be partially completed if the prefix matches several longer file names. In this case, the name is extended up to the ambiguous deviation, and the bell is sounded.

File name completion works equally well when other directories are addressed. In addition, the tilde (~) convention for home directories is understood in this context.

Viewing a File or Directory List

At any point in typing a command, you can request "what files are available" or "what files match my current specification". Thus, when you have typed:

```
% cd ~speech/data/bench/fritz/
```

you may wish to know what files or subdirectories exist (in **~speech/data/bench/fritz**), without aborting the command you are typing. Typing CTRL-D at this point lists the files available. Files are listed in multicolumn format, sorted by column. Directories and executable files are identified by a trailing / and *, respectively. Once printed, the command is re-echoed for you to complete. Additionally, you may want to know which files match a prefix, the current file specification so far. If you had typed:

```
% cd ~speech/data/bench/fr
```

followed by a CTRL-D, all files and subdirectories whose prefix was **fr** in the directory **~speech/data/bench** would be printed. Notice that the example before was simply a degenerate case of this with a null trailing file name. (The null string is a prefix of all strings.) Notice also that a trailing slash is required to pass to a new sub-directory for both file name completion and listing. Note that the degenerate case

```
% ~^D
```

prints a full list of login names on the current system.

Command Name Recognition

Command name recognition and completion works in the same manner as file name recognition and completion above. The current value of the environment variable **PATH** is used in searching for the command. For example

```
% newa [Escape]
```

might expand to

```
% newaliases
```

Also,

```
% new [Control]-[D]
```

lists all commands (along **PATH**) that begin with **new**. As an option, if the shell variable **listpathnum** is set, a number indicating the index in **PATH** is printed next to each command on a [Control]-[D] listing.

Autologout

A new shell variable has been added called **autologout**. If the terminal remains idle (no character input) at the shell's top level for a number of minutes greater than the value assigned to **autologout**, you are automatically logged off. The **autologout** feature is temporarily disabled while a command is executing. The initial value of **autologout** is 60. If unset or set to 0, **autologout** is entirely disabled.

Command Line Control

A `^R` re-prints the current command line; `^W` erases the last word entered on the current command line.

Sanity

C shell restores your terminal to a sane mode if it appears to return from some command in raw, cbreak, or noecho mode.

Saving Your History Buffer

`csh` has the ability to save your history list between login sessions. If the shell variable **`savehist`** is set to a number, that number of command events from your history list is saved. For example, placing the line

```
    set history=10 savehist=10
```

in your `.cshrc` file maintains a history buffer of length 10 and saves the entire list when you logout. When you log back in, the entire buffer is restored. The commands are saved in the file `.history` in your login directory.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name substitution.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as letters, and the characters matched by character class expressions in pattern matching notation.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, `csh` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

The `.cshrc` file should be structured such that it cannot generate any output on standard output or standard error, including occasions when it is invoked without an affiliated terminal. *rcp*(1) causes `.cshrc` to be sourced, and any output generated by this file, even to standard error causes problems. Commands such as *stty*(1) should be placed in `.login`, not in `.cshrc`, so that their output cannot affect *rcp*(1).

`csh` has certain limitations. Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves file name expansion is limited to one-sixth the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list.

To detect looping, the shell restricts the number of `alias` substitutions on a single line to 20.

When a command is restarted from a stop, `csh` prints the directory it started in if it is different from the current directory; this can be misleading (i.e., wrong) because the job may have changed directories internally.

Shell built-in functions are not stoppable/restartable. Command sequences of the form `a ; b ; c` are also not handled gracefully when stopping is attempted. If you interrupt b, the shell then immediately executes c. This is especially noticeable if this expansion results from an `alias`. It suffices to place the sequence of commands in parentheses to force it into a subshell; i.e., `( a ; b ; c )`.

Because of the signal handling required by csh, interrupts are disabled just before a command is executed, and restored as the command begins execution. There may be a few seconds delay between when a command is given and when interrupts are recognized.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by `?`, are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

Terminal type is examined only the first time you attempt recognition.

To list all commands on the system along **PATH**, enter [Space]-[Ctrl]-[D].

The csh metasequence !~ does not work.

In an international environment, character ordering is determined by the setting of LC_COLLATE, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particulary when using range expressions in file name generation patterns. For example, the command,

```
rm [a-z]*
```

might be expected to match all file names beginning with a lowercase alphabetic character. However, if dictionary ordering is specified by LC_COLLATE, it would also match file names beginning with an uppercase character (as well as those beginning with accented letters). Conversely, it would fail to match letters collated after **z** in languages such as Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form:

```
rm [[:lower:]]*
```

This uses **LC_CTYPE** to determine character classes and works predictably for all supported languages and codesets. For shell scripts produced on non-internationalized systems (or without consideration for the above dangers), it is recommended that they be executed in a non-NLS environment. This requires that **LANG**, **LC_COLLATE**, etc., be set to "C" or not set at all.

csh implements command substitution by creating a pipe between itself and the command. If the root file system is full, the substituted command cannot write to the pipe. As a result, the shell receives no input from the command, and the result of the substitution is null. In particular, using command substitution for variable assignment under such circumstances results in the variable being silently assigned a NULL value.

Relative path changes (such as `cd ..`), when in a symbolically linked directory, cause **csh**'s knowledge of the working directory to be along the symbolic path instead of the physical path.

Prior to HP-UX Release 9.0, **csh**, when getting its input from a file, would exit immediately if unable to execute a command (such as if it was unable to find the command). Beginning at Release 9.0, **csh** continues on and attempts to execute the remaining commands in the file. However, if the old behavior is desired for compatibility purposes, set the environment variable **EXITONERR** to 1.

AUTHOR

csh was developed by the University of California, Berkeley and HP.

FILES

~/.cshrc	A csh script sourced (executed) at the beginning of execution by each shell. See WARNINGS
~/.login	A csh script sourced (executed) by login shell, after **.cshrc** at login.
~/.logout	A csh script sourced (executed) by login shell, at logout.
/etc/passwd	Source of home directories for **~name**.
/bin/sh	Standard shell, for shell scripts not starting with a **#**.
/etc/csh.login	A csh script sourced (executed) before **~/.cshrc** and **~/.login** when starting a csh login (analogous to **/etc/profile** in the Bourne shell).
/tmp/sh*	Temporary file for **<<**.

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), umask(2), wait(2), tty(7), a.out(4), environ(5), lang(5), regexp(5).

C Shell tutorial in *Shells Users Guide* .

NAME
csplit - context split

SYNOPSIS
`csplit` [`-s`] [`-k`] [`-f` *prefix*] [`-n` *number*] *file arg1* [... *argn*]

DESCRIPTION
`csplit` reads *file*, separates it into *n*+1 sections as defined by the arguments *arg1* ... *argn*, and places the results in separate files. The maximum number of arguments (*arg1* through *argn*) allowed is 99 unless the `-n` *number* option is used to allow for more output file names. If the `-f` *prefix* option is specified, the resulting filenames are *prefix*`00` through *prefixNN* where *NN* is the two-digit value of *n* using a leading zero if *n* is less than 10. If the `-f` *prefix* option is not specified, the default filenames **xx00** through **xx***NN* are used. *file* is divided as follows:

Default Filename	Prefixed Filename	Contents
xx00	*prefix*00	From start of *file* up to (but not including) the line referenced by *arg1*.
xx01	*prefix*01	From the line referenced by *arg1* up to the line referenced by *arg2*.
		.
		.
		.
xx*NN*	*prefixNN*	From the line referenced by *argn* to end of *file*.

If the *file* argument is `-`, standard input is used.

`csplit` supports the Basic Regular Expression syntax (see *regexp*(5)).

Options
`csplit` recognizes the following options:

`-s`	Suppress printing of all character counts (`csplit` normally prints the character counts for each file created).
`-k`	Leave previously created files intact (`csplit` normally removes created files if an error occurs).
`-f` *prefix*	Name created files *prefix*`00` through *prefixNN* (default is `xx00` through **xx***NN*.
`-n` *number*	The output file name suffix will use *number* digits instead of the default **2**. This allows creation of more than 100 output files.

Arguments (*arg1* through *argn*) to `csplit` can be any combination of the following:

/regexp/	Create a file containing the section from the current line up to (but not including) the line matching the regular expression *regexp*. The new current line becomes the line matching *regexp*.
/regexp/+n */regexp/-n*	Create a file containing the section from the current line up to (but not including) the *n*th before (*-n*) or after (*+n*) the line matching the regular expression *regexp*. (e.g., `/Page/-5`). The new current line becomes the line matching *regexp* lines.
%regexp%	equivalent to */regexp/*, except that no file is created for the section.
line_number	Create a file from the current line up to (but not including) *line_number*. The new current line becomes *line_number*.
{*num*}	Repeat argument. This argument can follow any of the above argument forms. If it follows a *regexp* argument, that argument is applied *num* more times. If it follows *line_number*, the file is split every *line_number* lines for *num* times from that point until end-of-file is reached or *num* expires.

Enclose in appropriate quotes all *regexp* arguments containing blanks or other characters meaningful to the shell. Regular expressions must not contain embedded new-lines. `csplit` does not alter or remove the original file; it is the user's responsibility to remove it when appropriate.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions.

`LC_CTYPE` determines the characters matched by character class expressions in regular expressions.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `csplit` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

DIAGNOSTICS

Messages are self explanatory except for:

```
arg - out of range
```

which means that the given argument did not reference a line between the current position and the end of the file. This warning also occurs if the file is exhausted before the repeat count is.

EXAMPLES

Create four files, `cobol00` through `cobol03`. After editing the “split” files, recombine them back into the original file, destroying its previous contents.

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

Perform editing operations

```
cat cobol0[0-3] > file
```

Split a file at every 100 lines, up to 10,000 lines (100 files). The `-k` option causes the created files to be retained if there are fewer than 10,000 lines (an error message is still printed).

```
csplit -k file 100 '{99}'
```

Assuming that `prog.c` follows the normal C coding convention of terminating routines with a `}` at the beginning of the line, create a file containing each separate C routine (up to 21) in `prog.c`.

```
csplit -k prog.c '%main(%' '/^}/+1' '{20}'
```

SEE ALSO

sh(1), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

`csplit`: SVID2, XPG2, XPG3

NAME

ct - spawn getty to a remote terminal (call terminal)

SYNOPSIS

`ct` [`-w`*n*] [`-x`*n*] [`-h`] [`-v`] [`-s` *speed*] *telno* ...

DESCRIPTION

`ct` dials *telno*, the telephone number of a modem that is attached to a terminal, and spawns a *getty*(1M) process to that terminal.

`ct` tries each line listed in file `/usr/lib/uucp/Devices` until it finds an available line with appropriate attributes or runs out of entries. If no lines are free, `ct` asks whether it should wait for a line, and if so, how many minutes it should wait before giving up. `ct` searches again for an available line at one-minute intervals until the specified limit is exceeded. Note that normally, `ct` disconnects the current tty line, so that the line can answer the incoming call. This is because `ct` assumes that the current tty line is connected to the terminal to spawn the `getty` process.

The *telno* argument specifies the telephone number, which can be composed of characters `0` thru `9`, `-`, `=`, `*`, and `#`. Use equal signs to signify secondary dial tones and minus signs for delays at appropriate places. The maximum length of *telno* is 31 characters. If more than one telephone number is specified, `ct` tries each in succession until one answers; this is useful for specifying alternate dialing paths.

Options

`ct` recognizes the following options and command-line arguments:

`-w`*n*	Instruct `ct` to wait for a line a maximum of *n* number of minutes, if lines are busy. If this option is specified, `ct` does not query the user about whether to wait for a line.
`-x`*n*	Produce detailed output from program execution on the standard error output. This option is used for debugging. The debugging level *n* is a single digit; the most useful value is `-x9`.
`-h`	Prevent `ct` from disconnecting ("hanging up") the current tty line. This option is necessary if the user is using a different tty line than the one used by `ct` to spawn the `getty`.
`-v`	Verbose mode. The `-v` option is used with the `-h` option and causes `ct` to send a running narrative to the standard error output stream.
`-s`*speed*	Set the data rate where *speed* is expressed in baud. The default rate is `1200`.

After the user on the destination terminal logs out, `ct` prompts, `Reconnect?` If the response begins with the letter `n` the line is dropped. Otherwise, `getty` is restarted and the `login:` prompt is printed.

Of course, the destination terminal must be attached to a modem that can automatically answer incoming calls.

DEPENDENCIES

HP Clustered Environment

`ct` is not supported on cluster client nodes in an HP Cluster.

FILES

`/usr/adm/ctlog`
`/usr/lib/uucp/Devices`

SEE ALSO

cu(1), getty(1M), login(1), uucp(1).

UUCP tutorial in *Remote Access Users Guide*.

NAME

ctags - create a tags file

SYNOPSIS

`ctags` [`-xvFBatwu`] *files* ...

DESCRIPTION

`ctags` makes a tags file for *ex*(1) (or *vi*(1)) from the specified C, Pascal and FORTRAN sources. A **tags** file gives the locations of specified objects (for C, functions, macros with argments, and typedefs; Pascal, procedures, programs and functions; FORTRAN, subroutines, programs and functions) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Output is sorted in ascending collation order (see Environment Variables below). All objects except C *typedefs* are searched with a pattern, *typedefs* with a line number. Specifiers are given in separate fields on the line, separated by spaces or tabs. Using the *tags* file, `ex` can quickly find these objects' definitions.

`-x` Cause `ctags` to print a simple function index. This is done by assembling a list of function names, file names on which each function is defined, the line numbers where each function name occurs, and the text of each line. The list is then printed on the standard output. No *tags* file is created or changed.

`-v` Produce a page index on the standard output. This listing contains the function name, file name, and page number within that file (assuming 56-line pages to match *pr*(1)).

Files whose name ends in `.c` or `.h` are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C definitions.

Other options are:

`-F` Use forward searching patterns (`/`...`/`) (default).

`-B` Use backward searching patterns (`?`...`?`).

`-a` Add the information from the files to the *tags* file. Unlike re-building the *tags* file from the original files, this can cause the same symbol to be entered twice in the *tags* file. This option should be used with caution and then only in very special circumstances.

`-t` Create tags for typedefs.

`-w` Suppress warning diagnostics.

`-u` Update the specified files in *tags*; that is, all references to those files are deleted, and the new values are added to the file as in `-a` above. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

The tag `main` is treated specially in C programs. The tag formed is created by adding `M` to the beginning of name of the file, with any trailing `.c` removed, and leading pathname components also removed. This makes use of `ctags` practical in directories with more than one program.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

LC_CTYPE determines the interpretation of the single- and/or multi-byte characters within comments and string literals.

If **LC_COLLATE** or **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `ctags` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

DIAGNOSTICS

`Too many entries to sort.`
An attempt to get additional heap space failed; the sort could not be performed.

`Unexpected end of function in file` *file*`, line` *line*`.`
The tags file may be incorrect.

A } character was found unexpectedly in the first column. This can lead to incorrect entries in the *tags* file. `ctags` expects the source code to conform to the format as described on *cb*(1).

`Duplicate entry in file` *file*`, line` *line*`: name. Second entry ignored.`
The same name was detected twice in the same file. A *tags* entry was made only for the first name found.

`Duplicate entry in files` *file1* `and` *file2*`:` *name* `(Warning only).`
The same name was detected in two different files. A *tags* entry was made only for the first name found.

WARNINGS

Recognition of `functions`, `subroutines`, and `procedures` for FORTRAN and Pascal is done in a very simple way. No attempt is made to deal with block structure; if there are two Pascal procedures in different blocks with the same name, a warning message is generated.

The method of deciding whether to look for C or Pascal and FORTRAN functions is an approximation, and can be fooled by unusual programs.

`ctags` does not know about `#ifdefs` and Pascal types.

It relies on the input being well formed to detect *typedefs*.

Use of `-tx` shows only the last line of typedefs.

ex is naive about *tags* files with several identical tags; it simply chooses the first entry its (non-linear) search finds with that tag. Such files can be created with either the `-u` or `-a` options or by editing a *tags* file.

If more than one (function) definition appears on a single line, only the first definition is indexed.

AUTHOR

`ctags` was developed by the University of California, Berkeley.

FILES

`tags`	output tags file
`OTAGS`	temporary file used by `-u`

SEE ALSO

cb(1), ex(1), vi(1).

NAME

cu - call another (UNIX) system; terminal emulator

SYNOPSIS

`cu` [`-s` *speed*] [`-l` *line*] [`-h`] [`-q`] [`-t`] [`-d` *level*] [`-e` | `-o`] [`-m`] [`-n`] [*telno* | *systemname* | `dir`]

DESCRIPTION

`cu` calls up another system, which is usually a UNIX operating system, but can be a terminal or a non-UNIX operating system. `cu` manages all interaction between systems, including possible transfers of ASCII files.

Options

`cu` recognizes the following options and command-line arguments:

`-s`*speed*	Specify the transmission speed (110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200). The default value is 300.
`-l`*line*	Specify a device name to use as the communication line. This can be used to override searching for the first available line having the right speed. When the `-l` option is used without the `-s` option, the speed of a line is obtained from file `/usr/lib/uucp/Devices`. When the `-l` and `-s` options are used simultaneously, `cu` searches `/usr/lib/uucp/Devices` to determine whether the requested speed for the requested line is available. If so, the connection is made at the requested speed; otherwise, an error message is printed and the call is not made. The specified device is usually a directly connected asynchronous line (such as `/dev/ttyab`). In this case, a telephone number is not required, but the string `dir` can be used to specify that a dialer is not required. If the specified device is associated with an auto-dialer, a telephone number must be provided.
`-h`	Emulate local echo, supporting calls to other computer systems that expect terminals to be set to half-duplex mode.
`-q`	Use ENQ/ACK handshake (remote system sends ENQ, `cu` sends ACK.)
`-t`	Used when dialing an ASCII terminal that has been set to auto-answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.
`-d`*level*	Print diagnostic traces. *level* is a number from 0-9, where higher *level*s produce more detail in the diagnostic messages.
`-e (-o)`	Generate even (odd) parity for data sent to the remote.
`-m`	Specify a direct line that has modem controls. Modem controls are ignored by `cu`.
`-n`	Cause the telephone number that `cu` dials to be requested interactively from the user rather than taking it from the command line.
telno	When using an automatic dialer, *telno* is the telephone number, with equal signs for secondary dial tone or minus signs for delays appropriately placed in the *telno* string.
systemname	A UUCP system name can be used instead of a telephone number (see *uucp*(1)); in this case, `cu` obtains an appropriate direct line or telephone number from `/usr/lib/uucp/Systems` (including appropriate baud rate). `cu` dials each telephone number or direct line for *systemname* in the `Systems` file until a connection is made or all the entries are tried.
`dir`	Using `dir` ensures that `cu` uses the line specified by the `-l` option.

After making the connection, `cu` runs as two processes:

- *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system;
- *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output.

Normally, an automatic DC3/DC1 protocol is used to control input from the remote to ensure that the buffer is not overrun. "Prompt handshaking" can be used to control transfer of ASCII files to systems that have no type-ahead capability but require data to be sent only after a prompt is given. This is described in detail below. Lines beginning with ~ have special meanings.

Transmit Process Commands

The *transmit* process interprets the following commands:

`~.`, `~..`	Terminate the conversation. On hard-wired lines, `~.` sends several EOF characters to log out the session, whereas `~..` suppresses the EOF sequence. In general the remote hard-wired machine is unaware of the disconnect if `~..` is used. On dial-up connections, `~.` and `~..` do not differ.
`~!`	Escape to an interactive shell on the local system.
`~!`*cmd* ...	Run *cmd* on the local system (via `sh -c`).
`~&`	Similar to `~!` but kill the receive process, restarting it upon return from the sheii. This is useful for invoking sub-processes that read from the communication line where the receive process would otherwise compete for input.
`~&`*cmd* ...	Run *cmd* on the local system (via `sh -c`) and kill the receive process, restarting it later.
`~|` *cmd*	Pipe incoming data from the remote system through the standard input to *cmd* on the local system. To terminate, reset with either a `~&` or `~|` command.
`~|`	Resets the receive process following a `~|cmd` command.
`~$`*cmd* ...	Run *cmd* locally and send its output to the remote system.
`~%cd`	Change the directory on the local system. *Note*: `~!cd` causes the command to be run by a sub-shell, causing a return to the current directory upon completion.

`~%take` *remote_source_file* [*local_destination_file*]
: Copy file *remote_source_file* from the remote system to file *local_destination_file* on the local system. If *local_destination_file* is not specified, the *remote_source_file* argument is used in both places.

`~%put` *local_source_file* [*remote_destination_file*]
: Copy file *local_source_file* on local system to file *remote_destination_file* on remote system. If *remote_destination_file* is not specified, the *local_source_file* argument is used in both places.

`~~` ...	Send the line `~` ... to the remote system. If you use `cu` on the remote system to access a third remote system, send `~~.` to cause the second remote `cu` to exit.
`~%break`	Transmit a BREAK to the remote system.
`~%nostop`	Toggle between DC3/DC1 input control protocol and no input control. This is useful if the remote system does not respond properly to the DC3 and DC1 characters.
`~%<`*file*	Send the contents of the local file to the remote system using *prompt handshaking*. The specified file is read one line at a time, and each line is sent to the remote system when the *prompt sequence* is received. If no prompt is received by the time the *prompt timeout* occurs, the line is sent anyway. If the timeout is set to 0 seconds, or if the first character in the prompt sequence is a null character (`^@`), the handshake always appears to be satisfied immediately, regardless of whether or not the remote system generates a prompt. This capability is intended mainly to facilitate transfer of ASCII files from HP-UX to an HP 3000 system running MPE. This is usually accomplished by running the MPE `FCOPY` utility and giving the command `from=;to=`*destfile*`;new` and then running the `cu` input diversion to send the file to `FCOPY` which saves it in *destfile*. This facility might be useful with other systems also, such as an HP 1000 running RTE.
`~%setpt` *n*	Specify the number of seconds to wait for a prompt before giving up. The default is 2 seconds. Specifying a timeout of 0 seconds disables handshaking; that is, handshake appears to complete immediately.
`~%setps` *xy*	Set the handshake prompt to the characters *xy*. The default is DC1. The prompt can be any one or two characters. To specify a control character for *x* or *y*, use the Ctrl-X form where a circumflex (ASCII 94) precedes the character, as in `^X`. A null character can be specified with `^@`. (A null first character in the prompt implies a "null"

prompt, which always appears to be satisfied.) A circumflex is specified by `^^`.

`~%>[>]`*file* Divert output from the remote system to the specified file until another `~%>` command is given. When an output diversion is active, typing `~%>` terminates it, whereas `~%>` *anotherfile* terminates it and begins a new one. The output diversion remains active through a `~&` subshell, but unpredictable results can occur if input/output diversions are intermixed with `~%take` or `~%put`. The `~%>>` command appends to the named file. Note that these commands, which are interpreted by the *transmit* process, are unrelated to the `~>` commands described below, which are interpreted by the receive process.

~susp Suspend the `cu` session. *susp* is the suspend character set in the terminal when `cu` was invoked (usually `^Z` — see *stty*(1)). As in all other lines starting with tilde, a *~susp* line must be terminated by pressing **Return**.

Receive Process

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with `~>` initiates an output diversion to a file. The complete sequence is:

```
~>[>]: file
zero or more lines to be written to file
~>
```

Data from the remote is diverted (or appended, if `>>` is used) to *file*. The trailing `~>` terminates the diversion.

The use of `~%put` requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of `~%take` requires that the remote system support the `echo` and `cat` commands (see *echo*(1) and *cat*(1). Also, `stty tabs` mode should be set on the remote system if tabs are being copied without expansion. When connecting to a machine that uses the eighth bit as a parity bit, `stty istrip` mode should be set on the local system.

When `cu` is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed if `~~` is used. For example, using the keyboard on system X, `uname` can be executed on Z, X, and Y as follows where lines 1, 3, and 5 are keyboard commands, and lines 2, 4, and 6 are system responses:

```
uname
Z
~!uname
X
~~!uname
Y
```

In general, `~` causes the command to be executed on the original machine; `~~` causes the command to be executed on the next machine in the chain.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, `cu` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

Exit code is zero for normal exit; non-zero (various values) otherwise.

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud:

```
cu -s1200 9=2015551212
```

If the speed is not specified, 300 is the default value.

To log in on a system connected by a direct line:

```
cu -l/dev/ttyXX dir
```

To dial a system with the specific line and a specific speed:

```
cu -s1200 -l/dev/ttyXX dir
```

To dial a system using a specific line:

```
cu -l/dev/culXX 2015551212
```

To use a system name (yyyzzz):

```
cu yyyzzz
```

To connect directly to a modem:

```
cu -l/dev/culXX -m dir cu -l/dev/culXX -m dir
```

WARNINGS

cu buffers input internally.

DEPENDENCIES

HP Clustered Environment

cu is not supported on client nodes of an HP Cluster.

AUTHOR

cu was developed by AT&T and HP.

FILES

```
/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/lib/uucp/Dialers
/usr/spool/uucp/LCK..(tty-device)
/dev/null
```

SEE ALSO

cat(1), ct(1), echo(1), stty(1), uname(1), uucp(1), uuname(1).

UUCP tutorial in *Remote Access Users Guide* .

STANDARDS CONFORMANCE

cu: SVID2, XPG2, XPG3

NAME

cue - HP Character-Terminal User Environment (CUE)

SYNOPSIS

`/bin/cue`

DESCRIPTION

CUE provides an easy-to-use, attractive, customizable environment that allows users on Series 800 HP-UX systems to easily identify themselves to the system and begin a work session. CUE supports the following terminals:

HP 700/92	HP 700/94	HP 2392	HP 2394

A menubar is available for changing the native language of the session, changing the type of session to start upon a successful login, or getting on-line help. To obtain context-sensitive help at any time, press the function key labeled **HELP** (*f* 1).

A pulldown menu and function keys (*f* 1-*f* 8) are displayed, allowing the user to modify various options or to get help. Before the login is initiated, the user has the option of interactively changing the native language of the session and the type of session to start upon a successful login.

The default native language is C, but the language is easily modifiable by entering the Language Menu which is accessible by selecting the Configuration item in the menu bar. The native language can also be specified as a parameter to `cuegetty` (see *cuegetty*(1M)).

The default session type is the Bourne shell, `sh`, but the session type can be easily changed to `tsm`, `keysh`, `ksh`, or `csh` by entering the Session Type Menu which is accessible by selecting the Configuration item in the menu bar.

The following standard `login` features are available:

- password aging
- logging invalid login attempts in */etc/btmp*
- list of valid *ttys* for super-user login

CUE displays a visual screen that prompts for the username and password. CUE asks for the **username** and the corresponding password. If your username doesn't have a password, just press the <carriage return> key to skip this field. Terminal echo is turned off (where possible) during typing of the password so that it will not appear on any written record of the session. After three unsuccessful login attempts, a *hangup* signal is issued.

If password aging has been invoked by the super-user on your behalf, your password may have expired. In this case, you will be diverted into *passwd*(1) to change it, after which you can attempt to login again.

If login is not successfully completed within a certain period of time (e.g., five minutes), the terminal may be silently disconnected.

After a successful login, the accounting files are updated, user and group id's, group access list, and working directory are initialized. If the session type chosen is *tsm*, the SHELL to start in each *tsm* session is determined from corresponding user entries in the **/etc/passwd** file. CUE then forks the appropriate shell by using the last component of the shell pathname preceded by a - (for example, **-sh** or **-ksh**). When the session type is invoked with its name preceded by a minus in this manner, the shell performs its own initialization, including execution of profile, login, or other initialization scripts.

For example, if the user login shell is *sh*(1) or *ksh*(1) the shell executes the profile files **/etc/profile** and **$HOME/.profile** if they exist (and possibly others as well, depending on what they contain). Depending on what these profile files contain, messages regarding mail in your mail file or any messages you may have received since your last login may be displayed. At this point, *cuesession* is started to perform some accounting, display messages, and start your session.

If **/etc/btmp** is present, all unsuccessful login attempts are logged to this file. This feature is disabled if the file is not present. A summary of bad login attempts can be viewed by users with appropriate privileges by using **lastb**, see *last*(1M).

If **/etc/securetty** is present, login security is in effect, meaning that only users with appropriate privileges are allowed to log in successfully on the ttys listed in this file. Restricted ttys are listed by device name, one per line. Valid tty names are dependent on installation. Some examples could be **console**, **tty01**, **ttya1**, etc. Note that this feature does not inhibit a normal user from using **su**.

INTERNATIONALIZATION

All screens, labels, and messages are localizable. The message catalog *cue.cat* contains the localized representations of the default labels and messages. *Cue* will read the appropriate message catalog indicated by the **LANG** environment variable and display the localized strings. By selecting a native language in the Language Menu, the language of the CUE screens and the future work session can be specified. If the the message catalog exists for CUE in the language selected, CUE will be redisplayed in that language. If not, the CUE screens will continue in the current language and the work session that is started after a successful login will be started in the language selected. In either case, the **LANG** environment variable will be set appropriately for the resulting work session.

If CUE will be started on the command line or as the last item in the *be brought up using the language specified by the LANG environment variable. If CUE screens do not exist for the language specified by LANG, then the the default native language, C, will be used.*

If CUE will be started by *cuegetty*, it is possible to start up the CUE Login Screens in a language other than the default, C, by invoking *cuegetty* with the **-L** *nls_language* option. Of course, CUE screens and the *cue.cat* file must exist for the *nls_language* specified.

STARTING CUE

There are several methods that can be used to start *cue.*

- *cuegetty(1M)* entry can be placed in the */etc/inittab* file. This is the preferred method as the user does not need to do anything further to start *cue.*
- *cue(1)* can be started on the command line by merely typing: cue. Or, it can be started as the last line in the user's *login* configuration file.

Multiple *cue* logins may run simultaneously on separate terminals attached to the same local host. *Cuegetty(1M)* can be configured in the */etc/inittab* file for all users.

Remote users to the CUE system must access CUE by entering the *cue(1)* command at the command line prompt or as the last item in the user's *login* configuration file.

FILES

/etc/btmp	history of bad login attempts
/etc/logingroup	group file - defines group access lists
/etc/motd	message-of-the-day
/etc/passwd	password file - defines users, passwords, and primary groups
/etc/profile	system profile (initialization for all users)
/etc/securetty	list of valid ttys for root login
/etc/utmp	users currently logged in
/etc/wtmp	history of logins, logouts, and date changes
/usr/mail/*your-name*	mailbox for user *your-name*
/bin/cue	*cue* executable
/bin/cuegetty	*cuegetty* executable
/bin/cue.etc/cue.inittab	template for */etc/inittab*
/bin/cue.etc/cue.dm	screen descriptions
/bin/cue.etc/cuesession	starts selected session type
/usr/lib/nls/$LANG/cue.cat	NLS message catalog
/usr/man/man1/cue.1	man page for *cue(1)*
/usr/man/man1/cuegetty.1M	man page for *cuegetty(1M)*

ENVIRONMENT

Cue will invoke the user's session with the following default environment:

CUESESSION	is set to the session type selected valid values are : /bin/sh - Bourne Shell (DEFAULT) /usr/bin/tsm - manages up to 10 sessions at once /usr/bin/keysh - Easy Context-Sensitive Softkey Shell /bin/ksh - Korn Shell /bin/csh - C Shell
HOME	is set to the home directory of the user
LANG	is set to the native language selected (C is the default)
LOGNAME	is set to the user name
MAIL	is set to /usr/mail/$LOGNAME
NLSPATH	is set to the path applications search for NLS message catalogs usually /usr/lib/nls/%L/%N.cat
PATH	is set to the path to be searched for commands - :/bin:/usr/bin
SHELL	is set to the user's default shell (from /etc/passwd)

Several methods are available to modify or add to this list depending on the desired scope of the resulting environment variable.

Basic environment variables can be set for all CUE users on a system by setting the values in */etc/profile* and */etc/csh.login*. Personal environment variables can be set on a per-user basis in the script file $HOME/*.profile* for *sh* and *ksh* users or *.cshrc* for *csh* users.

Warning. Alias and function definitions need to be included in the file specified by ENV for *ksh* as this file will be sourced for each invocation of the shell. For *csh* users, the **.cshrc** file should be structured such that it cannot generate any output on standard output or standard error, including occasions when it is invoked without an affiliated terminal. *rcp*(1) causes **.cshrc** to be sourced, and any output generated by this file, even to standard error causes problems. Commands such as *stty*(1) should be placed in **.login**, not in **.cshrc**, so that their output cannot affect *rcp*(1).

For users with appropriate privileges, **PATH** is augmented to include `/etc`.

SEE ALSO

csh(1), cuegetty(1), env(1), keysh(1), ksh(1), login(1), nlsinfo(1), passwd(1), sh(1), tsm(1), btmp(4), environ(5), hpnls(5), lang(5).

NAME

cut - cut out (extract) selected fields of each line of a file

SYNOPSIS

`cut` `-c` *list* [*file* ...]
`cut` `-b` *list* [`-n`] [*file* ...]
`cut` `-f` *list* [`-d` *char*] [`-s`] [*file* ...]

DESCRIPTION

`cut` cuts out (extracts) columns from a table or fields from each line in a file; in data base parlance, it implements the projection of a relation. Fields as specified by *list* can be fixed length (defined in terms of character or byte position in a line when using the `-c` or `-b` option), or the length can vary from line to line and be marked with a field delimiter character such as the tab character (when using the `-f` option). `cut` can be used as a filter; if no files are given, the standard input is used.

When processing single-byte character sets, the `-c` and `-b` options are equivalent and produce identical results. When processing multi-byte character sets, when the `-b` and `-n` options are used together, their combined behavior is very similar, but not identical to the `-c` option.

Options

Options are interpreted as follows:

list A comma-separated list of integer *byte* (-b option), *character* (-c option), or *field* (-f option) numbers, in increasing order, with optional `-` to indicate ranges. For example:

- **`1,4,7`** Positions 1, 4, and 7.
- **`1-3,8`** Positions 1 through 3 and 8.
- **`-5,10`** Positions 1 through 5 and 10.
- **`3-`** Position 3 through last position.

`-b` *list* Cut based on a list of bytes. Each selected byte is output unless the `-n` option is also specified.

`-c` *list* Cut based on character positions specified by *list* (`-c 1-72` extracts the first 72 characters of each line).

`-f` *list* Where *list* is a list of fields assumed to be separated in the file by a delimiter character (see `-d`); for example, `-f 1,7` copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless `-s` is specified.

`-d` *char* The character following `-d` is the field delimiter (-f option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted. Adjacent field delimiters delimit null fields.

`-n` Do not split characters. If the high end of a range within a list is not the last byte of a character, that character is not included in the output. However, if the low end of a range within a list is not the first byte of a character, the entire character *is* included in the output."

`-s` Suppresses lines with no delimiter characters when using `-f` option. Unless `-s` is specified, lines with no delimiters appear in the output without alteration.

Hints

Use `grep` to extract text from a file based on text pattern recognition (using regular expressions). Use `paste` to merge files line-by-line in columnar format. To rearrange columns in a table in a different sequence, use `cut` and `paste`. See *grep*(1) and *paste*(1) for more information.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `cut` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

The delimiter specified with the `-d` argument must be a single-byte character. Otherwise, single- and multi-byte character code sets are supported.

EXAMPLES

Password file mapping of user ID to user names:

```
cut -d : -f 1,5 /etc/passwd
```

Set environment variable **name** to current login name:

```
name='who am i | cut -f 1 -d " "'
```

Convert file `source` containing lines of arbitrary length into two files where `file1` contains the first 500 bytes (unless the 500th byte is within a multi-byte character), and `file2` contains the remainder of each line:

```
cut -b 1-500 -n source > file1
cut -b 500- -n source > file2
```

DIAGNOSTICS

`line too long`
: Line length must not exceed `LINE_MAX` characters or fields, including the new-line character (see *limits*(5).

`bad list for b/c/f option`
: Missing `-b`, `-c`, or `-f` option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

`no fields` *list* is empty.

WARNINGS

`cut` does not expand tabs. Pipe text through *expand*(1) if tab expansion is required.

Backspace characters are treated the same as any other character. To eliminate backspace characters before processing by `cut`, use the `fold` or `col` command (see *fold*(1) and *col*(1)).

SEE ALSO

grep(1), paste(1).

STANDARDS CONFORMANCE

`cut`: SVID2, XPG2, XPG3, POSIX.2

NAME

cxref - generate C program cross-reference

SYNOPSIS

`cxref` [*options*] *files*

DESCRIPTION

`cxref` analyzes a collection of C files and attempts to build a cross-reference table. `cxref` utilizes a special version of `cpp` to include `#defined` information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the `-c` option, in combination. Each symbol contains an asterisk (*) before the declaring reference. Output is sorted in ascending collation order (see Environment Variables below).

Options

In addition to the `-D`, `-I`, and `-U` options (which are identical to their interpretation by `cc` (see *cc*(1)), `cxref`: recognizes the following *options*:

`-c`	Print a combined cross-reference of all input files.
`-w` *num*	Width option; format output no wider than *num* (decimal) columns. This option defaults to 80 if *num* is not specified or is less than 51.
`-o` *file*	Direct output to the named *file*.
`-s`	Operate silently; do not print input file names.
`-t`	Format listing for 80-column width.
`-Aa`	Choose ANSI mode. If not specified, compatibility mode (**`-Ac`** option) is selected by default.
`-Ac`	Choose compatibility mode. This option is selected by default if neither **`-Aa`** nor **`-Ac`** is specified.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the order in which the output is sorted.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `cxref` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files anyway.

EXAMPLES

Create a combined cross-reference of the files `orange.c`, `blue.c`, and `color.h`:

```
cxref -c orange.c blue.c color.h
```

Create a combined cross-reference of the files `orange.c`, `blue.c`, and `color.h`: and direct the output to the file `rainbow.x`:

```
cxref -c -o rainbow.x orange.c blue.c
```

WARNINGS

`cxref` considers a formal argument in a `#define` macro definition to be a declaration of that symbol. For example, a program that `#includes ctype.h` will contain many declarations of the variable `c`.

`cxref` uses a special version of the C compiler front end. This means that a file that will not compile probably cannot be successfully processed by `cxref`. In addition, `cxref` generates references *only* for those source lines that are actually compiled. This means that lines that are excluded by `#ifdefs` and the like (see *cpp*(1)) are not cross-referenced.

`cxref` does not parse the `CCOPTS` environment variable.

FILES

`/lib/cpp`	C-preprocessor.
`/lib/xpass`	Compatibility-mode special version of C compiler front end.
`/lib/xpass.ansi`	ANSI-mode special version of C compiler front end.

SEE ALSO

cc(1), cpp(1).

STANDARDS CONFORMANCE

`cxref`: SVID2, XPG2, XPG3

NAME

date - print or set the date and time

SYNOPSIS

`date` [`-u`]
`date` [`-u`] +*format*
`date` [`-u`] *mmddhhmm*[*yy*]

DESCRIPTION

`date` displays or sets the current HP-UX system clock date and time. The command has three forms:

`date [-u]` Display the current date and time.

`date [-u]` +*format*
: Display current date and time according to formatting directives specified in *format* (see Formatting Directives below).

The *format* string consists of zero or more directives and ordinary characters. A directive consists of a `%` character, an optional field width and precision specification, and a terminating character that determines the directive's behavior. All ordinary characters are copied unchanged into the output string, and the output string is always terminated with a new-line character.

The default *format* string is `%c` when *format* is not specified or the `date [-u]` command form is used.

`date [-u]` *mmddhhmm*[*yy*]
: Set HP-UX system clock to the date and time specified. Requires super-user privilege. If `-u` option is given, the specified date and time is assumed to be Universal Coordinated Time. The numeric argument is interpreted left-to-right as follows:

mm	Month number.
dd	Day number in the month.
hh	Hour number (24 hour system).
mm	Minute number.
yy	(optional) Last 2 digits of the year number. If not specified, current year is used.

Options

`-u` If this option is given, all operations occur as if the `TZ` environment variable were set to Coordinated Universal Time (UTC).

The HP-UX system operates in Coordinated Universal Time (UTC). `date` automatically converts to and from local standard and daylight time unless the `-u` option is given (see Environment Variables below).

If the super-user attempts to set the date backwards, `date` generates a warning, and requires an extra confirmation before proceeding.

When `date` is used to set the date, a pair of date change records is written to the file `/etc/wtmp`.

In an HP Clustered Environment, the date and time are automatically set when the system comes up as a cluster node. Setting the date and time from any node in a cluster sets the date and time on all nodes in the cluster.

Formatting Directives

The following directives, shown without the optional field width and precision specification, are replaced by the indicated characters:

`%a`	Abbreviated weekday name.
`%A`	Full weekday name.
`%b`	Abbreviated month name.

%B	Full month name.
%c	Current date and time representation.
%C	Century (a year divided by 100 and truncated to an integer) as a decimal number (00-99).
%d	Day of the month as a decimal number [01,31].
%D	Date in the format *mm* /*dd* /*yy*.
%e	Day of the month as a decimal number (1-31 in a two digit field with leading space-character fill).
%E	Combined Emperor/Era name and year.
%h	Same as **%b**.
%H	Hour (24-hour clock) as a decimal number [00,23].
%I	Hour (12-hour clock) as a decimal number [01,12].
%j	Day of the year as a decimal number [001,366].
%m	Month as a decimal number [01,12].
%M	Minute as a decimal number [00,59].
%n	New-line character.
%N	Emperor/Era name.
%o	Emperor/Era year.
%p	Equivalent of either AM or PM.
%r	12-hour clock time (01-12) using AM/PM notation.
%S	Second as a decimal number [00,59].
%t	Tab character.
%u	Weekday as a decimal number [1(Monday),7].
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.
%V	Week of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, it is considered week 1; otherwise, it is week 53 of the previous year, and the next week is week 1.
%w	Weekday as a decimal number [0(Sunday),6].
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.
%x	Current date representation.
%X	Current time representation.
%y	Year without century as a decimal number [00,99].
%Y	Year with century as a decimal number.
%Z	Time zone name (or no characters if time zone cannot be determined).
%%	%

The following directives are provided for backward compatibility. It is recommended that the preceding directives be used in preference to those which follow.

%D	Date in usual U.S. Format (%m/%d/%y) (use **%x** instead).
%F	Full month name (use **%B** instead).
%h	Abbreviated month name (use **%b** instead).

`%r`	Time in 12-hour US format (%I:%M:%S [AM | PM]) (use `%X` instead).
`%T`	Time in 24-hour US format (%H:%M:%S) (use `%X` instead).
`%z`	Time zone name (or no characters if time zone cannot be determined) (use `%Z` instead).

If a directive is not one of the above, the behavior is undefined.

Modified Formatting Directives

Some Formatting Directives can be modified by the `E` and `O` modifier characters to indicate a different format or specification as specified in the `LC_TIME` environment variable. If the corresponding keyword (see `era`, `era_year`, `era_d_fmt`, and `alt_digits`) is not specified or not supported, the unmodified field descriptor value is used.

`%EC`	The name of the base year in alternate representation.
`%Ex`	Alternate date representation.
`%Ey`	Offset from `%EC` (year only) in the alternate representation.
`%Od`	Day of month using the alternate numeric symbols.
`%Oe`	Day of month using the alternate numeric symobols with leading space-character fill if applicable.
`%OH`	Hour (24 hour clock) using the alternate numeric symbols.
`%OI`	Hour (12 hour clock) using the alternate numeric symbols.
`%Om`	Month using the alternate numeric symbols.
`%OM`	Minutes using the alternate numeric symbols.
`%OS`	Seconds using the alternate numeric symbols.
`%OU`	Week number of the year (Sunday as the first day of the week) using the alternate numeric symbols.
`%Ow`	Weekday as number using the alternate representation (Sunday = `0`).
`%OW`	Weekday number of the year (Monday as the first day of the week) using the alternate numeric symbols.
`%Oy`	Year (offset from `%C`) in alternate representation.

Field Width and Precision

An optional field width and precision specification can immediately follow the initial `%` of a directive in the following order:

[`-` | `0`]*w*	the decimal digit string *w* specifies a minimum field width in which the result of the conversion is right- or left-justified. Default is right-justified (with space padding). If the optional `-` option is specified, the result is left-justified with space padding on the right. If the optional `0` option is specified, the result is right-justified and padded with zeros on the left.
`.`*p*	the decimal digit string *p* specifies the minimum number of digits to appear for the `d`, `H`, `I`, `j`, `m`, `M`, `o`, `S`, `U`, `w`, `W`, `y`, and `Y` directives, and the maximum number of characters to be used from the `a`, `A`, `b`, `B`, `c`, `D`, `E`, `F`, `h`, `n`, `N`, `p`, `r`, `t`, `T`, `x`, `X`, `z`, `Z`, and `%` directives. In the first case, if a directive supplies fewer digits than specified by the precision, it is expanded with leading zeros. In the second case, if a directive supplies more characters than specified by the precision, excess characters are truncated on the right.

If no field width or precision is specified for a `d`, `H`, `I`, `m`, `M`, `S`, `U`, `W`, `y`, or `j` directive, a default of `.2` is used for all except `j` for which `.3` is used.

EXTERNAL INFLUENCES

Environment Variables

`TZ` determines the conversion between the system time in UTC and the time in the user's local time zone (see *environ*(5) and *tztab*(4)). `TZ` also determines the content (that is, the time-zone name produced by the `%z` and `%Z` directives) of date and time strings output by the `date` command.

If `TZ` is not set in the environment or is set to the empty string, a default of `EST5EDT` is used.

`LC_TIME` determines the content (for example, weekday names produced by the `%a` directive) and format (for example, current time representation produced by the `%X` directive) of date and time strings output by the `date` command.

`LC_CTYPE` determines the interpretation of the bytes within the *format* string as single and/or multi-byte characters.

`LC_NUMERIC` determines the characters used to form numbers for those directives that produce numbers in the output. The characters used are those defined by `ALT_DIGITS` (see *langinfo*(5)).

`LANG` determines the language in which messages (other than the date and time strings) are displayed.

If `LC_TIME` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `date` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`no permission`	Super-user privileges are required to change the date.
`bad conversion`	Date specification is syntactically incorrect.
`bad format character`	Field directive is not recognizable.

EXAMPLES

Set Date

Set date to Oct 8, 12:45 AM:

```
date 10080045
```

Display Formatted Date

Display current date and time using a specified format:

```
date ´+DATE: %m/%d/%y%nTIME: %H:%M:%S´
```

Output resembles the following:

```
DATE: 10/08/87
TIME: 12:45:05
```

Display Formatted Date Using Local Language

Using the date as set in the first example above and `LC_TIME` set to `german`:

```
date ´%-4.4h %2.1d %H:%M´
```

generates output similar to:

```
Okt   8 12:45
```

where the month field is four bytes long, flush-left, and space-padded on the right if the month name is shorter than four bytes. The day field is two bytes long, with leading zeros suppressed.

WARNINGS

Changing the date while the system is running in multi-user mode should be avoided to prevent disrupting user-scheduled and time-sensitive programs and processes. Also, changing the date can cause `make` and the SCCS and `cron` subsystems to behave in an unexpected manner. The `cron` daemon should be killed prior to setting the date backwards, then restarted (see *cron*(1M)). SCCS files should be checked with `val` if deltas have been made while the clock was set wrong (see *val*(1)).

Former HP-UX format option **A** has been changed to **W** for ANSI compatability.

AUTHOR

`date` was developed by AT&T and HP.

FILES

`/etc/wtmp`

SEE ALSO

stime(2), ctime(3C), getdate(3C), strftime(3C), tztab(4), environ(5), lang(5), langinfo(5).

STANDARDS CONFORMANCE

`date`: SVID2, XPG2, XPG3, POSIX.2

NAME

dc - desk calculator

SYNOPSIS

`dc` [*file*]

DESCRIPTION

`dc` is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See *bc*(1), a preprocessor for `dc` that provides infix notation and a C-like syntax that implements functions. `bc` also provides reasonable control structures for programs.) The overall structure of `dc` is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. An end of file on standard input or the `q` command stop dc. The following constructions are recognized:

number	The value of the number is pushed on the stack. A number is an unbroken string of the digits **0-9** or **A-F**. It can be preceded by an underscore (_) to input a negative number. Numbers can contain decimal points.
`+ - / * % ^`	The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored and a warning generated. The remainder is calculated according to the current scale factor; it is not the integer modulus function. `7 % 3` yields .1 (one tenth) if scale is 1 because `7 / 3` is 2.3 with .1 as the remainder.
`s`*x*	The top of the stack is popped and stored into a register named *x*, where *x* can be any character. If the `s` is capitalized, *x* is treated as a stack and the value is pushed on it.
`l`*x*	The value in register *x* is pushed on the stack. Register *x* is not altered. All registers start with zero value. If the `l` is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.
`d`	The top value on the stack is duplicated.
`p`	The top value on the stack is printed. The top value remains unchanged. `P` interprets the top of the stack as an ASCII string, removes it, and prints it.
`f`	All values on the stack are printed.
`q`	exits the program. If executing a string, the recursion level is popped by two. If `q` is capitalized, the top value on the stack is popped and the string execution level is popped by that value.
`x`	treats the top element of the stack as a character string and executes it as a string of `dc` commands.
`X`	replaces the number on the top of the stack with its scale factor.
`[ ... ]`	puts the bracketed ASCII string onto the top of the stack. Strings can be nested by using nested pairs of brackets.
`<`*x* `>`*x* `=`*x* `!<`*x* `!>`*x* `!=`*x*	The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.
`v`	Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.
`!`	Interprets the rest of the line as an HP-UX system command (unless the next character is <, >, or =, in which case appropriate relational operator above is used).
`c`	All values on the stack are popped.
`i`	The top value on the stack is popped and used as the number radix for further input.

`I`	pushes the input base on the top of the stack.
`o`	The top value on the stack is popped and used as the number radix for further output. See below for notes on output base.
`O`	pushes the output base on the top of the stack.
`k`	the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
`K`	pushes the scale factor on the top of the stack.
`z`	The stack level is pushed onto the stack.
`Z`	replaces the number on the top of the stack with its length.
`?`	A line of input is taken from the input source (usually the terminal) and executed.
`;` and `:`	Used by `bc` for array operations.
`Y`	Generates debugging output for `dc` itself.

The input base may be any number, but only the digits 0-9 and A-F are available for input, thus limiting the usefulness of bases outside the range 1-16. All 16 possible digits may be used in any base; they always take their conventional values.

The output base may be any number. Bases in the range of 2-16 generate the "usual" results, with the letters A-F representing the values from 10 through 16. Bases 0 and 1 generate a string of `1`s whose length is the value of the number. Base −1 generates a similar string consisting of `d`s. Other bases have each "digit" represented as a (multi-digit) decimal number giving the ordinal of that digit. Each "digit" is signed for negative bases. "Digits" are separated by spaces. Given the definition of output base, the command `Op` always yields "10" (in a representation appropriate to the base); `O1-p` yields useful information about the output base.

DIAGNOSTICS

x `is unimplemented`	Where *x* is an octal number.
`stack empty`	There are insufficient elements on the stack to do what was asked.
`Out of space`	The free list is exhausted (too many digits).
`Out of headers`	Too many numbers are being kept around.
`Out of pushdown`	Too many items are on the stack.
`Nesting Depth`	There are too many levels of nested execution.

EXAMPLES

This example prints the first ten values of n! (n factorial):

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1).

DC: An Interactive Desk Calculator tutorial in *Number Processing Users Guide*.

NAME

dd - convert, reblock, translate, and copy a (tape) file

SYNOPSIS

`dd` [*option=value*] ...

DESCRIPTION

`dd` copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. Input and output block size can be specified to take advantage of raw physical I/O.

Options

`dd` recognizes the following *option=value* pairs:

`if=`*file*	Input file name; default is standard input.
`of=`*file*	Output file name; default is standard output. The output file will be created using the same owner and group used by creat().
`ibs=`*n*	Input block size is *n* bytes (default 512).
`obs=`*n*	Output block size is *n* bytes (default 512).
`bs=`*n*	Set both input and output block size to the same size, superseding `ibs` and `obs`. This option is particularly efficient if no conversion is specified, because no in-core copy is necessary.
`cbs=`*n*	Conversion buffer size is *n bytes*.
`skip=`*n*	Skip *n* input blocks before starting copy.
`seek=`*n*	Seek *n* blocks from beginning of output file before copying. This option is ignored on a raw magnetic tape device. See *mt*(1) for information about operations on raw magnetic tape devices.
`count=`*n*	Copy only *n* input blocks.
`conv=`*option*	Data conversion option. Use one of the following:

`conv=ascii`	Convert EBCDIC to ASCII.
`conv=ebcdic`	Convert ASCII to EBCDIC
`conv=ibm`	Convert ASCII to EBCDIC using an alternate conversion table
`conv=lcase`	Map US ASCII alphabetics to lowercase
`conv=ucase`	Map US ASCII alphabetics to uppercase
`conv=swab`	Swap every pair of bytes
`conv=noerror`	Do not stop processing on an error
`conv=sync`	Pad every input block to input block size (ibs)
`conv=notrunc`	Do not truncate existing file on output
`conv=block`	Convert input record to a fixed length specified by `cbs`
`conv=unblock`	Convert fixed length records to variable length
`conv=...,` ...	Multiple comma-separated conversions

Where sizes are required, *n* indicates a numerical value in bytes. Numbers can be specified using the forms:

n	for *n* bytes
n`k`	for *n* Kbytes ($n \times 1024$),
n`b`	for *n* blocks ($n \times 512$), or
n`w`	for *n* words ($n \times 2$).

To indicate a product, use **x** to separate number pairs.

The `cbs` option is used when `block`, `unblock`, `ascii` or `ebcdic` conversion is specified. In case of `ascii` , *cbs* characters are placed into the conversion buffer, converted to ASCII, trailing blanks are trimmed, and a new-line is added before sending the line to the output. In case of `ebcdic` , ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks are added to make up an output block of size *cbs*.

Upon completion, `dd` reports the number of whole and partial input and output records.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

Environment Variables

The following environment variables will affect execution of `dd`:

LANG : This variable determines the locale when LC_ALL and a corresponding variable (beginning with LC_) do not specify a locale.

LC_ALL : This variable determines the locale used to override any values set by LANG or any environment variables beginning with LC_ .

LC_CTYPE : This variable determines the locale for the interpretation of sequences of bytes of text data as characters (single/multiple byte characters, upper/lower case characters).

LC_MESSAGES : This variable determines the language in which messages should be written.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occurred.

DIAGNOSTICS

f+p `records in`	Number of full and partial blocks read.
f+p `records out`	Number of full and partial blocks written.

EXAMPLES

Read an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into an ASCII file named **x**:

```
dd  if=/dev/rmt/0m  of=x  ibs=800  cbs=80  conv=ascii,lcase
```

Note the use of the raw magnetic tape device file. `dd` is especially suited to I/O on raw physical devices because it allows reading and writing in arbitrary block sizes.

WARNINGS

You may experience trouble writing directly to or reading directly from a cartridge tape. For best results, use *tcio*(1) as an input or output filter. For example, use

```
... |dd ... |tcio -ovVS 256 /dev/rct/c0
```

for output to a cartridge tape, or

```
tcio -ivS 256 /dev/rct/c0 |dd ... | ...
```

for input from a cartridge tape.

Some devices, such as 1/2-inch magnetic tapes, are incapable of seeking. Such devices must be positioned prior to running `dd` by using *mt*(1) or some other appropriate command.

ASCII and EBCDIC conversion tables are taken from the 256-character ACM standard, Nov, 1968. The `ibm` conversion, while less widely accepted as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-line characters are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

If `if` or `of` refers to a disk section of greater than 2 Gbytes and a block size greater than 2 Kbytes is specified, `dd` prints an error message and exits.

SEE ALSO

cp(1), mt(1), tr(1), mt(7).

STANDARDS CONFORMANCE

dd: SVID2, XPG2, XPG3, POSIX.2

NAME

delta - make a delta (change) to an SCCS file

SYNOPSIS

`delta` [`-r` *SID*] [`-s`] [`-n`] [`-g` *list*] [`-m` *mrlist*] [`-y` *comment*] [`-p`] *files*

DESCRIPTION

`delta` is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by `get` (called the *g-file*, or generated file). See *get*(1).

`delta` makes a delta to each named SCCS file. If a directory is named, `delta` behaves as though each file in the directory was specified as a named file, except that non-SCCS files (last component of the path name does not begin with `.s`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read (see WARNINGS). Each line of the standard input is taken to be the name of an SCCS file to be processed.

`delta` may issue prompts on the standard output, depending upon certain options specified and flags (see *admin*(1)) that may be present in the SCCS file (see `-m` and `-y` options below).

Options

Option arguments apply independently to each named file.

`-r`*SID* Uniquely identifies which delta is to be made to the SCCS file. Use of this option is necessary only if two or more outstanding `get`s for editing (`get -e`) on the same SCCS file were done by the same person (login name). The *SID* value specified with the `-r` option can be either the *SID* specified on the `get` command line or the *SID* to be made as reported by the `get` command (see *get*(1)). A diagnostic results if the specified *SID* is ambiguous, or, if necessary and omitted on the command line.

`-s` Suppresses issuing, on the standard output, of the created delta's *SID* as well as the number of lines inserted, deleted and unchanged in the SCCS file.

`-n` Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

`-g`*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (*SID*) created by this delta.

`-m`[*mrlist*] If the SCCS file has the `v` flag set (see *admin*(1)), a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

If `-m` is not used and the standard input is a terminal, the prompt `MRs?` is issued on the standard output before the standard input is read. If the standard input is not a terminal, no prompt is issued. The `MRs?` prompt always precedes the `comments?` prompt (see `-y` option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the `v` flag has a value (see *admin*(1)), it is assumed to be the name of a program (or shell procedure) that is to validate the correctness of the MR numbers. If a non-zero exit status is returned from the MR number-validation program, `delta` assumes that the MR numbers were not all valid and terminates.

`-y`[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If `-y` is not specified and the standard input is a terminal, the prompt `comments?` is issued on the standard output before the standard input is read. If the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

`-p` Causes `delta` to print (on the standard output in a *diff*(1) format) the SCCS file differences before and after the delta is applied.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **delta** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

SCCS files can be any length, but the number of lines in the text file itself cannot exceed 99 999 lines.

Lines beginning with an ASCII SOH character (octal 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(4)) and will cause an error.

A **get** of many SCCS files, followed by a **delta** of those files, should be avoided when the **get** generates a large amount of data. Instead, multiple **get/delta** sequences should be used.

If the standard input (**-**) is specified on the **delta** command line, the **-m** (if necessary) and **-y** options *must* also be present. Omission of these options causes an error.

Comments are limited to text strings not exceeding 512 characters.

FILES

All files of the form *?*-file are explained in the *SCCS User's Guide*. The naming convention for these files is also described there. All files below except the g-file are created in the same directory as the s-file. The g-file is created in the user's working directory.

g-file	Existed before the execution of *delta*; removed after completion of **delta** (unless **-n** was specified).
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the file retrieved by *get* and the *g-file*.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4).

SCCS User's Guide in *Programming on HP-UX*.

STANDARDS CONFORMANCE

delta: SVID2, XPG2, XPG3

NAME

deroff - remove nroff, tbl, and neqn constructs

SYNOPSIS

`deroff` [-m*x*] [-w] [-i] [*file* ...]

DESCRIPTION

`deroff` reads each *file* in sequence and removes all `nroff` requests, macro calls, backslash constructs, `neqn` constructs (between `.EQ` and `.EN` lines, and between delimiters — see *neqn*(1)), and `tbl` descriptions (see *tbl*(1)), replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. `deroff` follows chains of included files (`.so` and `.nx` `nroff/troff` formatter commands); if a file has already been included, a `.so` naming that file is ignored and a `.nx` naming that file terminates execution. If no input file is given, `deroff` reads the standard input.

The `-m` option can be followed by an `m`, `s`, or `l`. The `-mm` option causes the macros be interpreted such that only running text is output (that is, no text from macro lines). The `-ml` option forces the `-mm` option and also causes deletion of lists associated with the `mm` macros.

If the `-w` option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that contains at least two letters and is composed of letters, digits, ampersands (`&`), and apostrophes (`'`); in a macro call. However, a "word" is a string that begins with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

If the `-i` option is specified, `deroff` ignores the `.so` and `.nx` `nroff/troff` commands.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text and filenames as single and/or multi-byte characters. Note that multi-byte punctuation characters are not recognized when using the -w option.

LANG determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `deroff` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported except in the case of the `-w` option which supports only single-byte code sets.

WARNINGS

`deroff` is not a complete `nroff` interpreter; thus it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The `-ml` option does not handle nested lists correctly.

AUTHOR

`deroff` was developed by the University of California, Berkeley.

SEE ALSO

neqn(1), nroff(1), tbl(1).

NAME

diff - differential file and directory comparator

SYNOPSIS

`diff` [`-C` *n*] [`-S` *name*] [`-lrs`] [`-bcefhintw`] *dir1 dir2*
`diff` [`-C` *n*] [`-S` *name*] [`-bcefhintw`] *file1 file2*
`diff` [`-D` *string*] [`-biw`] *file1 file2*

DESCRIPTION

Comparing Directories

If both arguments are directories, `diff` sorts the contents of the directories by name, then runs the regular file `diff` algorithm (described below) on text files that have the same name in each directory but are different. Binary files that differ, common subdirectories, and files that appear in only one directory are listed. When comparing directories, the following options are recognized:

- `-l` Long output format; each text file `diff` is piped through `pr` to paginate it (see *pr*(1)). Other differences are remembered and summarized after all text file differences are reported.
- `-r` Applies `diff` recursively to common subdirectories encountered.
- `-s` `diff` reports files that are identical but otherwise not mentioned.
- `-S` *name* Starts a directory `diff` in the middle of the sorted directory, beginning with file *name*.

Comparing Files

When run on regular files, and when comparing text files that differ during directory comparison, `diff` tells what lines must be changed in the files to bring them into agreement. `diff` usually finds a smallest sufficient set of file differences. However, it can be misled by lines containing very few characters or by other situations. If neither *file1* nor *file2* is a directory, either can be specified as `-`, in which case the standard input is used. If *file1* is a directory, a file in that directory whose filename is the same as the filename of *file2* is used (and vice versa).

There are several options for output format. The default output format contains lines resembling the following:

n1 `a` *n3*`,`*n4*
n1`,`*n2* `d` *n3*
n1`,`*n2* `c` *n3*`,`*n4*

These lines resemble `ed` commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging `a` for `d` and reading backwards one may ascertain equally how to convert *file2* into *file1*. As in `ed`, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by `<`, then all the lines that are affected in the second file flagged by `>`.

Except for `-b`, `-w`, `-i`, or `-t` which can be given with any of the others, the following options are mutually exclusive:

- `-e` Produce a script of `a`, `c`, and `d` commands for the `ed` editor suitable for recreating *file2* from *file1*. Extra commands are added to the output when comparing directories with `-e`, so that the result is a shell script for converting text files common to the two directories from their state in *dir1* to their state in *dir2* (see *sh-bourne*(1)
- `-f` Produce a script similar to that of the `-e` option that is not useful with `ed` but is more readable by humans.
- `-n` Produce a script similar to that of `-e`, but in the opposite order, and with a count of changed lines on each insert or delete command. This is the form used by `rcsdiff` (see *rcsdiff*(1)).
- `-c` Produce a difference list with 3 lines of context. `-c` modifies the output format slightly: the output begins with identification of the files involved, followed by their creation dates, then each change separated by a line containing about twelve asterisks (`*`)s. Lines removed from *file1* are marked with `-`, and lines added to *file2* are marked `+`. Lines that change from one file to the other are marked in both files with with `!`. Changes that lie within 3 lines of each other in the file are grouped together on output.

`-C` *n* Output format similar to `-c` but with *n* lines of context.

`-h` Do a fast, half-hearted job. This option works only when changed stretches are short and well separated, but can be used on files of unlimited length.

`-D` *string*
Create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1*, while compiling the result with *string* defined is equivalent to compiling *file2*.

`-b` Ignore trailing blanks (spaces and tabs) and treat other strings of blanks as equal.

`-w` Ignore all whitespace (blanks and tabs). For example, `if ( a == b )` and `if(a==b)` are treated as equal.

`-i` Ignores uppercase/lowercase differences. Thus **A** is treated the same as **a**.

`-t` Expand tabs in output lines. Normal or `-c` output adds one or more characters to the front of each line. Resulting misalignment of indentation in the original source lines can make the output listing difficult to interpret. This option preserves original source file indentation.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the space characters for the `diff` command.

LANG determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `diff` and `diffh` behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that `diff` and `diffh` do not recognize multi-byte alternative space characters.

RETURN VALUE

Upon completion, `diff` returns with one of the following exit values:

0	No differences were found.
1	Differences were found.
>1	An error occurred.

EXAMPLES

The following command creates a script file `script`:

```
diff -e x1 x2 >script
```

The script file can then be used to create the file `x2` from the file `x1` using the editor `ed` in the following manner: `w" is added to the end`

```
echo w >> script
ed x1 < script
```

The following command produces the difference output with 2 lines of context information before and after the line that was different:

```
diff -C2 x1 x2
```

The following command ignores all blanks and tabs and ignores uppercase-lowercase differences.

```
diff -wi x1 x2
```

WARNINGS

Editing scripts produced by the `-e` or `-f` option are naive about creating lines consisting of a single dot (`.`).

When comparing directories with the `-b`, `-w`, or `-i` options specified, `diff` first compares the files in the same manner as `cmp`, then runs the `diff` algorithm if they are not equal. This may cause a small amount of spurious output if the files are identical except for insignificant blank strings or uppercase/lowercase differences.

The default algorithm requires memory allocation of roughly six times the size of the file. If sufficient memory is not available for handling large files, the `-h` option or `bdiff` can be used (see *bdiff*(1)).

When run on directories with the `-r` option, `diff` recursively descends sub-trees. When comparing deep multi-level directories, more memory may be required than is currently available on the system. The amount of memory required depends on the depth of recursion and the size of the files.

AUTHOR

`diff` was developed by AT&T, the University of California, Berkeley, and HP.

FILES

`/tmp/d?????`
`/usr/lib/diffh` used by `-h` option

SEE ALSO

bdiff(1), cmp(1), comm(1), diff3(1), diffh(1), diffmk(1), dircmp(1), ed(1), more(1), nroff(1), rcsdiff(1), sccsdiff(1), sdiff(1), terminfo(4).

STANDARDS CONFORMANCE

`diff`: SVID2, XPG2, XPG3, POSIX.2

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

`diff3 [-ex3]` *file1 file2 file3*

DESCRIPTION

`diff3` compares three versions of a file, and prints disagreeing ranges of text flagged with these codes:

`====`	all three files differ
`====1`	*file1* is different
`====2`	*file2* is different
`====3`	*file3* is different

The type of change required to convert a given range of a given file to some other is indicated in one of these ways:

f`:`*n1*`a`	Text is to be appended after line number *n1* in file *f*, where *f* = `1`, `2`, or `3`.
f`:`*n1*`,`*n2*`c`	Text is to be changed in the range line *n1* through line *n2*. If *n1* = *n2*, the range can be abbreviated to *n1*.

The original contents of the range follows immediately after a `c` indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the `-e` option, `diff3` produces a script for the `ed` editor that can be used to incorporate into *file1* all changes between *file2* and *file3* (see *ed*(1)); i.e., the changes that normally would be flagged `====` and `====3`. Option `-x` (`-3`) produces a script to incorporate only changes flagged `====` (`====3`). The following command applies the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed - file1
```

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

FILES

```
/tmp/d3*
/usr/lib/diff3prog
```

SEE ALSO

diff(1).

WARNINGS

Text lines that consist of a single `.` defeat `-e`.
Files longer than 64K bytes do not work.

NAME

diffmk - mark differences between files

SYNOPSIS

`diffmk` *name1 name2 name3*

DESCRIPTION

`diffmk` compares two versions of a file and creates a third file that includes `nroff/troff` "change mark" commands. *name1* and *name2* are the old and new versions of the file. `diffmk` generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (`.mc`) requests. When *name3* is formatted, changed or inserted text is shown by a | character at the right margin of each line. The position of deleted text is shown by a single `*`.

`diffmk` can also be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file `macs` contains:

```
.pl 1
.ll 77
.nf
.eo
```

The `.ll` request can specify a different line length, depending on the nature of the program being printed. The `.eo` request is probably needed only for C programs.

If the characters | and * are inappropriate, a copy of `diffmk` can be edited to change them (`diffmk` is a shell script).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

Aesthetic considerations may dictate manual adjustment of some output.

`diffmk` does not differentiate between changes in text and changes in formatter request coding. Thus, file differences involving only formatting changes with no change in actual text can produce change marks, such as replacing `.sp` with `.sp 2` in a text source file.

Although unlikely, certain combinations of formatting requests can cause change marks to either disappear or to mark too much. Manual intervention may be required because the subtleties of various formatting macro packages and preprocessors is beyond the scope of *diffmk*. `tbl` cannot tolerate `.mc` commands in its input (see *tbl*(1)), so any `.mc` request that would appear inside a `.TS` range is silently deleted. The script can be changed if this action is inappropriate, or `diffmk` can be run on two files that have both been run through the `tbl` preprocessor before any comparisons are made.

`diffmk` uses `diff`, and thus has the same limitations on file size and performance that *diff* may impose (see *diff*(1)). In particular the performance is non-linear with the size of the file, and very large files (well over 1000 lines) may take extremely long to process. Breaking the file into smaller pieces may be advisable.

`diffmk` also uses the *ed*(1) editor. If the file is too large for `ed`, `ed` error messages may be imbedded in the file. Again, breaking the file into smaller pieces may be advisable.

SEE ALSO

diff(1), nroff(1).

NAME

dircmp - directory comparison

SYNOPSIS

`dircmp` [`-d`] [`-s`] [`-w`*n*] *dir1 dir2*

DESCRIPTION

`dircmp` examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Sorted listings of files that are unique to each directory are generated for all the options. If no option is entered, a sorted list is output indicating whether the filenames common to both directories have the same contents.

`-d`	Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).
`-s`	Suppress messages about identical files.
`-w`*n*	Change the width of the output line to *n* characters. The default width is 72.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `dircmp` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

EXAMPLES

Compare the two directories `slate` and `sleet` and produce a list of changes that would make the directories identical:

```
dircmp -d slate sleet
```

SEE ALSO

cmp(1), diff(1).

STANDARDS CONFORMANCE

`dircmp`: SVID2, XPG2, XPG3

NAME

domainname - set or display name of Network Information Service domain

SYNOPSIS

`domainname` [*name_of_domain*]

DESCRIPTION

Network Information Service (NIS) uses domain names to refer collectively to a group of hosts. Without an argument, *domainname* displays the name of the NIS domain. Only super-user can set the domain name by providing *name_of_domain*. The domain name is usually set in the startup script `/etc/netnfsrc`.

DEPENDENCIES

NIS servers use the NIS domain name as the name of a subdirectory of `/usr/etc/yp`. Since the NIS domain name can be as long as 64 characters, *name_of_domain* may exceed the maximum filename length allowed on a given file system. If that length is exceeded, the subdirectory name becomes a truncated version of the NIS domain name.

For a machine that has a short-filename file system to act correctly as a NIS server, be sure the first 14 characters of all NIS domains on the network are unique.

AUTHOR

`domainname` was developed by Sun Microsystems, Inc.

SEE ALSO

ypinit(1M), getdomainname(2), setdomainname(2).

INTERNATIONAL SUPPORT

8-bit data, messages.

NAME

dos2ux, ux2dos - convert ASCII file format

SYNOPSIS

`dos2ux` *file* ...
`ux2dos` *file* ...

DESCRIPTION

`dos2ux` and **`ux2dos`** read each specified *file* in sequence and write it to standard output, converting to HP-UX format or to DOS format, respectively. Each *file* can be either DOS format or HP-UX format for either command.

A DOS file name is recognized by the presence of an embedded colon (`:`) delimiter; see *dosif*(4) for DOS file naming conventions.

If no input file is given or if the argument `-` is encountered, **`dos2ux`** and **`ux2dos`** read from standard input. Standard input can be combined with other files.

EXAMPLES

Print file `myfile` on the display:

```
dos2ux myfile
```

Convert `file1` and `file2` to DOS format then concatenate them together, placing them in `file3`.

```
ux2dos file1 file2 > file3
```

RETURN VALUE

`dos2ux` and **`ux2dos`** return `0` if successful or `2` if the command failed. The only possible failure is the inability to open a specified file, in which case the commands print a warning.

WARNINGS

Command formats resembling:

```
dos2ux file1 file2 > file1
```

overwrite the data in `file1` before the concatenation begins, causing a loss of the contents of `file1`. Therefore, be careful when using shell special characters.

SEE ALSO

doschmod(1), doscp(1), dosdf(1), dosls(1), dosmkdir(1), dosrm(1), dosif(4).

NAME

doschmod - change attributes of a DOS file

SYNOPSIS

`doschmod` [`-u`] *mode device* **:** *file* ...

DESCRIPTION

`doschmod` is the DOS counterpart of `chmod` (see *chmod*(1)).

Options

`doschmod` recognizes one option:

`-u`	Disable argument case conversion. In the absence of this option, all DOS file names are converted to uppercase.

A DOS file name is recognized by the presence of an embedded colon (`:`) delimiter; see *dosif*(4) for DOS file naming conventions.

The attributes of each named file are changed according to *mode*, which is an octal number in the range 000 to 0377. *mode* is constructed from the logical OR of the following modes:

200	Reserved. Do not use.
100	Reserved. Do not use.
040	Archive. Set whenever the file has been written to and closed.
020	Directory. Do not modify.
010	Volume Label. Do not modify.
004	System file. Marks files that are part of the DOS operating system.
002	Hidden file. Marks files that do not appear in a DOS directory listing using the DOS DIR command.
001	Read-Only file. Marks files as read-only.

WARNINGS

Specifying inappropriate *mode* values can make files and/or directories inaccessible, and in certain cases can damage the file system. To prevent such problems, do not change the mode of directories and volume labels.

Normal users should have no need to use *mode* bits other than 001, 002, and 040.

EXAMPLES

Mark file `/dev/rfd9122:memo.txt` as a hidden file:

```
doschmod 002 /dev/rfd9122:memo.txt
```

Mark file `driveC:autoexec.bat` read-only:

```
doschmod 001 driveC:autoexec.bat
```

SEE ALSO

chmod(1), dos2ux(1), doscp(1), dosdf(1), dosls(1), dosmkdir(1), dosrm(1), chmod(2), dosif(4).

NAME

doscp - copy to or from DOS files

SYNOPSIS

doscp [**-fvu**] *file1 file2*
doscp [**-fvu**] *file1* [*file2* ...] *directory*

DESCRIPTION

doscp is the DOS counterpart of **cp** (see *cp*(1)). **doscp** copies a DOS file to a DOS or HP-UX file, an HP-UX file to an HP-UX or DOS file, or HP-UX or DOS files to an HP-UX or DOS directory. The last name in the argument list is the destination file or directory.

A DOS file name is recognized by the presence of an embedded colon (**:**) delimiter; see *dosif*(4) for DOS file naming conventions.

The file name **-** (dash) is interpreted to mean standard input or standard output depending upon its position in the argument list.

Options

doscp recognizes the following options:

- **-f** Unconditionally write over an existing file. In the absence of this option, **doscp** asks permission to overwrite an existing HP-UX file.
- **-v** Verbose mode. **doscp** prints the source name.
- **-u** Disable argument case conversion. In the absence of this option, all DOS file names are converted to upper case.

Note: Shell metacharacters (*****, **?**, and **[** ... **]**) can be used when specifying HP-UX file names, but cannot be used when specifying a DOS file name, because file name expansion is done by the shell and the DOS utilities do not recognize metacharacters.

RETURN VALUE

doscp returns 0 if all files are copied successfully. Otherwise, it prints a message to standard error and returns with a non-zero value.

EXAMPLES

Copy the files in the HP-UX directory **abc** to the DOS volume stored as HP-UX file **hard_disk**:

```
doscp abc/* hard_disk:
```

Copy DOS file **/backup/log** through the HP-UX special file **/dev/rfd9127** to HP-UX file **logcopy** located in the current directory:

```
doscp /dev/rfd9127:/backup/log  logcopy
```

Copy DOS file **zulu** on the volume stored as HP-UX file **bb** to standard output:

```
doscp bb:zulu -
```

SEE ALSO

cp(1), dos2ux(1), doschmod(1), dosdf(1), dosls(1), dosmkdir(1), dosrm(1), dosif(4).

NAME

dosdf - report number of free disk clusters

SYNOPSIS

`dosdf` *device*[`:`]

DESCRIPTION

`dosdf` is the DOS counterpart of the `df` command (see *df*(1)). It prints the cluster size in bytes and the number of free clusters on the specified DOS volume.

SEE ALSO

df(1), dos2ux(1), doschmod(1), doscp(1), dosls(1), dosmkdir(1), dosrm(1), dosif(4).

NAME

dosls, dosll - list contents of DOS directories

SYNOPSIS

dosls [**-aAudl**] *device* **:**[*file*]
dosll [**-aAudl**] *device* **:**[*file*]

DESCRIPTION

dosls is the DOS counterpart of **ls** (see *ls*(1)).

For each directory named, **dosls** lists the contents of that directory. For each file named, **dosls** repeats its name and any other information requested. If invoked by the name **dosll**, the **-l** (ell) option is implied.

Options

dosls and **dosll** recognizes the following options:

- **-a** List all directory entries. In the absence of this option, hidden files, system files, and files whose names begin with a dot (**.**) are not listed.
- **-A** Same as **-a**, except the current directory and the parent directory are not listed. For the superuser, this option defaults to being set, and is disabled by **-A**.
- **-u** Disable argument case conversion. In the absence of this option, all DOS file names are converted to uppercase.
- **-d** If an argument is a directory, list only its name. Often used with **-l** to get the status of a directory.
- **-l** List in long format, giving file attribute, size in bytes, and the date and time of last modification for each file, as well as listing the DOS volume label. Long listing is disabled if this option is used with the **dosll** command.

A DOS file name is recognized by the presence of an embedded colon (**:**) delimiter; see *dosif*(4) for DOS file naming conventions.

EXAMPLES

These examples assume that a DOS directory structure exists on the device accessed through HP-UX special file **/dev/rdsk/0s1**.

The following example lists all of the files in the root directory of the DOS directory structure:

```
dosls -a /dev/rdsk/0s1:
```

The following example produces a long-format listing of all the information about the DOS directory **/dos/math**, but does not list the files in the directory:

```
dosls -ld /dev/rdsk/0s1:/dos/math
```

SEE ALSO

dos2ux(1), doschmod(1), doscp(1), dosdf(1), dosmkdir(1), dosrm(1), ls(1), dosif(4).

NAME

dosmkdir - make a DOS directory

SYNOPSIS

`dosmkdir` [`-u`] *device*`:`*directory* ...

DESCRIPTION

`dosmkdir` is the DOS counterpart of the `mkdir` command (see *mkdir*(1)). It creates specified directories. The standard entries, `.` for the directory itself and `..` for its parent, are made automatically.

There is one option:

`-u` Disable argument case conversion. In the absence of this option, all DOS file names are converted to uppercase.

A DOS file name is recognized by the presence of an embedded colon (`:`) delimiter; see *dosif*(4) for DOS file naming conventions.

DIAGNOSTICS

`dosmkdir` returns 0 if all directories were successfully created. Otherwise, it prints a message to standard error and returns non-zero.

EXAMPLES

Create an empty subdirectory named **`numbers`** under the directory **`/math/lib`** on the device accessed through HP-UX special file **`/dev/rfd9122`**:

```
dosmkdir /dev/rfd9122:/math/lib/numbers
```

SEE ALSO

dos2ux(1), doschmod(1), doscp(1), dosdf(1), dosls(1), dosrm(1), mkdir(1), dosif(4).

NAME

dosrm, dosrmdir - remove DOS files or directories

SYNOPSIS

`dosrm` [`-friu`] *device* `:`*file ...*
`dosrmdir` [`-u`] *device* `:`*file ...*

DESCRIPTION

`dosrm` and `dosrmdir` are DOS counterparts of `rm` and `rmdir` (see *rm*(1) and *rmdir*(1), respectively).

`dosrm` removes the entries for one or more files from a directory. If a specified file is a directory, an error message is printed unless the optional argument `-r` is specified (see below).

`dosrmdir` removes entries for the named directories, provided they are empty.

Options

`dosrm` and `dosrmdir` recognize the following options:

`-f`	(force) Unconditionally remove the specified file, even if the file is marked read-only.
`-r`	Cause `dosrm` to recursively delete the entire contents of a directory, followed by the directory itself. `dosrm` can recursively delete up to 17 levels of directories.
`-i`	(interactive) Cause `dosrm` to ask whether or not to delete each file. If `-r` is also specified, `dosrm` asks whether to examine each directory encountered.
`-u`	Disable argument case conversion. In the absence of this option, all DOS file names are converted to uppercase.

A DOS file name is recognized by the presence of an embedded colon (`:`) delimiter; see *dosif*(4) for DOS file naming conventions.

EXAMPLES

These examples assume that a DOS directory structure exists on the device accessed through the HP-UX special file `/dev/rfd9122`.

Recursively comb through the DOS directory `/tmp` and asks if each DOS file should be removed (forced, with no file mode checks):

```
dosrm -irf /dev/rfd9122:/tmp
```

Remove the DOS directory `doug` from the DOS volume stored as HP-UX file `hard_disk`:

```
dosrmdir hard_disk:doug
```

SEE ALSO

dos2ux(1), doschmod(1), doscp(1), dosdf(1), dosls(1), dosmkdir(1), rm(1), rmdir(1), dosif(4).

(Requires Optional NS/9000 Software)

NAME

dscopy - copy files between NS systems

SYNOPSIS

`dscopy` *copydescriptor*

`dscopy -i`

DESCRIPTION

dscopy is a Network File Transfer (NFT) command that:

- Copies files between HP9000 HP-UX, HP1000, and HP3000 systems, and HP PCs in a network, provided each system is equipped with the appropriate networking software.
- Translates file attributes automatically as appropriate when transferring files between dissimilar systems, and when explicitly specified by the user.
- Can be used to access remote accounts if the user supplies the correct login and password information.

This manual entry discusses file transfers between HP-UX systems only. Refer to the *NS Cross-System NFT Reference Manual* for information about copying files between dissimilar systems.

File Copying Formats

NFT uses two file copying formats:

Transparent Format	Default for transfers between similar systems, even if the transfer is initiated from a third, dissimilar system. Each file is copied directly from one system to the other without alteration.
Interchange Format	Default for transfers between dissimilar systems such as between HP1000 systems and 3000 or HP-UX systems. Files being copied are converted to Interchange Format which redefines file attributes such that converted files can be handled by any NS system. Interchange Format can be activated explicitly by specifying one or more of the Interchange Format options when copying a file. Interchange Format options can also be used to give a target file a different set of attributes from those characteristic of the source file from which it was copied, even if the files are being copied between computers of the same type.

Transfers Between HP-UX Systems

When transferring files between HP-UX systems:

- File mode assigned to new file is the same as that of the source file.
- Ownership of the new file is set to the user ID used for the destination system login.
- If the destination system file already exists and is write protected against the destination system file node user ID, *dscopy* returns an error message. To override write protection of an existing file, use the `-r` (replace) option described later in this manual entry.
- When processing user login information for a remote HP-UX system, the local HP-UX system prompts for a password with local echo turned off if the login name is followed by a colon but no password is given. The colon character tells the system to prompt you for the password. The login name and password are then passed to the remote computer as two separate entities.
- HP-UX *quit* and *interrupt* signals can be used during *dscopy* operations. *quit* aborts any transfer in progress and *interrupt* reports the percentage of completion of the current file transfer.

System Security

Security varies between different types of NS systems. Refer to the *NS Cross-System NFT Reference Manual* for security information if the source and target nodes are not similar systems.

Interactive and Non-Interactive Transfers

dscopy can be used either interactively or non-interactively (such as from a shell script or program). For interactive transfers, the `-i` option is used with no other arguments. Non-interactive transfers use a *copydescriptor* string to specify source, destination, and transfer options.

Interactive Transfers
To initiate an interactive transfer, use the **-i** option as follows:

```
dscopy -i
```

In Interactive Mode, *dscopy* prompts for a *copydescriptor* that specifies:

- Type of transfer,
- Login information for source and destination systems as appropriate,
- source and destination file locations.

dscopy then copies the specified file and returns with a prompt for the next *copydescriptor*, continuing until an end-of-file character (usually Ctrl-D) is received from the keyboard indicating that no more transfers are to be made. (End-of-file character is configurable and may vary between systems.) Upon completion of each file transfer, the connection between the source and target system is maintained, making it unnecessary to respecify node names and logins if subsequent copies are between the same two systems. To change the source or target node from a remote node to the local node, add a prefix of # to the local file name. To change the source and/or target nodes to different remote nodes, respecify the node name and login for each node being changed. Whenever the source or target node specification is changed, NFT terminates the existing connection and establishes a new connection.

Each *copydescriptor* is constructed as follows:

[*option*] [*option*]... [[*snode#*]*slogin#*]*sfile* [[*tnode#*]*tlogin#*]*tfile*

Parameters in *copydescriptor* are:

option Any of the *dscopy* options described later in this entry. Multiple options, if used, can appear in any order, and must be separated by a space.

snode
tnode Name of the source node or target node, respectively. NS node names have the following hierarchical structure:

node [*.domain* [*.organization*]]

organization, *organization* and *domain*, or all parts of the node name can be omitted. When *organization*, or *organization* and *domain*, are omitted, the default is the local organization and/or domain. If the entire node name is omitted, default is the local node.

slogin
tlogin Login and password, if any, at the source or target node, respectively. The login for HP-UX systems must be in the form:

userID **:** *password*

where *userID* is the login name on the source or target system, and *password* is the password, if any. The colon and *password* can be omitted if no password exists for the login name (a practice that should be strictly avoided unless system security is not important). If the colon is typed but *password* is not, the local system prompts for the password with local echo turned off.

The login must be separated from the *snode* or *tnode* and *sfile* or *tfile* parameters by pound signs (**#**). If *slogin* or *tlogin* is omitted, the corresponding login defaults to the local login.

sfile
tfile Path name of the source or target file, respectively. If the file is a local file, the path name can start from the current working directory or from the local system root directory. If the file is on a remote HP-UX system, the path name can start from the home directory of the login specified in the *slogin* or *tlogin* parameter, or from the remote root directory. HP-UX path names can be up to 1023 bytes long. NFT does not support the use of HP-UX wild card characters.

Login and file name syntax varies on non-HP-UX systems. Refer to the *NS Cross-System NFT Reference Manual* for login and file name syntax on other systems.

Non-Interactive Transfers
dscopy behaves much the same way in non-interactive transfers except that each transfer is an isolated

activity where *dscopy* establishes a connection between the source and destination system, transfers the specified file or files, then terminates the connection between systems upon completion of the transfer.

Non-interactive *dscopy* commands are constructed as follows:

`dscopy` [*option*] [*option*]... [[*snode#*]*slogin#*]*sfile* [[*tnode#*]*tlogin#*]*tfile*

For a description of parameters shown, refer to the description of *copydescriptor* in the previous discussion of interactive transfers.

Options

There are two classes of options to the *dscopy* command. They are part of *copydescriptor* and precede the file and node specifications.

File Transfer Options	Used to control output of status information and to overwrite existing files on the destination (target) system.
File Format Options	Used to alter file formats and attributes when copying files between dissimilar systems or to expressly modify file attributes when copying files on a single system or between similar systems.

Although Interchange Format options can be used when copying files on a given system or between similar systems, they are primarily intended for use when copying files between dissimilar systems and are most useful when used for that purpose. Using an Interchange Format option when copying a file between HP-UX systems overrides the default file copying format (Transparent Format) and causes the file to be copied in Interchange Format.

File Transfer Options

The two file transfer options are:

-p (Print Status) Prints *dscopy* status to the standard output. If **-p** is not present, no status information is produced.

-r (Replace) Replaces target file, if it exists, with the file being copied. Behavior of this option is determined by system configuration at the source and destination file nodes.

HP-UX-to-HP-UX	Mode and ownership of the target file are those of the source file.
HP-UX-to-non-HP-UX	Target file protection and/or ownership are the default values defined for the target login.
Non-HP-UX-to-HP-UX	Target file is set to mode 666 (rw-rw-rw-).
Non-HP-UX-to-non-HP-UX	Target file protection and/or ownership are the default values defined for the target login.

If the **-r** option is not used and:

HP-UX Target File Exists
File is automatically overwritten by the source file (unless the target file is write protected against the target node login). Target file retains ownership and file mode of the overwritten file.

Non-HP-UX Target File Exists
File is not overwritten and an error is returned.

Interchange Format Options

The following paragraphs describe behavior of Interchange Format options when source and target nodes are both HP-UX systems. Refer to the *NS Cross-System NFT Reference Manual* for information about other systems.

-A (ASCII option) Intended for use when copying files between dissimilar systems. Not useful when coping files between HP-UX systems.

-B (Binary option) Used for transferring files containing binary data (formatted in 256-byte records). Not useful when copying files between HP-UX systems unless used in conjunction with other Interchange Format options (see **-F** and **-L** options below).

-F	(Fixed-Length option) Converts "records" in an HP-UX ASCII source file to fixed-length "records" in an HP-UX target file (an HP-UX "record" is considered to be the data found between ASCII LF characters). This option truncates data or adds ASCII space characters between ASCII LF characters in the ASCII source file as necessary, so that the data is 160-bytes long (the default record length). If the **-B** option is used in conjunction with **-F**, source file data is divided into records every 256 bytes, and ASCII LF characters found in the source file are considered part of the data. If the last record is less than 256 bytes, it is padded with ASCII NUL characters.
-L or **-L**[*n*]	(Record Length option) Used to change the length of records in an HP-UX ASCII or Binary target file to *n* bytes. If *n* is not specified, default is 160 bytes for ASCII files and 256 bytes for Binary files. If *n* is zero, the record length is set equal to the longest record in the source file. Data between ASCII LF characters in the source file is truncated, if necessary, so that the record does not exceed *n* bytes in length. This option interprets an HP-UX record as the data found between ASCII LF characters for ASCII files.
-d **-d0***nnn* or **-d***c*	(Delimiter Character option) Specifies a delimiter character to be inserted after each record in the target file where *nnn* is a decimal equivalent of the ASCII value of the character to be used as a delimiter (must be preceded by a zero), or *c* is the character itself. The **-d** option with no argument defaults to an ASCII LF.
-s **-s0***nnn* or **-s***c*	(Search character option) Specifies a character to be recognized by NFT as a delimiter character when searching through records in the source file. Interpretation of *nnn* and *c* is the same as for the **-d** option.

RETURN VALUE

Exit status values are returned to **$?** for Bourne and Korn shells and **$status** for C shell. Values are:

0	Requested transfer completed successfully.
1	Source file or source node is inaccessible.
2	Target file or target node is inaccessible.
3	Syntax error.
4	Transfer began but did not complete successfully.
5	Internal error.

DIAGNOSTICS

Error messages are printed to standard error. These messages are listed in the *Using Network Services* manual.

EXAMPLES

Interactively copy two files from a remote HP-UX system to the local HP-UX system. After the first file is copied, use the established connection to copy a second file between the same two nodes (it is unnecessary to specify the remote node name and login parameters for the second transfer).

```
dscopy -i
dscopy>remnode1#logname#/users/lab/sfile  /users/testsite/d1file
dscopy>/users/lab/nextfile  /users/testsite/d2file
```

Same as above, but non-interactive mode. Note that the remote node name and login parameters are required in both commands.

```
dscopy remnode1#logname#/users/lab/sfile /users/testsite/d1file
dscopy remnode1#logname#/users/lab/nextfile /users/testsite/d2file
```

AUTHOR

dscopy was developed by HP

SEE ALSO

cp(1), proxy(1M), uucp(1C).

NAME

du - summarize disk usage

SYNOPSIS

`du [-a|-s][-brx][-t` *type* `][`*file* ...`]`

DESCRIPTION

`du` gives the number of 512-byte blocks allocated for all files and (recursively) directories within each directory and file specified by the *file* operands. The block count includes the indirect blocks of the file. A file with two or more links is only counted once. If *file* is missing, `.` is used.

By default, `du` generates an entry only for the *file* operands and each directory contained within those hierarchies.

Options

`du` recognizes the following options:

-a Print entries for each file encountered in the directory hierarchies in addition to the normal output.

-b For each *file* operand that is a directory for which file system swap has been enabled, print the number of blocks the swap system is currently using.

-r Print messages about directories that cannot be read, files that cannot be accessed, etc. `du` is normally silent about such conditions.

-s Print only the grand total of disk usage for each of the specified *file* operands.

-x Restrict reporting to only those files that have the same device as the file specified by the *file* operand. Disk usage is normally reported for the entire directory hierarchy below each of the given *file* operands.

-t *type*
Restrict reporting to file systems of the specified *type*. Accepted *type*s are **hfs**, **cdfs**, and **nfs** (see *checklist*(4)). Multiple **-t** *type* options can be specified. Disk usage is normally reported for the entire directory hierarchy below each of the given *file* operands.

EXAMPLES

Display disk usage for the current working directory and all directories below it, generating error messages for unreadable directories:

```
du -r
```

Display disk usage for the entire file system except for any `cdfs` or `nfs` mounted file systems:

```
du -t hfs /
```

Display disk usage for files on the root volume (/) only. No usage statistics are collected for any other mounted file systems:

```
du -x /
```

WARNINGS

Block counts are incorrect for files that have holes in them.

SEE ALSO

df(1), bdf(1), quot(1m), checklist(4).

STANDARDS CONFORMANCE

`du`: SVID2, XPG2, XPG3

NAME

echo - echo (print) arguments

SYNOPSIS

`echo` [*arg*] ...

DESCRIPTION

`echo` writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of `\`:

`\b`	backspace
~~`\c`~~	~~print line without appending a new-line~~
`\f`	form-feed
`\n`	new-line
`\r`	carriage return
`\t`	tab
`\v`	vertical tab
`\\`	backslash
`\`*n*	the 8-bit character whose ASCII code is the 1-, 2-, 3- or 4-digit octal number *n*, whose first character must be a zero.

`echo` is useful for producing diagnostics in command files and for sending known data into a pipe.

EXTERNAL INFLUENCES

Environment Variables **LC_CTYPE** determines the interpretation of *arg* as single and/or multi-byte characters.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `echo` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

SEE ALSO

sh(1).

NOTES

Berkeley `echo` differs from this implementation. The former does not implement the backslash escapes. However, the semantics of the `\c` escape can be obtained by using the `-n` option. The echo command implemented as a built-in function of `csh` follows the Berkeley semantics (see *csh*(1)).

BUGS

No characters are printed after the first `\c`. This is not normally a problem.

STANDARDS CONFORMANCE

`echo`: SVID2, XPG2, XPG3, POSIX.2

NAME

ed, red - text editor

SYNOPSIS

`ed [-s | -] [-p` *string* `] [-x] [` *file* `]`

`red [-s | -] [-p` *string* `] [-x] [` *file* `]`

DESCRIPTION

`ed` is a line-oriented text editor. It is most commonly used in scripts and non-interactive editing applications because, even though it can be used interactively, other editors such as *vi* and *ex* are typically easier to use in an interactive environment.

If *file* is specified, `ed` simulates an `e` command (see below) on the named file; that is to say, the file is read into `ed`'s buffer so that it can be edited.

Options

The following options are recognized:

`-s`	Suppress printing of byte counts by `e`, `E`, `r`, and `w` commands, and suppress the `!` prompt after a `!` *command*.
`-`	Same as `-s` option. The `-` option is obsolescent and will be removed in a future release.
`-p` *string*	Use *string* as the prompt string when in command mode. By default, there is no prompt string.
`-x`	Simulate an `x` command first to handle an encrypted file.

File Handling

`ed` operates on a copy of the file it is editing; changes made to the copy have no effect on the original file until a `w` (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

`red` is a restricted version of `ed` that only allows editing of files in the current directory and prohibits executing shell commands via `!`*shell command*. Attempts to bypass these restrictions result in a `restricted shell` error message.

Both `ed` and `red` support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking `ed` with the controlling terminal in `stty -tabs` or `stty tab3` mode (see *stty*(1)), the specified tab stops are automatically used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: When inputting text, `ed` expands tab characters as they are typed to every eighth column as a default.

Editor Commands Structure

Commands to `ed` have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command is allowed on a line. Append, change, and insert commands accept text input which is then placed in the buffer as appropriate. While `ed` is accepting text following an append, change, or insert command, it is said to be in *input mode*. While in input mode, *no* editor commands are recognized; all input is merely collected. To terminate input mode, type a period (`.`) alone at the beginning of a line.

Regular Expressions

`ed` supports the Basic Regular Expression (RE) syntax (see *regexp*(5)) with the following additions:

- The null RE (e.g., `//`) is equivalent to the last RE encountered.
- If the closing delimiter of a RE or of a replacement string (e.g., `/`) would be the last character before a new-line, that delimiter can be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2          g/s1          ?s1
s/s1/s2/p        g/s1/p        ?s1?
```

Line Addresses

To understand line addressing, remember that `ed` maintains a pointer to the default *current line*. Generally speaking, the current line is the last line affected by a command. The exact effect of a given command on the current line is discussed under the description of each command. Addresses are interpreted according to the following rules:

1. The character **.** refers to the current line.
2. The character **$** refers to the last line of the buffer.
3. A decimal number *n* refers to the *n*-th line of the buffer.
4. A **'***x* refers to the line marked with the mark name character *x*, which must be a lowercase letter. Lines are marked with the **k** command described below.
5. A RE enclosed by slashes (**/***RE***/**) refers to the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. (Also see WARNINGS below.)
6. A RE enclosed in question marks (**?***RE***?**) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. (Also see WARNINGS below.)
7. An address followed by a plus (**+**) or minus (**-**) sign followed by a decimal number specifies that address plus or minus the indicated number of lines. The plus sign can be omitted.
8. If an address begins with **+** or **-**, the addition or subtraction is calculated with respect to the current line. For example, **-5** is interpreted as **.-5**.
9. If an address ends with **+** or **-**, 1 is added to or subtracted from the address, respectively. As a consequence of this and rule 8 above, the address **-** refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the circumflex (**^**) and **-** characters are interpreted identically when encountered in addresses.) Moreover, multiple trailing **+** and **-** characters have a cumulative effect, so **- -** refers to the second line preceding the current line.
10. For convenience, a comma (**,**) represents the address pair **1,$**, while a semicolon (**;**) represents the pair **.,$**.

Commands require zero, one, or two addresses. Commands that do not use addresses treat the presence of an address as an error. Commands that accept one or two addresses assume default addresses when the number of addresses specified is insufficient. If more addresses are specified than a given command requires, the last one or two are used as appropriate.

Addresses are usually separated from each other by a comma (**,**). They can also be separated by a semicolon (**;**), in which case the current line (**.**) is set to the first address, after which the second address is calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5 and 6 above). The second address of any two-address sequence must correspond to a line in the buffer that follows the line corresponding to the first address.

Editor Commands

In the following list of `ed` commands, the default addresses are shown in parentheses (parentheses are *not* part of the address and should not be placed in an actual command except for other purposes).

It is generally illegal for more than one command to appear on a line. However, any command (except **e**, **f**, **r**, or **w**) can be suffixed by **l**, **n**, or **p** in which case the current line is respectively either listed, numbered, or printed, as discussed below under the **l**, **n**, and **p** commands.

(.)a
<text>

The a (append) command reads *<text>* and appends it after the addressed line. Upon completion, the new current line is the last inserted line, or, if no text was added, at the addressed line. Address 0 is legal for this command, causing the appended *text* to be placed at the beginning of the buffer.

.

(.,.)c
<text>
.
: The c (change) command deletes the addressed lines then accepts input text to replace the deleted lines. Upon completion, the new current line is the last line in *text* or, if no text was provided, at the first line after the deleted line or lines.

(.,.)d
: The d (delete) command deletes the addressed lines from the buffer. Upon completion, the new current line is the first line following the deleted text, or the last line in the file if the deleted line or lines were at the end of the buffer.

e *file*
: The e (edit) command deletes the entire contents of the buffer, then reads in the named *file*. Upon completion, the new current line is the last line in the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the f command). The number of characters read is displayed, and *file* is remembered for possible use as a default file name in subsequent e, r, or w commands. If the *file* name starts with !, the rest of the line is interpreted as a shell command whose standard output is to be read. Such a shell command is *not* remembered as the current file name. (Also see DIAGNOSTICS below.)

E *file*
: The E (forced edit) command is identical to e except that no check is made to ensure that the current buffer has not been altered since the last w command.

f *file*
: If *file* is specified, the f (file-name) command changes the currently-remembered file name to *file*. Otherwise, it prints the currently-remembered file name.

(1,$)g/*RE*/*command list*
: The g (global) command first marks every line that matches the given RE. Then, for every such line, the given *command list* is executed with the current line initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multiple-line list except the last line must end with a backslash (\). a, i, and c commands and associated input are permitted. The . that normally terminates input mode can be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the p command. The g, G, v, and V commands are *not* permitted in the *command list*. (Also see WARNINGS below.)

(1,$)G/*RE*/
: The interactive G (Global) command first marks every line that matches the given RE. Then, for every such line, the line is printed, then the current line is changed to that line and any *one* command (other than one of the a, c, i, g, G, v, and V commands) can be input after which it is executed. After executing that command, the next marked line is printed, and so on. A new-line character acts as a null command, and an & causes re-execution of the most recent command executed within the current invocation of G. Note that the commands input as part of the execution of the G command may address and affect *any* lines in the buffer. The G command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h
: The h (help) command gives a short error message explaining the reason for the most recent ? diagnostic.

H
: The H (Help) command causes ed to enter a mode in which error messages are printed for all subsequent ? diagnostics. It also explains the previous ? if there was one. The H command alternately turns this mode on and off. Initial default is off.

(.)i
<text>
.
: The i (insert) command inserts the given *text* before the addressed line. Upon completion, the current line is the last inserted line, or, if there were none, the addressed line. This command differs from the a command only in the placement of the input text. Address 0 is not legal for this command.

(.,.+1)j
: The j (join) command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*
: The k (mark) command marks the addressed line with name *x* which must be a lowercase letter. The address '*x* then addresses this line. Upon completion, the new current line remains unchanged from before.

(.,.)l
: The l (list) command writes the addressed lines to standard ouput in a visually unambiguous form. Characters listed in the following table are written as the corresponding escape

sequence. Non-printable characters not in the table are written as a three-digit octal number (with a preceding backslash character) for each byte in the character (most significant byte first).

Long lines are folded with the point of folding indicated by writing a backslash character followed by a new-line. The end of each line is marked with a `$`. An `l` (ell) command can be appended to any other command other than `e`, `E`, `f`, `q`, `Q`, `r`, `w`, or `!`. The current line number is set to the address of the last line written.

Escape Sequence	Represents	Escape Sequence	Represents
`\\`	backslash	`\r`	carriage-return
`\a`	alert	`\t`	horizontal tab
`\b`	backspace	`\v`	vertical tab
`\f`	form-feed		

`(.,.)m`*a* The `m` (move) command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a*, causing the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; Upon completion, the new current line is the last line moved.

`(.,.)n` The `n` (number) command prints the addressed lines, preceding each line by its line number and a tab character. Upon completion, the new current line is the last line printed. The `n` command can be appended to any other command other than `e`, `f`, `r`, or `w`.

`(.,.)p` The `p` (print) command prints the addressed lines. Upon completion, the new current line is the last line printed. The `p` command may be appended to any other command other than `e`, `E`, `f`, `q`, `Q`, `r`, `w`, or `!`. For example, `dp` deletes the current line and prints the new current line.

`P` The `P` (prompt) command causes `ed` to prompt with an asterisk (`*`) (or *string* if `-p` option is specified in the command line) for all subsequent commands. for all subsequent commands. The P command alternately turns this mode on and off; it is initially on if the `-p` option is specified, otherwise off. The current line number is unchanged.

`q` The `q` (quit) command causes `ed` to exit. No automatic write of a file is done (but see DIAGNOSTICS below).

`Q` The editor unconditionally exits without checking for changes in the buffer since the last `w` command.

`($)r` *file* The `r` (read) command reads the specified *file* into the buffer after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see `e` and `f` commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since `ed` was invoked. Address 0 is legal for `r` and places the contents of *file* at the beginning of the buffer. If the read is successful, the number of characters read is displayed; Upon completion, the new current line is the last line read into the buffer. If the *file* name starts with `!`, the rest of the line is interpreted as a shell command whose standard output is to be read. For example, `$r !ls` appends a listing of files in the current directory to the end of the file being edited. A shell command is *not* remembered as the current file name.

`(.,.)s`*/RE/replacement/flags*
The `s` (substitute) command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by *replacement* if the global replacement indicator `g` appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number *n* appears after the command, only the *n*th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line can be used instead of `/` to delimit the RE and *replacement*. Upon completion, the new current line is the last line on which a substitution occurred. (Also see WARNINGS below.)

If an ampersand (`&`) appears in *replacement*, it is replaced by the string matching the RE on the current line. The special meaning of `&` in this context can be suppressed by preceding it with `\`. As a more general feature, the characters `\`*n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between `\(` and `\)`. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of `\(` starting from the left. When the character `%` is the only character in *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The `%` loses its special meaning when it is in a replacement string containing more than one character or when preceded by a `\`.

A line can be split by substituting a new-line character into it. The new-line in *replacement* must be escaped by preceding it by `\`. Such substitution cannot be done as part of a `g` or `v` command list.

The value of *flags* is zero or more of:

count	Substitute for the *count*'th occurrence only of the *RE* found on each addressed line.
`l`	Write to standard output the final line in which a substitution was made. The line is written in the format specified for the `l` command.
`n`	Write to standard output the final line in which a substitution was made. The line is written in the format specified for the `n` command.
`p`	Write to standard output the final line in which a substitution was made. The line is written in the format specified for the `p` command.

(.,.) *ta*
Same as `m` command, except that a *copy* of the addressed lines is placed after address *a* (which can be 0). Upon completion, the new current line is the last line of the copy.

`u`
The `u` (undo) command nullifies the effect of the most recent command that modified anything in the buffer; that is, the most recent `a`, `c`, `d`, `g`, `i`, `j`, `m`, `r`, `s`, `t`, `v`, `G`, or `V` command. All changes made to the buffer by a `g`, `G`, `v`, or `V` global command are "undone" as a single change; if no changes were made by the global command (such as with `g/`*RE*`/p`), the `u` command has no effect. The current line number is set to the value it had immediately before the command started.

(1,$) v/*RE*/*command*
Same as the global command `g` except that lines marked during the first step are those that do *not* match the RE.

`(1,$)V/`*RE*`/`
This command is the same as the interactive global command `G` except that the lines that are marked during the first step are those that do *not* match the RE.

`(1,$)w`*file*
The `w` (write) command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless the current `umask` setting dictates otherwise (see *umask*(1). The currently-remembered file name is *not* changed unless *file* is the very first file name encountered since `ed` was invoked. If no file name is given, the currently-remembered file name, if any, is used (see `e` and `f` commands); Upon completion, the current line address is unchanged. If the command is successful, the number of characters written is displayed. If the *file* name starts with `!`, the rest of the line is interpreted as a shell command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X
A key string is demanded from the standard input. Subsequent `e`, `r`, and `w` *commands* will encrypt and decrypt the text with this key, using the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)=
The line number of the addressed line is displayed; Current line address is unchanged by this command.

`!`*shell command*
The remainder of the line after the `!` is sent to the shell (specified by the `SHELL` environment variable;

`/bin/sh` is used if `SHELL` is not set) to be interpreted and executed as a command. Within the text of that command, the unescaped character `%` is replaced with the remembered file name. If a `!` appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, `!!` repeats the last shell command. If any expansion is performed, the expanded line is echoed. Upon completion, current line address is unchanged.

(`.+1`) <new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to *.+1p*. This technique is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, `ed` prints a `?` and returns to *its* command level.

The following size limitations apply: 256 characters per global command list, 64 characters per file name, and 32M characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

EXTERNAL INFLUENCES

Environment Variables

`SHELL` determines the preferred command-line interpreter for use in all `!`-style commands. If this variable is null or not set, `sh` is used (see *sh*(1)).

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions.

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters, the classification of characters as non-printing, and the characters matched by character class expressions in regular expressions.

`LANG` determines the language in which messages are displayed.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `ed` behaves as if all internationalization variables are set to "C". See *environ*(5).

When set, the `TMPDIR` environment variable specifies a directory to be used for temporary files, overriding the default directory `/tmp`.

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`?`	Command error. Use `h` or `H` for detailed explanations.
`?`*file*	Inaccessible file. Use `h` or `H` for detailed explanations.

If changes have been made in the buffer since the last `w` command that wrote the entire buffer, `ed` warns the user if an attempt is made to destroy the buffer by means of an `e` or `q` command. `ed` displays `?` then continues normal editing unless a second `e` or `q` command is given, in which case the second command is executed. The `-s` or `-` command-line option inhibits this feature.

EXAMPLES

Make a simple substitution in `file_1` from a shell script, changing the first occurrence of `abc` in any line to `xyz`, and save the changes in `file_2`.

```
cat - << EOF | ed -s file_1 1,$ s/abc/xyz/ w file_2 q EOF
```

FILES

`/tmp/e#`	temporary buffer file where # is the process number.
`ed.hup`	work is saved here if the terminal is hung up.

SEE ALSO

awk(1), crypt(1), edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), vi(1), fspec(4), lang(5), regexp(5).

The ed Editor in *Text Processing Users Guide*.

WARNINGS

A `!` command cannot be subject to a `g` or a `v` command.

The `!` command and the `!` escape from the `e`, `r`, and `w` commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).

The sequence `\n` in a regular expression does not match a new-line character.

The `l` command does not handle DEL correctly.

Files encrypted directly with the `crypt` command with the null key cannot be edited (see *crypt*(1)).

If the editor input is coming from a command file (i.e., `ed file < ed-cmd-file)`, the editor exits at the first failure of a command in the command file.

When reading a file, `ed` discards ASCII NUL characters and all characters after the last new-line. This can cause unexpected behavior when using regular expressions to search for character sequences containing NUL characters or text near end of file.

STANDARDS CONFORMANCE

`ed`: SVID2, XPG2, XPG3, POSIX.2

`red`: SVID2, XPG2, XPG3

NAME

elm - process mail through screen-oriented interface

SYNOPSIS

`elm [-akKmz] [-f` *folder* `]`
`elm [-s` *subject* `]` *address-list*
`elm -h`

DESCRIPTION

`elm` is a screen-oriented electronic mail processing system. In interactive use, the main header index and mini-menu of commands are displayed upon initial invocation and at any point when the program is waiting for input.

There are three main ways to use `elm`:

- Explicitly send a single message by invoking `elm` with a list of mail addresses; `elm` then prompts for the subject, message body, and so on.
- Conveniently send files or the output of commands via command line redirection.
- Use `elm` as an interactive mail interface program (see EXAMPLES).

Options

The following options are recognized:

-a	Arrow - force the arrow cursor (instead of the inverse bar).
-f *folder*	File - read specified *folder* file rather than the incoming mailbox.
-h	Help - give a list of starting options.
-k	Softkeys off - disable use of softkeys (function keys).
-K	Keypad and softkeys off - disable use of softkeys and arrow cursor keys. If your terminal does not have HP 2622 function key protocols, this option is required.
-m	Menu off - Use the extra lines for more message headers.
-s *subject*	Subject - specify subject for message being sent.
-z	Zero - do not enter `elm` if no mail is pending.

INTERACTIVE (Command Mode) OPERATION

When `elm` is invoked without an address-list, it activates an interactive screen-oriented interface and enters command mode. In command mode, `elm` executes each user command on the message corresponding to the current message pointer (an inverse-video bar or arrow). Some commands can also operate on "tagged" messages. To execute a command, use the `j` and `k` keys to move the pointer to the appropriate message, then type the character for the desired command (Commands are discussed later in this entry).

Message Status

The header index list always displays the status of each message. The status field is composed of three separate character fields, the first of which indicates temporary status:

D Indicates a *deleted* message.

E Identifies an *expired* message. This flag is set according to the header field **`Expires:`**. If the date of this field is older than the today, this flag appears. `elm` accepts the following date formats:

- `Mon, 11 Jun 90`
- `Jun 11, 90`
- `11 Jun, 90`
- `900611`*HHMMZ* (ISO X.400 format)

N
Identifies a *new* message.

The second character field indicates permanent status:

`C`	For *confidential* mail. If `Sensitivity: 3` of the user-defined header field is attached, this can appear and the message is considered company confidential, as specified by the ISO X.400 standard.
`U`	For *urgent* mail. This flag is set if the message contains a `Priority:` header field.
`P`	For *private* mail. This flag is also associated with the `Sensitivity:` header field and is set if the header field contains `Sensitivity: 2`.
`A`	For messages that have an explicit action associated with them through inclusion of the `Action:` header field.
`F`	For a *form* letter.

When a message has more than one status flag of a particular type set, the highest-precedence indicator is displayed on the index page. For example, if a *form* letter is also marked as `company confidential`, the `C` rather than the `F` status character is displayed.

A `+` in the third status-character field indicates that the message is *tagged* (see Commands later in this entry)

Sending Messages

When sending a message, `elm` uses the editor specified in your `.elm/elmrc` file (see CUSTOMIZATION), the editor listed as `$EDITOR` in your environment, or `vi` as a default if neither is defined. If `builtin` is specified as your editor, a large set of commands is available while composing your message (see Tilde Escapes).

If the file `$HOME/.elm/elmheaders` exists, `elm` automatically reads in the contents of the file and adds it to the headers of all outbound mail (useful for adding an `Organization:` field, `Phone:` field, etc). `elm` also supports the use of backquotes in the `elmheaders` file, so an entry of the form:

```
Operating System: 'uname -srv'
```

can be used. Lines in the `elmheaders` file should contain a header string, followed by a colon, followed by a contents string.

Aliases

`elm` has its own alias system that supports both personal and system-wide aliases. Personal aliases are specific to a single user; system aliases are available to everyone on the system where the alias resides (see *elmalias*(1)). To enter the elm alias mode, use the `A` or `a` (alias) command at the main command prompt. You can then create and save an alias for the current message, check personal and system-wide aliases, and access other options (see Alias Commands).

When invoked, `elm` reads customized variables from file `$HOME/.elm/elmrc` (if it exists) to initialize parameters. This file can be saved from within `elm` and some of these variables can also be modified by the `O` or `o` (option) command (see CUSTOMIZATION).

FORMS MODE

One feature that is unique to `elm` is the ability to compose and reply to *form* letters and other types of forms.

To create a form message, first enable forms mode by adding `forms=ON` to file `$HOME/.elm/elmrc` (see CUSTOMIZATION). Also set the `userlevel` to intermediate (level 1) or advanced (level 2). This can be done either in the `elmrc` file or via the options command. As you compose the message, each field to be filled in by recipient should have a colon (`:`) followed by either the number of spaces allowed for the field, or a newline which indicates fields through the end of the line. Additionally, if a colon appears on a line by itself, the recipient is prompted for multi-line input. After the message has been created, select the `make form` option before sending the message.

Upon receipt of a form message, the user can reply (but not group reply), at which time `elm` prompts the user for each field, with any text present between the fields displayed as appropriate.

Here is an example of a simple form message:

```
----
      On-Line Phone and Address Database
      Please fill out and return as
      Name:
      Manager:
      Department:                    Division:
      Your home address
      ...
      home phone number:
      Thank you for your cooperation.
----
```

MAILER COMMANDS

`elm` recognizes the following keyboard commands:

?	Help. Press `?` once to enter *explain key* mode where pressing any key produces a one-line description of what the key does. Press `?` again to obtain a summary listing for each command available. To return to the main menu level, press ESC or type a period (`.`).
!	Shell Escape. Used to send a command to the shell without leaving `elm`.
|	Pipe. Pipes the current message or the set of *tagged* messages through other filters as desired. The shell used for the entire command is either the one specified in file `.elm/elmrc` if it exists, or `/bin/sh` otherwise.
+	Next index page. Displays the next header index page, when applicable.
-	Previous index page. Displays previous header index page, if applicable.
=	Set current to 1. Sets current message pointer to first message.
*	Set current to last. Sets current message pointer to last message.
$	Resync. You can decide whether or not to actually delete messages previously marked for deletion without quitting `elm`.
/pattern	Pattern match. On the top level, this command allows you to search through all the `from` and `subject` lines of the current mailbox. If the first character of *pattern* is also a `/`, `elm` tries to match *pattern* against *any* line in the mailbox. Both searches treat uppercase and lowercase as equivalent.
n	Specify new current message. Typing the message number causes `elm` to produce the prompt `Set current to:` *n*, where *n* is the message number typed. Note that changing the current message to a message not on the current page of headers results in a new page being displayed.
Return	Read current message. When the Return key is pressed, the screen is cleared and the current message is displayed by the pager specified by the user environment variable `$PAGER`, by `pager` in file `.elm/elmrc`, or the system default (see CUSTOMIZATION).
<	Scan message for calendar entries. A rather novel feature of the `elm` mailer is its ability to automatically incorporate calendar or agenda information from a mail message into the user's calendar file. This is done quite simply; any message that contains the pattern -> *calendar entry* or

- *multi-line*
- *calendar entry*

is automatically added to the user's `calendar` file after stripping the `->` or `-` if the `<` command is used (see CUSTOMIZATION).

`a`	Alias. The *alias* command is a means by which more complex mail addresses can be shortened for the mail user. In alias mode, it provides the `a` command for creating an alias for the current message, `p` for checking an alias, `e` for checking an alias with full expanded addresses, `s` for checking system aliases, `u` for checking user aliases, `m` for making a new alias, and `d` for deleting an alias (see also Alias Commands).
`b`	Bounce mail. This "remails" mail to someone else in such a way as to make the return address the original sender rather than you (as opposed to the *forward* command, which makes the return address *you* rather than the original sender).
`c`	Change mailbox. This command is used to change the mailbox file that is currently being read. As with the `save` command, this command expands filenames with `~` being your home directory and `=` being your `maildir` directory, if defined (see CUSTOMIZATION). This command also recognizes the special character `!` which can be used to change the default incoming mailbox.
`d`, `u`	Delete and Undelete. Neither of these commands has any prompt. Action is indicated by either adding a `D` to the current message index entry (indicating deletion pending) or removing the `D` (indicating that the message is not marked for deletion).
Ctrl-D	This command marks all messages that contain a specific *from* or *subject* pattern for deletion. When Ctrl-D is pressed, `elm` prompts for the string to match in either the `from` or `subject` line of each message).
Ctrl-U	This is the direct opposite Ctrl-D. This command removes any mark for deletion from all messages that match the specified pattern.
`f`	Forward. This command is used to forward the current message to another person. The message is copied into the edit buffer where you can add your own message before sending if you desire to do so (also see `bounce` above).
`g`	Group reply. Identical to `reply` below, except that the response is mailed to *all recipients* of the original message (see CUSTOMIZATION `alternatives`)
`h`	Same as **Return**, except that the message is displayed with all headers.
`j`, `k`	Similar to `j` and `k` commands in `vi` and other screen-oriented programs. `j` key moves the current message pointer down to the next message; `k` key moves the message pointer to the previous message. Uppercase `J` and `K` behave in the same manner as their lowercase counterparts, even when the `skipdeleted` option is ON (see CUSTOMIZATION).
`l`	(ell) Limit. This specifies a subset of the existing messages to be dealt with. It is valid for `subject`, `from`, and `to` fields. To set the criteria, type `"subject` *string*`"`, `"from` *string*`"`, or `"to` *string*`"`. To clear all the criteria and restore the normal display, type `"all"` as the limiting criteria.
Ctrl-L	Rewrite the screen. If the screen is confused, you can redraw screen with this command (same as in `vi` editor).
`m`	Mail. Send mail to a specified user.
`n`	Next message. Advances pointer to next message, clears the screen, and displays the message (see also **Return**).
`o`	Options. Used to alter the settings of a number of option values (see CUSTOMIZATION).
`p`	Print. Used to print out the current message or the tagged messages to a previously defined printing method *print* (see CUSTOMIZATION).
`q`	Quit. Gracefully terminate `elm` and perform message cleanup according to defined personal preference. You can choose to keep undeleted mail in the incoming mailbox or move

it to an `mbox` file specified by `maildir` in file **`$HOME/.elm/elmrc`**. You can also decide whether or not to actually delete messages previously marked for deletion (see CUSTOMIZATION).

Ctrl-Q or **DEL**
: Exit (abort). Same as **`x`**. Aborts `elm` immediately without any changing the *mailbox*.

`r`
: Reply. Reply to the author of the current message. If `autocopy` is not specified (see CUSTOMIZATION), you can specify whether or not a copy of the source message is to be copied into the edit buffer. If copied in, all lines from the message are preceded by the prefix character sequence defined as `prefix` (see CUSTOMIZATION).

`s`
: Save to file. This command copies the current message or set of tagged messages into a user-specified file or folder. After saving a file, each message is marked for deletion and, if saving just one message, the current message pointer is incremented.

`t`
: Tag. Tag the current message for operation `|`, `p`, or `s`. Use of this command on a tagged message removes the tag.

Ctrl-T
: Tag all messages containing the specified pattern. Since tagging messages can occur on screens other than the one being viewed, `elm` first checks to see if any messages are currently *tagged* and asks if the tags should be removed. After that, it , similar to **Ctrl-D**, sets criteria (see **Ctrl-D**).

`x`
: Exit. This is the quickest way to exit `elm` without changing the *mailbox*.

Tilde Escapes

Tilde escape sequences are used to alter current or previously defined `elm` behavior:

`~?`
: Print a brief help menu.

`~b`
: Change the Blind-Carbon-Copy list.

`~c`
: Change the Carbon-Copy list.

`~e`
: Invoke the editor specified by the `$EDITOR` environment variable on the message, if possible.

`~f` *options*
: Add the specified list of messages or the current message. This uses `readmail` which means that all `readmail` options are available (see *readmail*(1)).

`~h`
: Change all the available headers (`To`, `Cc`, `Bcc`, and `Subject`)

`~m` *options*
: Same as `~f`, but with the current *prefix*.

`~o`
: Invoke a user specified editor on the message.

`~p`
: print out the message as typed in so far.

`~r` *filename*
: Include (read in) the contents of the specified file.

`~s`
: Change the Subject line.

`~t`
: Change the To list.

`~v`
: Invoke `$VISUAL` in your environment on the message if possible.

`~<` *command*
: Execute the specified shell command, entering the output of the command into the editor buffer upon completion (for example `~< who` includes the output of the `who` command in your message)

`~!` *command*
: Execute a shell command if one is given (as in `~!ls`) or give you a shell (either your shell setting as a `shell` (see CUSTOMIZATION) or `$SHELL` in your environment).

`~~`
: Add a line prefixed by a single `~` character.

Alias Commands

The following alias commands are used to manipulate user-defined mail aliases:

`a`
: Alias current message. This allows you to create an alias that has the return address of the current message as the address field of the alias. It prompts for a unique alias name.

`d`	Delete user alias. This prompts for alias name to be deleted. The alias is deleted from your alias_text file (**`$HOME/.elm/aliases.text`**).
`e`	Check personal alias with full expanded addresses. This is the same as **`p`**, except that this key fully expands aliases in the list of addresses.
`m`	Make user alias. **`elm`** prompts for a unique alias name, then for an address. The information provided is added to your individual alias_text file (**`$HOME/.elm/aliases.text`**), then added to the database.
`p`	Check personal alias. This is a simple way of checking what is in the alias database. It prompts for an alias name, and returns the address or the list of addresses associated with that name or the error message **`alias not found`** as appropriate.
`s`	Check system aliases. This is for checking what aliases are currently installed as system aliases. This command lists all current system aliases.
`u`	List user aliases. This is for checking what aliases are currently installed as user aliases. This command lists all current user aliases.
`r`	Return. Return to the main level of **`elm`** program.

CUSTOMIZATION

Like many HP-UX programs, **`elm`** supports automatic configuration by means of an **`rc`** file. The file must be named **`$HOME/.elm/elmrc`** and can contain any combination of the string, numeric, and boolean variables described below. If directory **`$HOME/.elm`** does not exist, **`elm`** asks whether you want to create it or not. If the answer is **`yes`**, **`elm`** creates **`$HOME/.elm`** automatically.

String Variables

`alternatives`

This string lists other machine and user name combinations that you receive forwarded mail from. **`elm`** uses this information when a *group reply* is being processed to ensure that a reply message is not sent to a user and/or machine address that would simply forward the reply message back to the originator. No default.

`calendar`	Name of calendar file. This is used in conjunction with the **`<`** command which scans messages for calendar entries. Default is **`$HOME/calendar`**.
`editor`	Specifies which editor to use when creating new mail. Choices also include **`none`** or **`builtin`** for the built-in editor. The built-in editor is available for all mail that does not already have text in the buffer (in replying, mailing with a **`signature`**, etc). Default is the editor defined by the current environment variable **`$EDITOR`**, or **`vi`** if **`$EDITOR`** is not defined.
`escape`	Escape character used in *built-in* editor. Default is tilde (~).
`fullname`	This is the name the mailer will use when sending mail from you. Default is the "gecos" field from the **`/etc/passwd`** file.
`mailbox`	This is where to put incoming mail after you've read it. When you answer **`no`** (**`n`**) to the **`keep messages in incoming mailbox?`** prompt, this is where the messages go. Default is **`$HOME/mbox`**.
`maildir`	This is the default mail directory, and is used to expand filenames in **`elm`** when specified using the = metacharacter. For example, if you save to file **`=/archive`**, the = is expanded to the current value of **`maildir`**. Default is **`$HOME/Mail`**. If the directory specified by **`maildir`** does not exist, **`elm`** asks whether you want to create it or not. If the answer is **`yes`**, **`elm`** creates the specified directory automatically, setting access permissions mode to 700.
`pager`	This defines the program to be used to display each message. This can be changed while within the **`elm`** program by selecting the appropriate entry in the `Option` Menu. Default is **`builtin`**.
`prefix`	Value of prefix for included line. When you *reply* to a message or *forward* a message to another person, you can optionally include the original message. This prefix indicates the included line. Default is **>** followed by a space character.

`print`	The command to run when p)rint`commandisexecuted.` This indicates how to print out a message. There are two possible formats for this string, either a command that can have a filename affixed to it (as a suffix) then sent to the system for execution, or a string that contains the meta-sequence **`%s`** which will be replaced by the name of the message file and also sent to the shell. Default is **`pr %s	lp`**.
`savemail`	This is where outgoing mail will have a copy silently saved. This will only be used if the **`copy`** flag is turned on. Also note that if the **`savename`** feature is enabled, this filename may be ignored since the program first looks for a mailbox that has the same name as the login of the person you are sending to, using that instead if found. Default is **`$HOME/mbox`**.
`shell`	This defines the shell to use when doing **!** escapes and such. Default is **`$SHELL`** in your current environment.
`signature`	This file, if defined, will be automatically appended to all outbound mail before the editor is invoked. Furthermore, if you'd like a different "signature" file for *local* mail and *remote* mail (remote being via other hosts), you can alternatively define two variables, **`localsignature`** and **`remotesignature`**, to have the same functionality. No default.
`sortby`	When reading mailboxes, either incoming or specified, you can have them sorted by any number of different ways. This can be changed without leaving **`elm`** by changing the S)orting**`criteria`**field in o)ption mode, but it can also be predefined to any of **`from`**, **`sent`**, **`received`**, **`subject`**, **`lines`**, or *status*. Each of these fields can also optionally be prefixed with the sequence **`reverse-`** to reverse the order of the sort. Default is **`received`**.
`weedout`	When specifying this option, you can then list headers that you don't want to see when you are reading mail. This is effective with **weed** is ON. This list can continue for as many lines as desired, as long as the continued lines all have leading indentation. Default is **`>From`**, **`In-Reply-To:`**, **`References:`**, **`Newsgroups:`**, **`Received:`**, **`Apparently-To:`**, **`Message-Id:`**, **`Content-Type:`**, **`From`**, and **`Mailer:`**.

Numeric Variables

`timeout`	This is the interval, in seconds, between resynchronizing. **`elm`** internally resynchronizes every *timeout* seconds. Default is 600 seconds (10 minutes).
`userlevel`	**`elm`** uses this value to determine the relative level of user's sophistication. Acceptable values are **`0`** for new users (default), **`1`** for moderately experienced **`elm`** users, and **`2`** for experts.

Boolean Variables

`alwaysdelete`	When set, this changes the default answer of the prompt **`Delete messages?`** to the indicated value. Default is ON for YES.
`alwaysleave`	This changes the default answer on the **`keep mail in incoming mailbox?`** prompt to the value indicated. Default is ON for YES.
`arrow`	This is identical in function to the **`-a`** command line option. Default is OFF.
`ask`	This is used to tell **`elm`** that you would rather not be asked **`Delete message?`** and such each time you leave the program, and instead **`elm`** should just use the values of **`alwaysdelete`** and **`alwaysleave`** without prompting. Default is ON.
`askbcc`	If turned on, the prompt **`Blind-Copies-To:`** appears for each message. If **`askbcc`** is OFF, you can add a "bcc" list by **`~b`** in the built-in editor or by using the header editor. Default is OFF.
`askcc`	If turned off, this allows you to send mail without being presented the **`Copies-To:`** prompt for each message. This still allows you to explicitly include addresses in the "cc" list via either **`~c`** in the built-in editor, or via using the screen-oriented header editor. Default is ON.

`autocopy` This is a boolean flag, and if set automatically copies the text of the message you are replying to into the edit buffer. Default is OFF.

`copy` This, in combination with the `savemail` option, allows you to have silent copies of all outgoing mail made on the outbound step. Default is OFF.

`expand` If this flag is on, tabs in your message written are expanded to spaces. This ensures that your message is displayed in its original layout when displayed on a terminal screen having different tab settings. This flag can be changed without leaving `elm` by changing the `T)abs-to-spaces` field in `o)ption` mode. Default is OFF.

`forms` This allows you to mail forms. Default is OFF.

`keep` By default, the mail system deletes mailboxes when you have removed everything from them. With this option ON, it instead preserves them as zero-length files. Default is OFF.

`keypad` If on, this tells `elm` that you have an HP terminal and enables the **Next**, **Prev**, **Home**, and **Shift-Home** keys. Default is ON.

`menus` If turned off, this inhibits the menu display on all `elm` program screen displays. Default is ON.

`movepage` If enabled, commands that move through the mailbox by pages (the `+` and `-` keys) also move the current message pointer to the top of that page of messages. If turned off, moving through the pages does not alter the current message pointer location. Default is OFF.

`names` Show only the user names when expanding aliases, rather than the name and electronic mail address on the `To:` field when sending mail. Default is OFF.

`noheader` This tells the mailer not to include the headers of messages when copying a message into a file buffer for replying to or forwarding. Default is ON.

`pointnew` If this is turned on, the mailer is automatically pointing to the first new message in your mailbox when started, instead of at message #1. This is only used for the incoming mailbox since other mailboxes are assumed not to have 'new' and 'old' mail. Default is ON.

`resolve` If this option is enabled, as soon as mail is 'dealt with', `elm` moves to the next message in the mailbox, after deletion, undeletion, saving a message, or forwarding a message. Default is ON.

`savename` When the user `save`s the messages, `elm` constructs the filename from the *login name* of the person who sent the message rather than *savemail* value. Similarly, when sending mail out, instead of just blindly saving it to the *savemail* file, `elm` first tries to save it to a file based on the *login name* of the person who is to receive the mail. If the needed outbound mail file does not already exist, the message is saved in the *savemail* file. Default is ON.

`skipdeleted`
If this flag is on, current message pointer skips the message with deleted flag `D` when the `j` or `k` command is used. If `J` or `K` is used, flagged messages are *not* skipped. Default is OFF.

`softkeys` If on, this tells `elm` to recognize HP 2622 terminal function-key protocol when interacting with your terminal. Default is ON.

`titles` Used with the flag **weed**, this flag allows you to have the first line of a message titled with:

`Message` *N/M* `from` *username date* `at` *time*

where all the information has been previously extracted from the message. Default is ON.

`warnings`
`elm` normally warns you when you send mail to a machine that cannot be directly accessed. Setting this

flag allows you to disable such warning messages. Default is ON.

`weed`
This is a boolean flag that, in combination with the `weedout` list, allows you to custom define the set of headers you would like to not have displayed while reading messages. Default is ON.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

When set, the `TMPDIR` environment variable specifies a directory to be used for temporary files, overriding the default directory `/tmp`.

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

To send a message without loading the main `elm` mail-processing program, use the simple command form consisting of the name of the program followed by the recipient's login name and optional address. `elm` prompts for Subject, Copies, and Blind-Copies, then starts an editor so you can compose the message (user responses are in boldface type):

> **$ elm j_doe**
>
> To: doe (John Doe)
>
> Subject: **this is a test**
>
> Copies To: [**Return**]
>
> Blind-Copies To: [**Return**]
>
> *...invokes editor, message composed, then...*
>
> Your options now are:
>
> S)end the message, E)dit it again, change/add H)eaders or F)orget it
>
> What is your choice? **s**
>
> mail sent!

To send files or output from commands by use of command-line redirection, use command resembling:

```
$ elm j_doe < help.c
```

Which reads file `help.c` and transmits it to the specified recipient.

To add a subject line to the message, use `-s` *subject* in the command line:

```
$ elm -s "File help.c transmission" j_doe < help.c
```

WARNINGS

Using two separate mail programs to access the same mail file simultaneously (usually inadvertently from two separate windows) can cause unpredictable results.

AUTHOR

`elm` was developed by Hewlett-Packard Company.

FILES

`/usr/mail`	directory for incoming mail (mode **755**, group ID `mail`)
`/usr/mail/`*username*`.lock`	lock for mail directory
`/usr/mail/`*username*	incoming mailbox for *user*. (mode **660**, group ID `mail`)
`/usr/lib/nls/C/elm.cat`	location of the message catalogue
`$HOME/.elm`	directory for `elm`
`$HOME/.elm/elmrc`	personal customized file
`$HOME/.elm/elmheaders`	contents of additional headers
`/usr/lib/elm/elm_help.0`	help file for main screen

`/usr/lib/elm/elm_help.1`	help file for alias screen
`/usr/lib/elm/elm_help.2`	help file for option screen
`/usr/lib/elm/elmrc-info`	comment file for `.elm/elmrc` file
`/usr/mail/.elm`	directory for `elm` mailer system aliases
`/usr/mail/.elm/aliases.hash`	system alias hash table
`/usr/mail/.elm/aliases.data`	system alias data table
`/usr/mail/.elm/aliases.text`	system alias text file
`$HOME/.elm/aliases.hash`	user alias hash table
`$HOME/.elm/aliases.data`	user alias data table
`$HOME/.elm/aliases.text`	user alias text file
`/tmp/snd.`*pid* and `sndh.`*pid*	outgoing mail edit buffer
`/tmp/form.`*pid*	editor buffer for form message
`/tmp/print.`*pid*	temporary file for printing message
`/tmp/alias.`*pid*	temporary file for deleting alias
`$HOME/.elm/readmail`	used by `readmail`
`/tmp/mbox.`*logname*	temporary mbox for user *logname*
`$HOME/Cancelled.mail`	cancelled message of non-interactive use.

SEE ALSO

elmalias(1), mailfrom(1), newmail(1), readmail(1), vi(1).

NAME

elmalias - create and verify elm user and system aliases

SYNOPSIS

`elmalias` [`-q`]
`elmalias` *alias-list*
`elmalias -l` [*regular_expression*]

DESCRIPTION

`elmalias` is used for setting, checking and listing aliases. If no options are specified, `elmalias` attempts to install a new set of personal aliases.

`elmalias` recognizes the following options:

`-c` *alias_list* Search for the alias or aliases specified by *alias_list*; User file is searched first, then the system alias file. If a match is found, the value of the alias is printed out; otherwise an error message is generated.

`-l` *regular_expression*
List all aliases available to the requesting user in the user and system alias files. Aliases are sorted alphabetically and listed, one line per alias, in a format similar to the following:

alias *address* (*fullname*)

If an optional *regular_expression* is used, only aliases matching the specified expression are listed. Otherwise, all are listed. *regular_expression* follows Extended Regular Expression syntax (see *regexp*(5)).

`-q`
Install the system-wide alias set for all elm users on the machine. This option is restricted to users with appropriate privileges. A complete list of system-wide aliases to be installed must exist in file `/usr/mail/.elm/aliases.text`. `elmalias` then creates `aliases.hash` and `aliases.data`, to complete installation of the new system-wide alias list.

none specified
Install local user aliases. A complete list of user aliases to be installed must exist in file `$HOME/.elm/aliases.text` Installation is complete when `elmalias` has created the data files `$HOME/.elm/aliases.hash` and `$HOME/.elm/aliases.data`.

Alias Text Files

The format of file `aliases.text` is:

alias [, *alias* , ...] = [*fullname* =] *address* [, *address* , ...]

Addresses can be explicit routing or addressing information for a specific mailbox (such as `user@host.domain`) or another alias. If more than one alias is listed as the first part of the line, all will have the same value. If more than one address is listed, the specific alias is assumed to be a group rather than individual alias. An alias definition can be continued across multiple lines of the `aliases.text` file by starting the second and successive lines of the definition with either a space or tab character. (See EXAMPLES)

EXAMPLES

Assume file `aliases.text` contains the following:

```
# sample alias file
mom                     = my_mother@a.computer
dad,father,pop = Father = host!otherhost!dad
parents                 = mom dad
siblings                = brother1
        brother2
        sister
```

Listing this file produces:

```
dad     host!otherhost!dad (Father)
father  host!otherhost!dad (Father)
mom     my_mother@a.computer
```

```
parents !mom,dad
pop     host!otherhost!dad (Father)
siblings!brother1,brother2,sister
```

WARNINGS

The user alias file is always searched before the system alias file. Thus a user can override a system alias by having an alias of the same name defined in the user file. To prevent unexpected results, this factor should be considered when creating system and user alias files.

AUTHOR

`elmalias` was developed by HP.

FILES

`$HOME/.elm/aliases.text`	alias source for user
`$HOME/.elm/aliases.hash`	alias hash table for user
`$HOME/.elm/aliases.data`	alias data file for user
`/usr/mail/.elm/aliases.text`	alias source for system
`/usr/mail/.elm/aliases.hash`	alias hash table for system
`/usr/mail/.elm/aliases.data`	alias data file for system

SEE ALSO

elm(1), regexp(5).

NAME
enable, disable - enable/disable LP printers

SYNOPSIS
`enable` *printers*
`disable` [`-c`] [`-r` [*reason*]] *printers*

DESCRIPTION
`enable` activates the named *printers*, enabling them to print requests taken by `lp`. Use `lpstat` to find the status of printers (see *lp*(1) and *lpstat*(1)).

`disable` deactivates the named *printers*, disabling them from printing requests taken by `lp`. By default, any requests that are currently printing on the designated printers are reprinted in their entirety either on the same printer or on another member of the same class. Use `lpstat` to find the status of printers. Options useful with `disable` are:

`-c`	Cancel any requests that are currently printing on any of the designated printers.
`-r`[*reason*]	Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next `-r` option. If the `-r` option is not present or the `-r` option is given without a reason, a default *reason* is used. *reason* is reported by `lpstat`. The maximum length of the *reason* message is 80 bytes.

HP Clustered Environment
In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXTERNAL INFLUENCES
Environment Variables
LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, `enable` and `disable` behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support
Single- and multi-byte character code sets are supported.

EXAMPLES
Enable printer `snowwhite` to accept requests:

```
enable snowwhite
```

Deactivate printer `snowwhite` and cancel any logged jobs:

```
disable -c snowwhite
```

WARNINGS
If the restrict cancel feature (selected by the `lpadmin -orc` option — see *lpadmin*(1M)) is enabled, `disable` ignores the `-c` option.

`enable` and `disable` perform their operation on the local system (or HP cluster) only.

FILES
`/usr/spool/lp/*`

SEE ALSO
accept(1M), lp(1), lpadmin(1M), lpsched(1M), lpstat(1), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M).

NAME

env - set environment for command execution

SYNOPSIS

env [-] [-i] [*name* = *value*] ... [*command* [*arguments* ...]]

DESCRIPTION

env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name*=*value* are merged into the inherited environment before the command is executed. The -i option causes the inherited environment to be ignored completely so that the command is executed with exactly the environment specified by the arguments. The - option is obsolete and has the same effect as the -i option.

If no command is specified, the resulting environment is printed, one name-value pair per line.

RETURN VALUE

If *command* is invoked, the exit status of **env** is the exit status of *command*; otherwise, **env** exits with one of the following values:

0	**env** completed successfully.
1-125	**env** encountered an error.
126	*command* was found but could not be invoked.
127	*command* could not be found.

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

WARNING

The - option is obsolete. Use -i instead.

SEE ALSO

sh(1), exec(2), profile(4), environ(5).

STANDARDS CONFORMANCE

env: SVID2, XPG2, XPG3, POSIX.2

NAME

ex, edit - extended line-oriented text editor

SYNOPSIS

ex [**-**] [**-v**] [**-r**] [**-R**] [**-+***command*] [**-l**] [**-t***tag*] [**-V**] [**-w***size*] [**-x**] [*file* ...]
edit [**-**] [**-v**] [**-r**] [**-R**] [**-+***command*] [**-l**] [**-t***tag*] [**-V**] [**-w***size*] [**-x**] [*file* ...]

Remarks:

ex, **edit**, **vedit**, **vi**, and **view** are separate personalities of the same program. This manual entry describes the behavior of the **ex** personality. The **edit** personality is very similar to **ex** except that some of the editor configuration options have different defaults. On many HP-UX and other similar systems, the **e** and **ex** commands are synonymous.

DESCRIPTION

ex is a line-oriented text editor that supports both command and display editing (see *vi*(1)).

edit is identical to **ex**, except that the following configuration option defaults are altered to make the editor somewhat friendlier for beginning and casual users: **report=1**, **novice**, **showmode**, and **report=1** (see Edit Options later in this entry).

The following command-line options are recognized:

-	Suppress all interactive-user feedback. This is useful when editor commands are taken from scripts.
-v	Invoke *vi*.
-r	Recover *file* or files after an editor or system crash. If no *file* is specified a list of all saved files is printed.
-R	Set "read-only" mode to prevent overwriting a file inadvertently.
+*command*	Begin editing by executing the specified editor search or positioning *command*.
-l	*Lisp* mode; indents appropriately for Lisp program source code; the **( ) { } [[** and **]]** commands in **vi** are modified to be meaningful for Lisp.
-t *tag*	Edit the file containing *tag*, and position the editor at its definition (see the **tag** command below and *ctags*(1)).
-V	Verbose mode; editor commands are displayed as they are executed when input from a user's **.exrc** file or a source file (see the **source** command below).
-w *size*	Set the value of the **window** editor option to *size*.
-x	Encryption mode; the user is prompted for a key to initiate creation or editing of an encrypted file (see the **crypt** command below).

file specifies the file or files to be edited. If more than one *file* is specified, they are processed in the order given.

The name of the file being edited by **ex** is called the **current file**. Text from the current file is read into a buffer, and all editing changes are performed on this buffer. Changes do not affect the original file until the buffer is explicitly written back to the file.

The **alternate file** is the name of the last file mentioned in an editor command, or the previous current file name if the last file mentioned becomes the current file. If the **%** character is used as a file name, it is replaced by the current file name. Similarly, the **#** character is replaced by the alternate file name.

Twenty-six buffers named ASCII **a** through **z** can be used for saving blocks of text during the edit. If the buffer name is specified in uppercase, text is appended to the existing buffer contents rather than overwriting it.

The read-only mode can be cleared from within the editor by setting the **noreadonly** edit option (see Edit Options below). Writing to a different file is allowed in read-only mode. Also, a write can be forced to a read-only file by using **!** after the write command (see the **write** command below).

If an interrupt signal is received, and commands are being supplied from a keyboard, **ex** returns to the command level. If editor commands are coming from a file, an interrupt signal causes **ex** to abort.

If the system crashes or **ex** aborts due to an internal error or unexpected signal, **ex** attempts to preserve the buffer if any unwritten changes were made. Use the **-r** command-line option to retrieve the saved changes.

ex starts up in command mode, as indicated by the colon (**:**) prompt. **ex** switches to input mode whenever an **append**, **insert**, or **change** command is encountered. To terminate input mode and return to command mode, type a period (**.**) alone at the beginning of a line.

Command lines beginning with the double quote character (") are ignored (this is useful for placing comments in an editor script).

Multiple commands can be combined on a single line by separating them with a vertical bar character (|). However, global commands, comments, and the shell escape command must be the last command on a line because they cannot be terminated by a | character.

Addressing

ex recognizes the following line address forms:

.	Dot (**.**) refers to the current line. There is always a current line whose position might be the result of an explicit movement by the user or the result of a command that affected multiple lines (in which case it is usually the last line affected).
n	The *n*th line in the buffer. Lines are numbered sequentially, starting at line 1.
$	The last line in the buffer.
%	Abbreviation for **1,$**, meaning the entire buffer.
+n *-n*	An offset relative to the current line. (For example, the forms **.+3**, **+3**, and **+++** are equivalent.)
/re/ *?re?*	Line containing the pattern *re*, scanning forward (*/re/*) or backward (*?re?*). The trailing / or ? can be omitted if the line is only being printed. If *re* is omitted, **ex** uses the more recently set of either the substitution string or scanning string (see Regular Expressions below).
'x	Lines can be marked using single ASCII lowercase letters (see the **mark** command below); *'x* refers to line marked *x*. In addition, the previous current line is marked before each non-relative motion. This line can be referred to by using ' for *x* (thus '' refers to the previous current line).

Addresses for commands consist of a series of line addresses (specified as above), separated by comma (**,**) or semicolon (**;**). Such address lists are evaluated left-to-right. When the separator is a semicolon, the current line is set to the value of the previous address before the next address is interpreted. If more addresses are given than the command requires, then all but the last one or two are ignored. Where a command requires two addresses, the first line addressed must precede the second one in the buffer. A null (missing) address in a list defaults to the current line.

Command Names and Abbreviations

The command names enclosed within parenthesis in the table below are available only in their abbreviated forms.

Command	Abbr.	Command	Abbr.	Command	Abbr.
abbreviate	`ab`	open	`o`	unabbreviate	`una`
append	`a`	next	`n`	undo	`u`
args	`ar`	number	`# nu`	unmap	`unm`
change	`c`	preserve	`pre`	version	`ve`
chdir	`chd cd`	print	`p`	visual	`vi`
copy	`co t`	put	`pu`	write	`w`
crypt	`cr X`	quit	`q`	xit	`x`
delete	`d`	read	`r`	yank	`ya`
edit	`e ex`	recover	`rec`	(window)	`z`
file	`f`	rewind	`rew`	(escape)	`!`
global	`g v`	set	`se`	(lshift)	`<`
insert	`i`	shell	`sh`	(rshift)	`>`
join	`j`	source	`so`	(scroll)	`^D`
list	`l`	stop	`st ^Z`	(line number)	`=`
map		substitute	`s & ~`	(execute buffer)	`* @`
mark	`k ma`	suspend	`su ^Z`		
move	`m`	tag	`ta`		

Command Descriptions

In the following, *line* is a single-line address, given in any of the forms described in the Addressing section above. *range* is a pair of line addresses separated by a comma or semicolon (Addressing explains the difference between the two). *count* is a positive integer specifying the number of lines to be affected by the command. *flags* is one or more of the characters `#`, `p`, and `l`. The corresponding command to print the line is executed after the command completes. Any number of `+` or `-` characters can also be given with these flags.

When *count* is used, *range* is ineffective. Instead, only a line number should be specified to indicate the first line affected by the command. (If a range is given, the last line of the range is interpreted as the starting line for the command.)

These modifiers are all optional. Defaults are as follows, unless otherwise stated: the default for *line* is the current line; the default for *range* is the current line only (`.,.`); the default for *count* is 1; the default for *flags* is null.

When only a *line* or a *range* is specified (with a null command), the implied command is `print`; if a null line is entered, the next line is printed (equivalent to `.+1p`)

`abbreviate` `ab[breviate]` *word rhs*
: Add the named abbreviation to the current list. In visual mode, if *word* is typed as a complete word during input, it is replaced by the string *rhs* (right-hand string).

`append` *line* `a[ppend][!]`
: Enter input mode; the input text is placed after the specified line. If line 0 is specified, the text is placed at the beginning of the buffer. The last input line becomes the current line, or the target line if no lines are input.

  Appending `!` to the command toggles the `autoindent` edit option setting for this insert only.

`args` `ar[gs]`
: Prints the argument, placing the current argument between `[` and `]`.

`change` *range* `c[hange][!]` *count*
: Enter input mode; the input text replaces the specified lines. The last input line becomes the current line; if no lines are input, the effect is the same as a delete.

  Appending `!` to the command toggles the `autoindent` edit option setting for this insert only.

`chdir`
: `chd[ir][!]` [*directory*]
`cd[!]` [*directory*]
Change the working directory to *directory*. If *directory* is omitted, the value of the **HOME** environment variable is used. If the buffer has been modified since the last write and the name of the file being edited does not begin with a slash (/), a warning is issued and the working directory is not changed. To force a change of directory in this case, append the character `!` to the command.

`copy`
: *range* `co[py]` *line flags*
range `t` *line flags*
A copy of the specified lines (*range*) is placed after the specified destination *line*; line 0 specifies that the lines are to be placed at the beginning of the buffer. (The letter `t` is an alternative abbreviation for the `copy` command.)

`crypt`
: `cr[ypt]`
X
The user is prompted for a key with which to enter encryption mode. This command can also be used to change the key entered from a previous `crypt` command or the **-x** command line option. If no key is supplied in response to the prompt (that is, only **Return** is pressed), encryption mode is canceled and the buffer is written out in plain-text form by subsequent write commands.

While in encryption mode, all file input is decrypted using the current key. However, while an input file is being processed, if a block of text (approximately 1024 bytes) is encountered that contains only 7-bit ASCII characters, that block of text is assumed to be plain-text and is not decrypted. All file output, except that piped via a `!` shell escape to another command, is encrypted using the current key.

The temporary file used by the editor to manage the buffer is not encrypted until the current buffer is discarded (or written out) and editing begins on a new file. When creating a new file that requires encryption protection, ensure that the buffer file is also encrypted by specifying the **-x** option when invoking the editor.

`delete`
: *range* `d[elete]` *buffer count*
The specified lines are deleted from the buffer. If a named *buffer* is specified, the deleted text is saved in it. If no buffer is specified, the unnamed buffer is used (that is, the buffer where the most recently deleted or yanked text is placed by default). The new current line is the line after the deleted lines or the last line of the file if the deleted lines were at the end of the file.

`edit`
: `e[dit][!]` [+ *line*] *file*
`ex[!]` [+ *line*] *file*
Begin editing a new file (**ex** is an alternative name for the `edit` command). If the current buffer has been modified since the last write, a warning is printed and the command is aborted. This action can be overridden by appending the character `!` to the command (`e!` *file*). The current line is the last line of the buffer unless it is executed from within *vi*, in which case the current line is the first line of the buffer. If the *+line* option is specified, the current line is set to the specified position, where *line* can be a number (or `$`) or specified as `/`*re* or `?`*re*.

`file`
: `f[ile]`
Print the current file name and other information, including the number of lines and the current position.

`global`
: *range* `g[lobal][!]` `/`*re*`/` *command ...*
range `v` `/`*re*`/` *command ...*
Perform *command* on lines within *range* (or on the entire buffer if no *range* is given) that contain *re*. First mark the lines within the given *range* that match the pattern *re*. If the pattern is omitted, the more recently set of either the substitution string or the scanning string is used (see Regular Expressions below). Then the given *command* or *commands* are executed with `.` set to each marked line. Any character other than a letter or a digit can be used to delimit the pattern instead of the `/`.

command can be specified on multiple lines by hiding new-lines with a backslash. If *command* is omitted, each line is printed. `append`, `change`, and `insert` commands are allowed; the terminating dot can be omitted if it ends *command* or *commands*. The `visual` command is also permitted (unless the `global` command itself has been issued from visual mode), and takes input from the terminal. (If *command* contains a visual-mode command (that is, `open` or `visual`), the visual-mode command must be terminated by the visual-mode `Q` command in order to proceed to the next marked line.)

The `global` command itself and the `undo` command are not allowed in *command*. Edit options `autoprint`, `autoindent`, and `report` are inhibited.

Appending a `!` to the `global` command (that is, `g!` ...) or using the alternate name `v` causes *command* to be run on the lines within *range* that do not match the pattern.

`insert`
: *line* `i[nsert][!]`

Enter input mode; the input text is placed before the specified line. The last line input becomes the current line, or the line before the target line, if no lines are input.

Appending `!` to the command toggles the `autoindent` edit option setting for this insert only.

`join`
: *range* `j[oin][!]` *count flags*

Join together the text from the specified lines into one line. White space is adjusted to provide at least one blank character (two if a period appears at the end of a line, or none if the first character of a line is a closing parenthesis (`)`)). Extra white space at the beginning of a line is discarded.

Appending a `!` to the command causes a simpler join with no white-space processing.

`list`
: *range* `l[ist]` *ount flags*

Print the specified lines with tabs displayed as `^I` and the end of each line marked with a trailing `$`. (The only useful flag is `#` for line numbers.) The last line printed becomes the current line.

`map`
: `map[!]` [*x* | #*n*] *rhs*

The `map` command is used to define macros for use in visual mode. The first argument, *x*, can be a single character or a multi-character sequence. In the special sequence, #*n*, *n* is a digit referring to the function key *n*. When *x* or the function key corresponding to #*n* is typed in visual mode, `map` interprets the action as though *rhs* were typed. If `!` is appended to the command `map`, the mapping is effective during input mode rather than command mode. Special characters, white space, and new-line must be escaped with a `^V` to be entered in the arguments. The first argument cannot contain the colon (`:`) as the first character, nor can multicharacter sequences begin with an alphabetic character. See also the Edit Options `timeout`, `timeoutlen`, `keyboardedit`, and `keyboardedit!` below.

`mark`
: *line* `ma[rk]` *x*
line `k` *x*

(The letter `k` is an alternative abbreviation for the `mark` command.) The specified line is given the specified mark *x*, which must be a single ASCII lowercase letter (`a-z`) (*x* must be preceded by a space or tab). The current line position is not affected.

`move`
: *range* `m[ove]` *line*

Move the specified lines (*range*) to follow the target *line*. The first line moved becomes the current line.

`next`
: `n[ext][!]` [*file* ...]

The next file from the command line argument list is edited. Appending a `!` to the command overrides the warning about the buffer having been modified since the last write (and discards any changes unless the `autowrite` edit option is set). The argument list can be replaced by specifying a new one on this command line.

`number`
: *range* `nu[mber]` *count flags*
range `#` *count flags*

(The **#** character is an alternative abbreviation for the `number` command.) Print the lines, each preceded by its line number (the only useful flag is **1**). The last line printed becomes the current line.

`open`
: *line* o[`pen`] */re/ flags*
Enter open mode, which is similar to visual mode with a one-line window. All the visual-mode commands are available. If a match is found for the optional regular expression in *line*, the cursor is placed at the start of the matching pattern. The visual mode command Q exits open mode. For more information, see *vi*(1).

`preserve`
: `pre[serve]`

The current editor buffer is saved as if the system had just crashed. This command is used in emergencies, for example when a write does not work and the buffer cannot be saved in any other way.

`print`
: **range** p[`rint`] *count*
Print the specified lines, with non-printing characters printed as control characters in the form `^x`; DEL is represented as `^?`. The last line printed becomes the current line.

`put`
: *line* pu[`t`] *buffer*
Place deleted or "yanked" lines after *line*. A buffer can be specified; otherwise, the text in the unnamed buffer (that is, the buffer in which deleted or yanked text is placed by default) is restored.

`quit`
: q[`uit`][`!`]
Terminate the edit. If the buffer has been modified since the last write, a warning is printed and the command fails. To force termination without preserving changes, append `!` to the command.

`read`
: *line* r[`ead`] *file*
Place a copy of the specified *file* in the buffer after the target line (which can be line 0 to place text at the beginning). If no *file* is named, the current file is the default. If no current file exists, *file* becomes the current file. The last line read becomes the current line except in visual mode where the first line read becomes the current line.

If `file` is given as `!`*string, string* is interpreted as a system command and passed to the command interpreter; the resultant output is read into the buffer. A blank or tab must precede the `!`.

`recover`
: rec[`over`][`!`] *file*
Recover *file* from the save area, after an accidental hangup or a system crash. If the current buffer has been modified since the last write, a warning is printed and the command is aborted. This action can be overridden by appending the character `!` to the command (`rec!` *file*).

`rewind`
: rew[`ind`][`!`]
The argument list is rewound, and the first file in the list is edited. Any warnings can be overridden by appending a `!`.

`set`
: se[`t`][`?`][`all`] *parameter*
With no arguments, the `set` command prints those editor configuration options whose values have been changed from the default settings; if `all` is specified, it prints all current configuration option values.

Giving an option name followed by a `?` causes the current value of that option to be printed. The `?` is necessary only for Boolean-valued options. Boolean options are given values by the form `se` *option* to turn them on, or `se` no*option* to turn them off; string and numeric options are assigned by the form `se` *option=value*. More than one parameter can be given; interpretation is left-to-right.

See Edit Options below for further details about options.

`shell`
: sh[`ell`]
The user is put into the command interpreter specified by the `shell` edit option (see below). Editing is resumed on exit from the command interpreter.

`source` `so[urce]` *file*
Read and execute commands from the specified *file*. `so` commands can be nested.

`substitute` *range* `s[ubstitute]` */re/repl/ options count flags*
range `s` *options count flags*
range `&` *options count flags*
range `sr` *options count flags*
range `~` *options count flags*
range `s\?`*repl*
range `s\&`*repl*
On each specified line, the first instance of the pattern *re* is replaced by the string *repl*. (See Regular Expressions and Replacement Strings below.) Any character other than a letter or a digit can be used to delimit the pattern instead of the `/`. If *options* includes the letter `g` (global), all instances of the pattern in the line are substituted. If the option `c` (confirm) is included, the user is queried about whether to perform each individual substitution, as follows: Before each substitution the line is typed with the pattern to be replaced marked with carat characters (`^`); a response of `y` causes the substitution to be performed, while any other input aborts it. The last line substituted becomes the current line.

If the substitution pattern *re* is omitted (`s//`*repl*`/`), the more recently set of either the substitution string or the scanning string is used (see Regular Expressions below).

If the alternative forms `s` or `&` of the command are used, the substitution pattern defaults to the previous substitution string and the replacement string defaults to the previous replacement string used.

If the alternative forms `sr` or `~` of the command are used, the substitution pattern defaults to the more recently set of either the substitution string or the scanning string and the replacement string defaults to the previous replacement string used.

The alternative form `s\?`*repl* is equivalent to `s/`*scan-re*`/`*repl*`/`, where *scan-re* is the previous scanning string.

The alternative form `s\&`*repl* is equivalent to `/`*subs-re*`/`*repl*`/`, where *subs-re* is the previous substitution string.

`suspend` `su[spend][!]`
`stop` `st[op][!]`
^Z
Suspend the editor job and return to the calling shell. `^Z` and the `stop` command are alternative names for the `suspend` command. `^Z` is the user process control suspend character which is typically the character control-Z (ASCII SUB) (see *stty*(1)). This command is disabled if the calling shell does not support job control or has disabled it.

The buffer is written to the current file before the editor is suspended if the `autowrite` editor option is set, the `readonly` editor option is not set, and the buffer has been modified since the last write. To override this action, append the `!` character to the `suspend` or `stop` command.

`tag` `ta[g][!]` *tag*
The files specified by the `tags` edit option (see below) are searched sequentially until a tag definition with the name of *tag* is found. If found, edit the file and position the editor at the address specified by the tag definition.

The buffer is written to the current file before the new file is edited if a file different from the current file is being edited, the `autowrite` editor option is set, the `readonly` editor option not set, and the buffer has been modified since the last write. To override this action, append the `!` character to the command.

`unabbreviate`
`una[bbreviate]` *word*
Delete *word* from the list of abbreviations (see the `abbreviate` command above).

`undo` `u[ndo]`
Reverse the changes made by the previous editing command. For this purpose, `global`

and **visual** are considered single commands. Commands that affect the external environment, such as **write**, **edit** and **next**, cannot be undone. An **undo** can itself be reversed.

unmap — **unm**[**ap**][**!**] *x*
The macro definition for *x* is removed (see the **map** command above).

version — **ve**[**rsion**]
Print the current version of the editor.

visual — *line* **vi**[**sual**] *type count flags*
Enter visual mode at the specified *line*. The *type* is optional, and can be one of the designated characters **+**, **-**, **.**, or **^**, as in the **z** command, to specify the position of the specified line on the screen window (default is to place the line at the top of the screen window). A *count* specifies an initial window size; the default is the value of the edit option **window**. The flags **#** and **l** (ell) cause lines in the visual window to be displayed in the corresponding mode (see **number** and **list** commands). The **Q** command exits visual mode. For more information, see *vi*(1).

write — *range* **w**[**rite**][**!**][**>>**] *file*
range **wq**[**!**][**>>**] *file*
Write the specified lines (or the entire buffer, if no *range* is given) out to *file*, printing the number of lines and characters written. If *file* is not specified, the default is the current file (the command fails with an error message if there is no current file and no file is specified).

If an alternate file is specified and the file exists, the write fails, but can be forced by appending **!** to the command. To append to an existing file, append **>>** to the command. If the file does not exist, an error is reported.

If the file is specified as **!***string*, *string* is interpreted as a system command; the command interpreter is invoked, and the specified lines are passed as standard input to the command.

The command **wq** is equivalent to a **w** followed by a **q**; **wq!** is equivalent to **w!** followed by **q**; and **wq>>** is equivalent to **w>>** followed by **q**.

xit — **x**[**it**][**!**][**>>**] *file"*
If changes have been made to the buffer, a **write** command is executed with any options (such as **!**, **>>**, or **file**) used by the **write** command. Then (in any case) the **quit** command is executed.

yank — *range* **ya**[**nk**] *buffer count*
Place the specified lines in the named *buffer*. If no buffer is specified, the unnamed buffer is used (that is, the buffer where the most recently deleted or yanked text is placed by default).

(window) — *line* **z** *type count flags*
The number of lines specified by *count* are displayed. The default for *count* is the value of the edit option **window**.

If *type* is omitted, *count* lines following the specified *line* (default current line) are printed.

If *type* is specified, it must be one of the following designated characters: **+**, **-**, **.**, **^**, or **=**. **+** displays a window of lines following the addressed line, **-** causes the addressed line to be placed at the bottom of the window of displayed lines, **.** causes the addressed line to be placed at the center of the window, **^** displays a window of lines that are two windows prior to the addressed line, and **=** displays the addressed line at the center of the window with a line of dashes above and below the addressed line.

The last line printed becomes the current line, except for the **=** type, in which case the addressed line becomes the current line.

(escape) — **!** *command*
The remainder of the line after the **!** is passed to the system command interpreter for execution. A warning is issued if the buffer has been changed since the last write. A single **!** is printed when the command completes. The current line position is not affected.

Within the text of *command*, `%` and `#` are expanded as filenames, and `!` is replaced with the text of the previous `!` command (thus `!!` repeats the previous `!` command). When such expansion is performed, the expanded line is echoed.

(escape) *range* `!` *command*
In this form of the `!` command, the specified lines (there is no default; see previous paragraph) are passed to the command interpreter as standard input; the resulting output replaces the specified lines.

(lshift) *range* < *count flags*
Shift the specified lines to the left; the number of spaces to be shifted is determined by the edit option `shiftwidth`. Only white space (blanks and tabs) is lost in shifting; other characters are not affected. The last line changed becomes the current line.

(rshift) *range* > *count flags*
Shift the specified lines to the right by inserting white space (see previous paragraph for further details).

(scroll) `^D`
Ctrl-D (ASCII EOT) prints the next *n* lines, where *n* is the value of the edit option `scroll`.

(line number) *line* = *flags*

Print the line number of the specified `line` (default last line). The current line position is not affected.

(exec buffer) `*` [*buffer*]
`@` [*buffer*]
Execute the contents of the named buffer as an editor command. There is no difference between the `*` and the `@` forms of this command. If a buffer is not specified or *buffer* is `*` or `@`, the buffer last named in the `exec` *buffer* command is executed.

Regular Expressions

The editor maintains copies of two regular expression strings at all times: the substitution string, and the scanning string. The substitute command sets the substitution string to the regular expression used. Both the global-command and the regular-expression form of line addressing (see Addressing above) for all commands set the scanning string to the regular expression used. These strings are used as default regular expressions as described under Addressing, the `global` command, and the `substitute` command.

The editor supports Basic Regular Expressions (see *regexp*(5)) with the following modifications:

`\<`	The `\<` matches the beginning of a "word"; that is, the matched string must begin in a letter, digit, or underline, and must be preceded by the beginning of the line or a character other than the above.
`\>`	The `\>` matches the end of a "word" (see previous paragraph).
`~`	Match the replacement part of the last `substitute` command.
[*string*]	The positional quoting within bracket expressions defined by Basic Regular Expressions is replaced by the use of the backslash (`\`) to quote bracket-expression special characters.
`\{...\}`	The constructs `\{`*m*`\}`, `\{`*m*`,\}`, and `\{`*m*`,`*n*`\}` are not supported.
`nomagic`	When the editor option `nomagic` is set, the only characters with special meanings are `^` at the beginning of a pattern, `$` at the end of a pattern, and `\`. The characters `.`, `*`, `[`, and `~` lose their special meanings unless escaped by a `\`.

Replacement Strings

The character `&` (`\&` if `nomagic` is set) in the replacement string stands for the text matched by the pattern to be replaced. The character `~` (`\~` if `nomagic` is set) is replaced by the replacement part of the previous `substitute` command. The sequence `\`*n*, where *n* is an integer, is replaced by the text matched by the pattern enclosed in the *n*th set of parentheses `\(` and `\)`. The sequence `\u` (`\l`) causes the immediately following character in the replacement to be converted to uppercase (lowercase), if the character is a letter. The sequence `\U` (`\L`) turns such conversion on, until the sequence `\E` or `\e` is encountered, or the end of the replacement string is reached.

Edit Options

The command `ex` has a number of options that modify its behavior. These options have default settings, which can be changed using the `set` command (see above). Options can also be set at startup by putting a `set` command string in the environment variable **EXINIT**, or in the file `.exrc` in the **HOME** directory, or in `.exrc` in the current directory. If **EXINIT** exists, the `.exrc` file in the **HOME** directory is not executed. If the current directory is not the **HOME** directory and the `exrc` editor option (see below) is set, the `.exrc` file in the current directory is executed after **EXINIT** or the **HOME** directory `.exrc`.

The editor obtains the horizontal and vertical size of the terminal screen from the `terminfo` database (see *terminfo*(4)). These values can be overridden by setting the **COLUMNS** and **LINES** environment variables. See the `window` option below for more information.

Options are Boolean unless otherwise specified.

`autoindent, ai`
: If autoindent is set, each line in insert mode is indented (using blanks and tabs) to align with the previous line. (Indentation begins after the line appended, or before the line inserted or the first line changed.) Additional indentation can be provided as usual; succeeding lines are automatically indented to the new alignment. Reducing the indent is achieved by typing `^D` one or more times; the cursor is moved back `shiftwidth` spaces for each `^D`. (A `^` followed by a `^D` removes all indentation temporarily for the current line; a `0` followed by a `^D` removes all indentation.)

`autoprint, ap`
: The current line is printed after each command that changes buffer text (autoprint is suppressed in `global` commands).

`autowrite, aw`
: The buffer is written to the current file if the buffer has been modified and a `next`, `rewind`, or `!` command is given.

`beautify, bf`
: Cause all control characters other than tab, new-line, and form-feed to be discarded from the input text.

`directory, dir`
: Specify the directory to which the editor buffer should be placed. This option only takes effect when a new buffer is created. It should be set in **EXINIT** or `.exrc` to affect the location of the buffer file for the edit file specified on the command line.

  If the specified directory is set from **EXINIT** or a `.exrc` file and is not writable by the user, the editor quits; if set interactively by the user, the editor issues an error message.

`doubleescape`
: When set, two consecutive ESC (escape) characters are required to leave input mode. In input mode, a single ESC character followed by a different character causes *vi*(1) to issue an audible or visual warning (see the `flash` edit option) and insert both characters into the buffer. This option is not set by default.

  The character sequences transmitted by keyboard editing keys of some terminals are identical to some sequences of *vi*(1) user commands. If the mapping of these keys is enabled (see the `keyboardedit` and `keyboardedit!` options), *vi*(1) might not be able to reliably distinguish between the character sequence transmitted by an editing key and the same character sequence typed by a user. This problem is most likely to occur when the user is typing ESC to terminate input mode immediately followed by another *vi*(1) command. By setting the `doubleescape` option, the ambiguity of this case is removed.

`edcompatible, ed`
: Cause the presence of `g` and `c` suffixes on substitute commands to be remembered, and toggled by repeating the suffixes.

`errorbells, eb`
: When set, error messages are preceded with a bell only on terminals that do not support a standout or highlighting mode such as inverse video. If the terminal supports highlighting, the bell is never used prior to error messages and this option has no effect. Note that visual-mode errors are signaled by the bell (regardless of the setting of this option) without an accompanying error message.

`exrc`
: When set, the `.exrc` file in the current directory is processed during editor initialization if the current directory is not the **HOME** directory. This option is not set by default and must be set in the **EXINIT** environment variable or the **HOME** directory

`.exrc` file to have any effect. See the Edit Options introductory text above.

`flash, fl` When set, the screen flashes instead of beeping, provided an appropriate `flash_screen` entry is present in the `/usr/lib/terminfo/*` database for the terminal being used.

`hardtabs, ht` Define the spacing between hardware tab settings and the number of spaces used by the system when expanding tab characters. Tab stops are placed in each column number (starting at the left edge of the screen) that corresponds to an integer multiple of the numeric value of this option.

`ignorecase, ic` All uppercase characters in the text are mapped to lowercase in regular expression matching. Also, all uppercase characters in regular expressions are mapped to lowercase, except in character class specifications.

`keyboardedit` When set, any keyboard editing key mappings that are loaded automatically at initialization for command-mode use are enabled. If not set, these mappings are disabled (but not deleted). Use the **map** command to get a list of the currently enabled command-mode mappings. This option is set by default.

`keyboardedit!` When set, the keyboard editing key mappings automatically loaded at initialization for insert-mode use are enabled. If not set, these mappings are disabled (but not deleted). Use the **map!** command to list the currently enabled insert mode mappings. This option is not set by default for terminals whose keyboard editing keys send HP-style escape sequences (an ESC followed by a single letter). This option is set by default for all other terminals.

`lisp` **Autoindent** mode, and the `( ) { } [[ ]]` commands in visual mode are suitably modified for `lisp` code.

`list` All printed lines are displayed with tabs shown as **^I**, and the end of line marked by a **$**.

`magic` Change the interpretation of characters in regular expressions and substitution replacement strings (see the relevant sections above).

`mesg` If set, other users can use the `write` command (see *write*(1)) to send messages to your terminal, possibly disrupting the screen display. Unsetting this option blocks write permission to your terminal from other system users while you are using the editor.

`modelines, ml` If set when the editor reads in a file, any **ex** commands embedded in the first five and last five lines of the file are executed after **.exrc** and **EXINIT** commands are processed but before editing control is given to the user. The **ex** commands must be prefixed by **ex:** or **vi:** and terminated by **:** in a single line. Any number of other characters with the exception of the colon (**:**) can precede or follow the embedded command.

`novice` Use the version of the editor available for novices, known as `edit` or `vedit`.

`number, nu` Cause lines to be printed with line numbers.

`optimize,opt` Suppress automatic carriage returns on terminals that do not support direct cursor addressing. This streamlines text output in certain situations such as when printing multiple lines that contain leading white space.

`paragraphs, para`
The value of this option is a string whose successive pairs of characters specify the names of text-processing macros that begin paragraphs. (A macro appears in the text in the form **.***xx*, where the **.** is the first character in the line.)

If any macros have a single-character name, use a space character to substitute for the missing second character in the name. When typing a space character in such situations, the space must be preceded by a backslash (\) to prevent the editor from interpreting it as a delimiter.

`prompt`	When set, command mode input is prompted for with a colon (`:`); when unset, no prompt is displayed.
`readonly, ro`	Set the read-only flag for the file being edited, thus preventing accidental overwriting at the end of the session. This option is equivalent to invoking `vi` or `ex` with the `-R` option or using the `view` command.
`redraw`	The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.)
`remap`	If set, macro translation allows for macros defined in terms of other macros; translation continues until the final product is obtained. If unset, a one-step translation only is done.
`report`	The value of this option gives the number of lines that must be changed by a command before a report is generated on the number of lines affected.
`scroll`	The value of this option determines the number of lines scrolled by a `^D` command and the number of lines displayed by the `z` command (twice the value of scroll).
`sections`	The value of this option is a string, in that successive pairs of characters specify the names of text-processing macros that begin sections. (See `paragraphs` option above.)
`shell, sh`	The value of this option specifies the file name of the user shell to be used for the `!` shell escape and `shell` commands. It is set by default to the value of the user's **SHELL** environment variable, if set, and otherwise to `/bin/sh`.
`shiftwidth, sw`	The value of this option gives the width of a software tab stop, and is used during `autoindent` and by the shift commands.
`showmatch, sm`	When a `)` or `}` is typed while in visual mode, the matching `(` or `{` is shown if it is still on the screen.
`showmode, smd`	When set, `showmode` displays the current editor mode (such as **INPUT MODE**) in the lower right-hand corner of the screen during the `visual` and `open` commands.
`slowopen, slow`	In visual mode, `slowopen` prevents screen updates during input to improve throughput on unintelligent terminals.
`tabstop, ts`	The value of this option specifies the software tab stops to be used by the editor to expand tabs in the input file.
`taglength, tl`	Specify the maximum number of characters in a tag that should be treated as significant. Characters beyond the limit are ignored. A value of zero (default value) means that all characters in the tag are significant.
`tags`	Specify which files should be used by the `tag` command and the `-t` option. The default is `tags /usr/lib/tags`. Each line of a tags file contains the following three fields separated by one or more space or tab characters: the tag name, the name of the file to be edited, and an address specification associated with the tag. A `tags` file must be sorted in order by tag name (see EXTERNAL INFLUENCES below). The `ctags` command (see *ctags*(1)) can be used to create **tags** files from C, Pascal and FORTRAN source files.
`term`	Define the type of terminal being used with the editor. The value is obtained from the **TERM** environment variable by default. There is no difference between the `term` and `ttytype` editor options. Setting either one results in both being changed.
`terse`	When set, error messages are shorter.
`timeout , to`	When set, all the characters of a multicharacter macro name (the first argument to the map command) must be received within the amount of time specified by the `timeoutlen` option to be accepted as a match for the macro and mapped to the defined string. If not set, no limit is placed on how long `vi` waits for the completion of a macro name. This option is set by default.

`timeoutlen`	The value of this option specifies in milliseconds (ms) the length of the macro timeout window (see the `timeout` option). The value must be at least 1 ms and the default is 500 ms. The value of this option has no effect unless the `timeout` option is enabled.
`ttytype, tty`	Define the type of terminal being used with the editor. The value is obtained from the **TERM** environment variable by default. There is no difference between the `term` and `ttytype` editor options. Setting either one results in both being changed.
`warn`	Warn if there has been `no write since last change` before a `!` or `shell` command escape.
`window, wi`	The number of lines in a text window in visual mode. The default value is set to one less than the size of your terminal screen (as defined by the `LINES` environment variable, if set, or the *terminfo*(4) data base otherwise). However, if the terminal baud rate (see *stty*(1) is set to less than 1200 or 2400, the default value is reduced to a maximum of 8 or 16 lines, respectively. The default value can be overridden by specifying a window size via the `-w` option when invoking the editor.
`w300`	If the terminal baud rate is less than 1200, set the `window` editor option to the value specified.
`w1200`	If the terminal baud rate is greater than or equal to 1200 but less than 2400, set the `window` editor option to the value specified.
`w9600`	If the terminal baud rate is greater than or equal to 2400, set the `window` editor option to the value specified.
`wrapscan, ws`	When set, editor searches using `/`*re*`/` (or `?`*re*`?`) resume from the beginning (or end) of the file upon reaching the end (or beginning) of the file (that is, the scan "wraps around"). When unset, editor searches stop at the beginning or the end of the file, as appropriate.
`wrapmargin, wm`	In visual mode, if the value of this option is greater than zero (that is, `wrapmargin=`*n*), a newline is automatically added to an input line at a word boundary, so that lines end at least *n* spaces from the right margin of the terminal screen.
`writeany, wa`	Inhibits the checks otherwise made before write commands, allowing a write to any file (provided the system allows it).

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions and in processing the `tags` file.

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters, the classification of characters as uppercase or lowercase letters, the shifting of the case of letters, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, the editor behaves as if all internationalization variables are set to "C". See *environ*(5).

When set, the TMPDIR environment variable specifies a directory to be used for temporary files, overriding the default directory `/tmp`.

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

The `undo` command causes all marks to be lost on lines that are changed and then restored.

The `z` command prints a number of logical rather than physical lines. More than a screenful of output might result if long lines are present.

Null characters are discarded in input files and cannot appear in resultant files.

On some systems, recovery of an edit with the `-r` option might be possible only if certain system-dependent actions were taken when the system was restarted.

On HP terminals, the attribute field of any function key specified via a `map #`*n* ... command should be set to `normal` rather than to the default of `transmit`.

For information about line-length limits, file size limits, etc., see WARNINGS section of *vi*(1).

AUTHOR

ex was developed by the University of California, Berkeley. The 16-bit extensions to **ex** are based in part on software of the Toshiba Corporation.

FILES

`/usr/lib/terminfo/*/*`	describes capabilities of terminals
`./.exrc`	editor initialization file
`$HOME/.exrc`	editor initialization file
`/usr/lib/exstrings`	error messages
`/usr/lib/exrecover`	recover command
`/usr/lib/expreserve`	preserve command
`/tmp/Ex2`*nnnnn*	editor temporary
`/tmp/Rx2`*nnnnn*	named buffer temporary
`/usr/preserve`	preservation directory

SEE ALSO

ctags(1), ed(1), expreserve(1), stty(1), vi(1), write(1), terminfo(4), environ(5), lang(5), regexp(5).

The Ultimate Guide to the vi and ex editors, Benjamin/Cummings Publishing Company, Inc., ISBN 0-8053-4460-8, HP part number 97005-90015.

STANDARDS CONFORMANCE

ex: SVID2, XPG2, XPG3

NAME

expand, unexpand - expand tabs to spaces, and vice versa

SYNOPSIS

expand [-t *tablist*] [*file* ...]

unexpand [-a] [-t *tablist*] [*file* ...]

Obsolescent:

expand [*-tabstop*] [*-tab1 , tab2 , ..., tabn*] [*file* ...]

DESCRIPTION

expand processes the named files or the standard input and writes to the standard output with tabs changed into spaces. Backspace characters are preserved in the output and decrement the column count for tab calculations. If a tab character is found after the last tab position, it is replaced by a single space. **expand** is useful for preprocessing character files that contain tabs (before sorting, looking at specific columns, etc).

expand recognizes the following command-line options and arguments:

[-t *tablist*] *tablist* specifies where to set the tab positions instead of the default 8. *tablist* can take two forms. If it is a single number, tabs are set *tablist* spaces apart. *tablist* can also be a blank- or comma-separated list of increasing positions where tabs are to be set.

[*-tabstop*] This option is obsolescent and is equivalent to using -t *tabstop*.

[*-tab1 , tab2 , ..., tabn*]
This option is obsolescent and is equivalent to using -t *tab1 ,tab2,... , tabn*.

unexpand processes the named files or the standard input and writes to the standard output with spaces changed into tabs where possible. By default, only leading spaces and tabs are converted to maximal strings of tabs. The default tab position is every 8 characters. Backspace characters are preserved into the output, and decrement the column count for tab calculations.

unexpand recognizes the following command-line options and arguments:

-a Tabs are inserted whenever they would compress the resultant file by replacing two or more spaces before a tab position.

-t *tablist* *tablist* specifies the tab positions. *tablist* can take two forms. If it is a single number, tabs are set every *tablist* spaces apart. If *tablist* is a blank- or comma-separated list of increasing positions, tabs are set at those locations. The -t option implies the -a option. If the -t option is not specified, the default is equivalent to specifying -t 8 except that -a is not implied for this case.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

expr - evaluate arguments as an expression

SYNOPSIS

`expr` *arguments*

DESCRIPTION

`expr` takes *arguments* as an expression, evaluates, then writes the result on the standard output. Terms in the expression must be separated by blanks. Characters special to the shell must be escaped. Note that `0`, rather than the null string, is returned to indicate a zero value. Strings containing blanks or other special characters should be quoted. Integer-valued arguments can be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by `\`. The list is in order of increasing precedence with equal-precedence operators grouped within `{ }` symbols.

expr `|` *expr*	returns the first *expr* if it is neither null nor `0`, otherwise returns the second *expr*.
expr `\&` *expr*	returns the first *expr* if neither *expr* is null or `0`, otherwise returns `0`.
expr `=` *expr* *expr* `\>` *expr* *expr* `\>=` *expr* *expr* `\<` *expr* *expr* `\<=` *expr* *expr* `!=` *expr*	returns the result of an integer comparison if both arguments are integers; otherwise returns the result of a lexical comparison (note that `=` and `==` are identical, in that both test for equality).
expr `+` *expr*	addition of integer-valued arguments.
expr `-` *expr*	subtraction of integer-valued arguments.
expr `\*` *expr*	multiplication of integer-valued arguments.
expr `/` *expr*	division of integer-valued arguments.
expr `%` *expr*	remainder of the integer-valued arguments.
expr `:` *expr*	The matching operator `:` compares the first argument with the second argument which must be a regular expression. *expr* supports the Basic Regular Expression syntax (see *regexp*(5)), except that all patterns are "anchored" (i.e., begin with `^`) and, therefore, `^` is not a special character, in that context. Normally, the matching operator returns the number of characters matched (`0` on failure). Alternatively, the `\( ... \)` pattern symbols can be used to return a portion of the first argument.
`length` *expr*	The length of *expr*.
`substr` *expr expr expr*	Takes the substring of the first *expr*, starting at the character specified by the second *expr* for the length given by the third *expr*.
`index` *expr expr*	Returns the position in the first *expr* which contains a character found in the second *expr*.
`match`	Match is a prefix operator equivalent to the infix operator `:`.
`\( ... \)`	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of **EXPR_NEST_MAX** as specified in the header file `<limits.h>`.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions and the behavior of the relational operators when comparing string values.

`LC_CTYPE` determines the interpretation of text as single- and/or multi-byte characters, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If **LC_COLLATE** or **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **expr** behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

As a side effect of expression evaluation, *expr* returns the following exit values:

0	Expression is neither null nor zero.
1	Expression is null or zero.
2	Invalid expression.
>2	An error occurred while evaluating the expression.

DIAGNOSTICS

`syntax error`	Operator or operand errors
`non-numeric argument`	Arithmetic attempted on a string

EXAMPLES

Add 1 to the shell variable **a**:

```
a='expr $a + 1'
```

For **$a** equal to either **/usr/abc/file** or just **file**, return the last segment of a path name (i.e., **file**). Beware of **/** alone as an argument because *expr* interprets it as the division operator (see WARNINGS below):

```
expr $a : '.*/\(.*\)' \| $a
```

A better representation of the previous example. The addition of the **//** characters eliminates any ambiguity about the division operator and simplifies the whole expression:

```
expr //$a : '.*/\(.*\)'
```

Return the number of characters in **$VAR**:

```
expr $VAR : '.*'
```

WARNINGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

```
expr $a = '='
```

resembles:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

SEE ALSO

sh(1), test(1), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

expr: SVID2, XPG2, XPG3, POSIX.2

NAME

expreserve - preserve editor buffer

SYNOPSIS

`expreserve`

DESCRIPTION

`expreserve` should be executed prior to executing `vi -r` to recover files that were being edited and were lost due to a client system crash in an HP Clustered Environment. **`expreserve`** recovers only those files owned by the user executing the command.

This command is intended for use only on client nodes in clustered client/server networks and is not useful on stand-alone HP-UX systems.

FILES

`/usr/lib/expreserve`

SEE ALSO

ex(1), vi(1).

NAME

f77, fort77 - FORTRAN 77 compiler

SYNOPSIS

`f77` [*options*] *file* ...
`fort77` [*options*] *file* ...

DESCRIPTION

`f77` is the HP-UX FORTRAN 77 compiler. The compiler accepts the following types of *file* arguments (see the DEPENDENCIES section for additional *file* types):

- Arguments that end with `.f` are compiled as FORTRAN 77 source files. If each source compiles successfully, its corresponding object file is left in the current directory in a file whose name is that of the source, with `.o` substituted for `.f`.
- Arguments that end with `.F` are processed by the C preprocessor, with the output written to a temporary file. The file is then compiled and removed.
- Arguments that end with `.o` are passed on to the linker (see IR ld (1)) to be linked into the final program.
- Arguments that end with `.r` are assumed to be `ratfor` source programs (see *ratfor*(1)). These are first transformed by the `ratfor` preprocessor then compiled, producing `.o` files.
- Arguments that end with `.c` or `.s` are assumed to be C or assembly source programs, and are compiled or assembled, producing `.o` files.

The `fort77` command provides a POSIX-compliant interface to the HP-UX FORTRAN 77 compiler. Except where indicated, functionality is identical to the `f77` command.

Arguments can be passed to the compiler through the **FCOPTS** environment variable as well as on the command line. The compiler picks up the value of **FCOPTS** and places its contents before any arguments on the command line. For example, in *sh*(1) or *ksh*(1)

```
FCOPTS=-v
export FCOPTS
f77 -L prog.f
```

is equivalent to

```
f77 -v -L prog.f
```

The `|` character can be used in the **FCOPTS** environment variable. Options appearing before `|` in **FCOPTS** are recognized before any options on the command line. Options appearing after `|` in **FCOPTS** are recognized after any options on the command line. For example,

```
FCOPTS="-O | -lmylib"
export FCOPTS
f77 -v prog.f
```

is equivalent to

```
f77 -O -v prog.f -lmylib
```

Except where indicated, when conflicting or related arguments to `f77` are given, arguments later in the list take precedence over the related arguments earlier in the list. An appropriate warning message may also be issued.

Options

`f77` and `fort77` recognize the following options:

-a	Issue warnings for non-ANSI features.
-A*secondary*	Generate errors for features not part of the ANSI 77 standard. (As a side-effect, the **-K** option is set if neither *secondary* option **R** nor **r** is specified.)
	secondary is one or more of the letters from the set **{hHmMnNrRsSaA}**, and specifies what exceptions to the standard are allowed. *secondary* is optional. If it is not specified, no exceptions to the ANSI standard are allowed. If *secondary* is specified, only the specified exceptions are allowed and all other exceptions are recognized with

an error. If *secondary* is an uppercase letter, no warning or error message is generated for the specified exception. If *secondary* is a lowercase letter, the exception specified generates a warning but not an error.

A *secondary* specification causes the following exceptions to be recognized:

H or **h**	Recognize Hollerith data as a non-ANSI exception. **H** produces no warnings; **h** produces warnings.
M or **m**	Allow MIL-STD-1753 extensions. **M** produces no warnings; **m** produces warnings.
N or **n**	Allow NAMELIST as a non-ANSI exception. **N** produces no warnings; **n** produces warnings.
R or **r**	Allow recursion. **R** produces no warnings; **r** produces warnings. As a side-effect, **DATA** statements amid executable statements produce errors.
S or **s**	Allow selected syntactical features: lower case letters, tabs, quotation marks, and symbolic names with a length greater than six characters. **S** produces no warnings; **s** produces warnings.
A or **a**	Allow recognition of all the exceptions listed above; all other ANSI extensions are disallowed. **A** produces no warnings for all exceptions allowed; **a** produces warnings.

+A
Align data using alignment rules where non-character items 4 bytes and larger are aligned on 2-byte boundaries instead of 4-byte boundaries. This is the same as specifying the **$HP1000 ALIGNMENT** directive.

+A*secondary*
Set the alignment of data items within FORTRAN **STRUCTURE**s, **COMMON** blocks, and **EQUIVALENCE** classes. Defined values for *secondary* are:

3	Set the alignment of data items within FORTRAN **STRUCTURE**s, **COMMON**, and **EQUIVALENCE** to be the same as the alignment produced by the Series 300/400 C compiler. This option is the same as the **$HP9000_300 ALIGNMENT** directive. This is the default on Series 300/400 systems.
8	Set the alignment of data items within FORTRAN **STRUCTURE**s, **COMMON**, and **EQUIVALENCE** to be the same as the alignment produced by the Series 800 C compiler. This is the same as the **$HP9000_800 ALIGNMENT** directive. This option is the default on Series 700 and 800 systems.

See the DEPENDENCIES section for more *secondary* values.

+apollo
Apollo Domain compatible defaults are enabled in data representation, intrinsics support, and procedure calling conventions. See the reference manual for **$APOLLO** directives that permit finer control over this feature.

+B
Treat the backslash character (****) as a C-like escape character.

-c
Suppress linking and produce an object (**.o**) file from each source file.

-C
Enable range checking. This is the same as the **$OPTION RANGE ON** directive.

-D
Compile debug lines as source statements. Source lines with a **D** or **d** in column 1 are treated as comments by default.

-D*name* [**=***def*]
Define a symbol *name* to the C preprocessor. If no *def* is given, *name* is defined as **1**. This applies to

`.F` suffix files only.

`+e`
Enable "other-vendor" compatibility mode for importing non-HP code. This is equivalent to specifying all the `+E` options (except `+E4`). It is also the same as specifying the **$NOSTANDARD** directive at the beginning of the program file. This option turns on the `-K` option.

`+es`
Allow more than 72 characters on a single source line (Extended Source).

`+E`*secondary*
This option enables specific "other-vendor" compatibility modes. Defined values for *secondary* are:

0 Enable extended-range `DO` loops and jumps into `IF` blocks. This option may degrade run-time performance.

1 Enable access to the industry-standard intrinsic routines **DATE**, **IDATE**, **SECNDS**, **SIZEOF**, **TIME**, **RAN**, and **EXIT**. This is the same as specifying the **$NOSTANDARD SYSTEM** directive.

2 Treat −1 as **.TRUE.** and zero as **.FALSE.** This is the same as specifying the **$NOSTANDARD LOGICALS** directive.

4 This option prints a zero before the decimal point when printing numbers through the **E**, **G**, and **F** format specifiers.

When the `+I300` option is used on the Series 300/400, this option does not print leading zeros. Instead, it allows the output of numeric data types (INTEGER, REAL, DOUBLE and QUAD) with alpha format specifiers (**A** and **R**) in a manner that is "other-vendor" compatible. This may produce incorrect results for output of Hollerith data with alpha format specifiers.

This is the same as specifying the **$NOSTANDARD IO** directive.

5 Allow multiple FORTRAN UNITs to be attached to the same HP-UX file. This is the same as specifying the **$NOSTANDARD OPEN** directive.

6 Enable full variable-format expressions as a compatibility extension. Partial support for variable-format expressions is provided on the Series 300 by default.

7 Use static storage for arguments passed to subroutines and functions.

`-F`
Apply the C preprocessor to `.F` files and put the output in a file of the same name with the suffix changed to `.f`. The `.f` files are not compiled.

`-g`
Generate additional information needed by the symbolic debugger **xdb** (see *xdb*(1)). This option is incompatible with optimization. `-g` overrides the `-s` option, regardless of the order in which they are encountered on the command line.

`-G`
Prepare object files for profiling with *gprof* (see *gprof*(1)).

`-I2`
Make default size of integers and logicals **INTEGER*2** and **LOGICAL*2**. This is the same as specifying the **$OPTION SHORT** directive.

`-I4`
Make default size of integers and logicals **INTEGER*4** and **LOGICAL*4**. This is the compiler's default.

`-I`*directory*
Add *directory* to the list of directories searched for **$INCLUDE** files whose names do not begin with a `/`. After searching the source directory and any `-I` directories, the following directories are searched: the current working directory, `/usr/include`. If the C preprocessor is used, then this option is passed through to it.

`-K`
Automatically SAVE all local variables in all subprograms. This option forces static storage for these

variables in order to provide a convenient path for importing FORTRAN 66 and FORTRAN 77 programs that were written to depend on static allocation of memory (that is, variables retaining their values between invocations of their containing program units). This option has two side-effects:

- All non-initialized variables are initialized to zero.
- The DATA statement can appear among executable statements.

-l*x*
Cause the linker to search first in the library named by **/lib/lib***x***.a** and/or **/lib/lib***x***.sl** then in **/usr/lib/lib***x***.a** and/or **/usr/lib/lib***x***.sl** See *ld*(1) for details. Also see the **-L***directory* option (**fort77** only).

-L
(**f77** only)
Write a program listing to the standard output during compilation. This option is required in order to enable the **$LIST ON** directive. The **-L** option overrides the **-V** option, regardless of the order in which they are encountered on the command line.

-L*directory*
(**fort77** only)
For libraries named in **-l** operands, look in the directory named by the *directory* pathname before looking in the standard places. Directories named in **-L** options are searched in the specified order. This option is effective only if it precedes the **-l** option on the command line (see *ld*(1) for details).

+mr
Permit the reading or writing of multiple unformatted records when an I/O statement's variable list requires or specifies more values than are in the record. This corresponds to the **$MULTI_RECORD** directive.

-n
Cause the output file from the linker to be marked *shared*.

-N
(**f77** only)
Prevent the output file from the linker from being marked *shared*.

-N
(**fort77** only)
Disallow any special code optimization. This option overrides the **-O** and **+O***n* options, regardless of the order in which they are encountered on the command line.

+N*tableN*
Set the initial size of internal compiler tables. These tables are expanded automatically by the compiler as needed. The letter *table* and integer *N* are required. Table sizes can be re-specified using one of the *table* letters and the number *N* as follows:

c	Initial size of control statements table (default is 20 table entries).
n	Initial size of the hash table of symbols (default is 401 table entries).
q	Initial size of equivalence table (default is 150 table entries).
s	Initial size of statement label table (default is 201 table entries).
x	Initial size of external symbol table (default is 200 table entries).

No spaces are permitted between the **+N** option and *table*, or between *table* and *N*.

For other values that *secondary* can assume, see the DEPENDENCIES section below.

-o *outfile*
Name the output file from the linker *outfile* instead of *a.out*. When **-o** is used with the **-S** option, the assembly code file is named *outfile* instead of *filename***.s**. When **-o** is used with the **-F** option, the output from **cpp** is named *outfile* instead of *filename.***f**.

-onetrip
Execute any DO loop at least once.

-O
Perform level 1 and level 2 optimization.

+O*optlevel*
Perform specific optimizations. Defined values for *optlevel* are:

1	Perform level 1 optimization.
2	Perform level 2 optimization.
3	Perform level 3 optimization.

-p
Prepare object files for profiling with *prof* (see *prof*(1)).

+ppu
Append underscores at the end of definitions of and references to externally visible symbols.

-q
Cause the output file from the linker to be marked *demand load*.

-Q
Prevent the output file from the linker from being marked *demand load*.

+Q*dfile*
Cause *dfile* to be read before compilation of each source file. *dfile* must contain only compiler directives. This option allows machine-dependent compiler directives to affect the compilation without requiring changes to the source code.

-R4
Make the default size of floating-point constants **REAL*4**. This is the compiler's default.

-R8
Make the default size of floating-point constants **REAL*8**. For example, the constant **3.5** is treated as if it were the constant `3.5D0`.

+R
Write cross reference and symbol table information to standard output during compilation. When used with the `fort77 -V` option, output is written to the `.l` file instead.

-s
Cause the output of the linker to be stripped of symbol table information (see *ld*(1) and *strip*(1)). This option is incompatible with symbolic debugging.

-S
Compile the named source files and leave the assembly language output in corresponding files whose names are suffixed with *.s* (no *.o* files are created).

-t*c*,*name*
Substitute or insert subprocess *c* with *name* where *c* is one or more implementation-dependent set of identifiers indicating the subprocess(es). Works in two modes:

1. If *c* is a single identifier, *name* represents the full path name of the new subprocess.
2. If *c* is a set of identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path names of the new subprocesses.

One or more values that *c* can assume are:

c	Compiler body (standard suffix is `f77pass1`)
0	Same as *c*
F	C preprocessor
r	`ratfor` preprocessor (standard suffix is `ratfor`).
a	Assembler (standard suffix is `as`)
l	Linker (standard suffix is `ld`).

For other values that *c* can assume, see the DEPENDENCIES section below.

-u
Force types of identifiers to be implicitly undeclared (same as specifying **IMPLICIT NONE**; no other **IMPLICIT** statements are permitted).

-U
Use uppercase for external names (default is lowercase).

`+U`
Distinguish upper case and lower case (make case significant). Keywords are recognized regardless of case.

`+U77`
Invoke support for the `BSD 3F` library. This option is incompatible with the `+I300` option.

`-v`
Enable the verbose mode, producing a step-by-step description of the compilation process on the standard error output.

`-V`
Produce a source listing. The listing is written to a listing (`.l`) file corresponding to each source (`.f`) file. If both `-V` and `-L` are specified with the `f77` command, `-L` takes precedence, regardless of the order in which they are encountered on the command line.

`-w`
Suppress warning messages. This is the same as specifying the `$OPTION WARNINGS OFF` directive.

`-w66`
Enable warnings about FORTRAN 66 features used.

`-Wc,`*arg1*[,*arg2*,...,*argN*]
Cause *arg1* through *argN* to be handed off to subprocess *c*. The *argi* take the form -*argoption* [, *argvalue*], where *argoption* is the name of an option recognized by the subprocess and *argvalue* is a separate argument to *argoption* where necessary. The *c* can assume the values listed under the `-t` option.

`-y`
Generate additional information needed by static analysis tools, and ensure that the program is linked as required for static analysis. This option is incompatible with optimization. `-y` overrides the `-s` option, regardless of the order in which they are encountered on the command line.

`-Y` [*lang*]
Enable 8-bit and 16-bit Native Language Support (NLS) in strings and comments. In the default case, NLS is not enabled.

`+z`
Generate Position Independent Code (PIC) with short displacements for use in shared libraries. The `+z` option overrides the `-p`, and `-G` options, regardless of the order in which they are encountered on the command line.

`+Z`
Generate Position Independent Code (PIC) with long displacements for use in shared libraries. The `+Z` option overrides the `-p`, and `-G` options, regardless of the order in which they are encountered on the command line.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported in comments and strings.

DIAGNOSTICS

The diagnostics produced by `f77` are intended to be self-explanatory. Errors and warnings are written to the standard error output. If a listing is requested (`-L` or `-V` options), errors and warnings are also written to the listing file.

EXAMPLES

This command illustrates how to have the compiler generate PA1.1 code for the program, but use the PA1.0 math libraries. (This applies only on Series 700 and 800 systems.)

```
f77 prog.f +DA1.1 -Wl,-L,/usr/lib,-L,/lib
```

DEPENDENCIES

Series 300/400 only

The `-t` and `-W` options allow *c* to assume the following values:

`p`	Procedure integrator (standard suffix is `fc0`).
`1`	Code generator (standard suffix is `f1`).
`2`	Peephole optimizer (standard suffix is `fc2`).
`g`	Global optimizer (standard suffix is `fc1`).

Specifying **`-Wg,-ALL`** causes the global optimizer to apply certain difficult optimizations to complicated procedures that it would not have otherwise attempted. If the **`-v`** flag is given to **`f77`**, the global optimizer is more verbose about those procedures that can benefit from the **`-Wg,-ALL`** option. Note that this option can greatly increase compilation time.

Specifying **`-W0,-l`** causes source file line numbers to be printed as assembly code comments for debugging purposes.

The **`+e`** option turns off register allocation of DO loop variables.

The **`+A`***secondary* option also allows *secondary* to assume the value:

`N` Set the alignment of data items within FORTRAN **`STRUCTURE`**s to be the same as the alignment produced by an industry standard FORTRAN compiler. This is the same as specifying the **`$NOSTANDARD ALIGNMENT`** directive.

The following implementation-specific options are supported:

`+bfpa` Generate code that uses the HP 98248A/B floating-point accelerator card if one is installed at run-time. If the card is not installed, floating-point operations are done on the MC68881/2 math coprocessor.

`+ffpa` Generate code for the HP 98248A/B floating-point accelerator card. This code does not run unless the card is installed.

`+I300` Use the pre-8.0 I/O library. This option is provided temporarily for backward compatibility with object files created before release 8.0. Attempting to mix old and new object files without this option results in unresolved references to names beginning with **`_F_`** or **`_Ftn_`**.

At a future release, the compatibility library will no longer be provided. Users and application providers are encouraged to migrate to the new library.

`+M` Prevent the compiler from generating in-line code for the MC68881/2 math coprocessor. Library routines are used (for *matherr* capability).

`+N`*tableN* Set the initial size of internal compiler tables. *table* can assume the following values:

`a`	Initial size of external label name storage table (default is 10000 bytes).
`e`	Initial number of expression tree nodes (default is 1000 entries).
`t`	Initial size of external symbol storage table (default is 40000 bytes).

No spaces are permitted between the **`+N`** option and *table*, or between *table* and *N*.

The **`TMPDIR`** environment variable specifies a directory for temporary files to be used instead of the default directories **`/tmp`** and **`/usr/tmp`**.

Series 700 *and* 800 only

The **`-ta,`***name* and **`-Wa,`***name* options are only effective when operating on a **`.s`** file.

The **`-t`** and **`-W`** options allow *c* to assume the following value:

`P` optimizing preprocessor (standard suffix is *ftnopp).*

The following implementation-specific option is supported:

`+800` This option provides compilation in 800-compatible mode. Pre-9.0 Series 800 compatible defaults are enabled in data representation, data alignment, argument passing conventions and I/O. See the reference manual for **`$HP9000_800`** directives that permit finer control over this feature.

It is necessary to use the **`+800`** option to produce object files that are compatible with object files produced by the pre-9.0 Series 800 FORTRAN compiler or with libraries compiled for use with the pre-9.0 Series 800 FORTRAN compiler. Alternatively, the **`$HP9000_800`** directive can be used to ensure compatibility with object files generated by the pre-9.0 Series 800 FORTRAN compiler.

The `+E0` option is ignored; extended-range DO loops and jumps into IF blocks are enabled by default.

The following additional options are recognized:

+autodblpad
Promotes all single precision floating point items to double precision, and all double precision floating point items to extended precision. Items affected are: constants, scalar variables, arrays, record fields, specific intrinsics, user-defined functions, and types specified in **ON** statements. Integer and logical items that might share storage space with a promoted item are padded to ensure that the storage sharing relationship that existed before promotion is maintained; some restrictions apply, see the reference manual for details. This option is incompatible with the **+OP** option. If both **+autodblpad** and **+OP** are specified, **+OP** is ignored. This is the same as specifying the **$AUTODBL DBLPAD** directive. See the reference manual for **$AUTODBL** directives that permit finer control over this feature.

+df*name*
This option is used with profile-based optimization and the **+P** option. It allows you to specify the name of the profile database file to use. This option is only needed when the file **flow.data** has been renamed. **flow.data** is created by running a program instrumented with profile-based optimization information.

+DA*model*
Generate code for a specific version of the PA-RISC architecture. *model* can be either a model number (e.g., `750` or `870`), or one of the following generic architecture specifications:

`1.0` Precision Architecture RISC, version 1.0. This is the default for all Series 800 models.
`1.1` Precision Architecture RISC, version 1.1. This is the default for all Series 700 models.

If this option is not used, code is generated for the architecture on which the program is compiled. Object code generated for PA-RISC 1.1 will not execute on PA-RISC 1.0 implementations. When the target architecture is `1.1` (whether explicit or implicit), the compiler will automatically access the `1.1` math libraries by inserting `-L /usr/lib/pa1.1` on the `ld` line.

+DS*model*
Perform instruction scheduling appropriate for a specific implementation of the PA-RISC architecture. *model* can be either a model number (e.g., `750` or `870`), or one of the following generic architecture specifications:

`1.0` Precision Architecture RISC, version 1.0.
`1.1` Precision Architecture RISC, version 1.1.

The default scheduling is based on the model number returned by **uname()** (see *uname*(2)). This option affects only the performance of the code by scheduling the code based on the specific latencies of the target implementation. The resulting code will execute correctly on other PA-RISC implementations (subject to the **+DA** option above).

+FP*flags*
Specifies how the run time environment for floating-point operations should be initialized at program startup. The default is that all behaviors are disabled. See ld(1) for specific values for *flags*. To dynamically change these settings at run time, refer to *fpgetround*(3M).

+I
Generate object code for application profiling using profile-based optimization. The results of profiling can be used for code optimization when the **+P** option is used. This option is incompatible with the **-G**, **-p**, **-s**, **-S**, and **-y** options. This option may not be used on the same command line as **+P**. See the **+P** option.

-lisam
Link the ISAM library into the program. ISAM (Indexed Sequential Access Method) is purchased separately.

+Obb*num*

Specify the maximum number of basic blocks allowed in a procedure being optimized at level 2. A basic block is a sequence of code with a single entry point and single exit point, with no internal branches. Optimizing procedures with a large number of basic blocks can take a long time and may use a large amount of memory. If the limit is exceeded, a warning is printed giving the name of the procedure and the number of basic blocks it contains, and level 1 optimization is done. The default value for this limit, if this option is not present, is 500. This option implies level 2 optimization (equivalent to -O or +O2), unless explicitly overridden with another +O*optlevel* option.

+OP[*n*] [*act*]

Invoke the Fortran optimizing preprocessor. The optional value *n* indicates the level of optimization to be performed by the preprocessor. The value of *n* can be from 0 to 4, with a default value of 2.

The optional argument *act* controls additional actions involving the preprocessor. The values that *act* can assume are:

P	Do not remove the intermediate transformed file. This intermediate file has the same name as the source, with a .P substituted for the .f. The default is for this file to be removed.
c	Only invoke the optimizing preprocessor, suppressing the compile and linking steps. The intermediate transformed file is not removed.

This option implies level 2 optimization (equivalent to -O or +O2), unless explicitly overridden with another +O*optlevel* option.

+OPunroll

Invoke the Fortran optimizing preprocessor to unroll loops where appropriate to improve run-time performance.

+P

Use profile information (created by using the +I option) to guide code generation and profile-based optimization. This option is incompatible with the -g, -S, and -y options. See the +I, +pgm*name*, and +df*name* options. See the *FORTRAN/9000 Reference* or *Programming on HP-UX* for more information on profile-based optimization.

+pgm*name*

This option is used with profile-based optimization and the +P option. It specifies the program *name* to use as the look-up string in the profile database file flow.data.

+T

Cause the running program to issue a procedure traceback for run-time errors. Enable proper handling of the ON statement.

Arguments whose names end with .P are assumed to be intermediate files from the optimizing preprocessor. Those files are processed in the same manner as .f files. The +OP option cannot be specified if .P arguments are also present.

AUTHOR

f77 and fort77 were developed by HP.

FILES

All Systems:

file.f	input file (FORTRAN source file)
file.F	input file (FORTRAN source with cpp directives)
file.r	input file (ratfor source file)
file.s	input file (assembly source file)
file.c	input file (C source file)
file.o	object file
a.out	linked executable output file
/lib/libc.a	C library; see Section 3 of this manual and *intro*(3).
/lib/libm.a	math library (PA-RISC 1.0 version on Series 700 and 800)
/lib/lib*.sl	sharable versions
/usr/lib/lib*.sl	of libraries
/usr/lib/end.o	symbolic debugger string buffer
/usr/bin/f77	compiler driver

`/usr/bin/fort77`	compiler driver
`/usr/lib/libU77.a`	library containing +U77 routines
`/usr/libp/libU77.a`	profiling library containing +U77 routines
`/usr/lib/nls/C/libU77.cat`	libU77 message catalog
`/usr/man/man3.Z/*.3f`	man pages for libU77 routines
`/bin/as`	assembler
`/usr/lib/f77pass1`	compiler
`/tmp/fc*`	compiler temporary files
`/tmp/fcp*`	temporary file for cpp output

Series 300/400, additional files

`/lib/frt0.o`	run-time initialization and startup code
`/lib/gfrt0.o`	startup code for use with profiling via `gprof`
`/lib/mfrt0.o`	startup code for use with profiling via `prof`
`/lib/fc0`	procedure integrator (level 3 optimization)
`/lib/fc1`	FORTRAN global optimizer (level 2 optimization)
`/lib/fc2`	FORTRAN peephole optimizer (level 1 optimization)
`/lib/f1`	compiler code generator
`/usr/lib/libFext.a`	compatibility function library
`/usr/lib/libF77.a`	intrinsic function library
`/usr/lib/libquad.a`	REAL*16 intrinsic functions
`/usr/lib/libIO77.a`	I/O library (Series 700/800 compatible)
`/usr/lib/libI77.a`	backward-compatible I/O library (there is no `/usr/lib/libI77.sl`)
`/usr/lib/libvis.a`	Vector Instruction Set library (there is no `/usr/lib/libvis.sl`)
`/lib/libp/libFext.a`	profiling version of libFext
`/lib/libp/libF77.a`	profiling version of libF77
`/usr/libp/libquad.a`	profiling version of libquad
`/usr/libp/libIO77.a`	profiling version of libIO77
`/usr/libp/libI77.a`	profiling version of libI77
`/usr/libp/libvis.a`	profiling version of libvis
`/usr/tmp/fxr*`	temporary files for cross referencing
`/usr/lib/nls/$LANG/f77IO.cat`	I/O message catalog
`/usr/lib/nls/$LANG/f77pass1.cat`	compiler error message catalog
`/usr/lib/nls/$LANG/f77.cat`	driver error message catalog

Series 700 *and* 800, additional files

`/lib/icrt0.o`	startup code for use with profile-based optimization
`/lib/crt0.o`	run-time startup
`/lib/gcrt0.o`	startup code for use with profiling via `gprof`
`/lib/mcrt0.o`	startup code for use with profiling via `prof`
`/usr/lib/libisamstub.a`	archive version of ISAM stubs
`/usr/lib/libisamstub.sl`	shared version of ISAM stubs
`/usr/lib/libisam.a`	archive version of Informix's C-ISAM library
`/usr/lib/sched.models`	processor implementation file
`/lib/pa1.1/libm.a`	PA-RISC 1.1 Math Library
`/lib/pa1.1/libm.sl`	PA-RISC 1.1 Math Library (shared-bound version)
`/lib/pa1.1/libvec.a`	Vector library support for the optimizing preprocessor, tuned for PA-RISC 1.1
`/usr/lib/libvec.a`	Vector library support for the optimizing preprocessor, tuned for PA-RISC 1.0
`/usr/lib/ftnopp`	FORTRAN optimizing preprocessor.
`/usr/lib/libcl.a`	FORTRAN math and I/O libraries
`/usr/lib/libportnls.a`	NLS run-time library
`/usr/lib/libnlsstubs.a`	stubs for NLS library routines
`/usr/lib/nls/$LANG/f77_msgs.cat`	800 compiler error message catalog

`/usr/lib/libfsys.a`	pre-9.0 FORTRAN-to-HP-UX system call interfaces
`/usr/lib/nls/$LANG/ftnopp.cat`	optimizing preprocessor message catalog

SEE ALSO

Program management and analysis tools:

fsplit(1)	split f77, ratfor, or efl files
lintfor(1)	check for erroneous, non-portable, or wasteful use of FORTRAN
ratfor(1)	preprocess rational FORTRAN dialect into ordinary FORTRAN
softstatic(1)	static analysis tool (shipped with SoftBench software)

Profiling and debugging tools:

gprof(1)	display call graph profile data
prof(1)	display profile data
xdb(1)	invoke the FORTRAN, C, C++, and Pascal symbolic debugger

System tools:

ar(1)	create archived libraries
as(1)	translate assembly code to machine code
asa(1)	interpret ASA carriage-control characters
cc(1)	invoke the HP-UX C compiler
c89(1)	invoke the HP-UX POSIX-conformant C compiler
cpp(1)	invoke the the C language preprocessor
ld(1)	invoke the link editor

Miscellaneous:

matherr(3M)	trap math errors
strip(1)	strip symbol and line number information from an object file

STANDARDS CONFORMANCE

`fort77`: POSIX.2

NAME

factor, primes - factor a number, generate large primes

SYNOPSIS

`factor` [*number*]

`primes` [*start* [*stop*]]

DESCRIPTION

If no arguments are provided on the command line, `factor` waits for a number to be typed in. If a positive number is typed, it factors the number and print its prime factors; each one is printed the proper number of times. It then waits for another number. `factor` exits if it encounters a zero or any non-numeric character.

If an argument is provided on the command line, `factor` factors the number as above, then exits.

Maximum time to factor is proportional to sqrt(*n*) and occurs when *n* is prime or the square of a prime.

The largest number that can be dealt with by `factor` is 1.0e14.

`primes` prints prime numbers between a lower and upper bound. If no arguments are provided on the command line, `primes` waits for two numbers to be typed in. The first number is interpreted as the lower bound; the second as the upper bound. All prime numbers in the resulting inclusive range are printed.

If *start* is specified, all primes greater than or equal to *start* are printed. If both *start* and *stop* are given, all primes occurring in the inclusive range *start* through *stop* are printed.

start and *stop* values must be integers represented as long integers.

If the stop value is omitted in either case, *primes* runs either until overflow occurs or until it is stopped by typing the interrupt character.

The largest number that can be dealt with by *primes* is 2,147,483,647.

DIAGNOSTICS

Both commands print `Ouch` when the input is out of range, illegal characters are encountered, or when *start* is greater than *stop*.

EXAMPLES

Print the prime factorization for the number 12:

```
factor 12
```

Print all prime numbers between 0 and 20:

```
primes 0 20
```

NAME

file - determine file type

SYNOPSIS

`file` [`-m` *mfile*] [`-c`] [`-f` *ffile*] *file* ...

DESCRIPTION

`file` performs a series of tests on each *file* in an attempt to classify it. If *file* appears to be an ASCII file, `file` examines the first 512 bytes and tries to guess its language. If *file* is an executable `a.out` file, `file` prints the version stamp, provided it is greater than 0 (see the description of the `-V` option in *ld*(1)).

`file` uses the file `/etc/magic` to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of `/etc/magic` explains the format.

Options

`file` recognizes the following command-line options:

`-m` *mfile*	use alternate magic file *mfile*.
`-c`	Check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file classification is done when this option is specified.

`-f` *ffile* Obtain the list of files to be examined from file *ffile*. `file` classifies each file whose name appears in *ffile*.

SEE ALSO

ld(1).

STANDARDS CONFORMANCE

`file`: SVID2, XPG2

NAME

find - find files

SYNOPSIS

`find` *path-name-list* [*expression*]

DESCRIPTION

`find` recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, one or more path names) seeking files that match a Boolean *expression* written in the primaries given below. By default, `find` does not follow symbolic links.

The Boolean expression is evaluated using short-circuit evaluation. This means that whenever the result of a Boolean operation (AND or OR) is known from evaluating the left-hand argument, the right-hand argument is not evaluated.

In the descriptions of the primaries, the argument *n* is used as a decimal integer; +*n* means more than *n*, -*n* means less than *n*, and *n* means exactly *n*.

The following primaries are recognized:

-depth A position-independent term which causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when `find` is used with *cpio*(1) to transfer files that are contained in directories without write permission. It is also useful when using *cpio*(1) and the modification dates of directories must be preserved. Always true.

-follow A position-independent term which causes `find` to follow symbolic links. Always true.

-hidden A position-independent term which causes `find` to include hidden subdirectories (context-dependent files) in the directory hierarchy of each path name in the *path-name-list*. In addition, the normally hidden components of the path name become visible as when using the **-print** primary (see *cdf*(4)). Always true.

-fsonly *type* A position-independent term which causes `find` to stop descending any directory whose file system is not of the type specified by *type*, where *type* is one of **nfs**, **cdfs**, or **hfs**. In this context, mount points inherit the fstype of their parent directory. This means that when **-fsonly hfs** has been specified and `find` encounters an NFS mount point that is mounted on an HFS file system, the mount point will be visited but entries below that mount point will not. It is important to note that when **-fsonly nfs** has been specified, any HFS file systems that are beneath the mount point of an NFS file system are not traversed. Always true.

-xdev A position-independent term that causes `find` to avoid crossing any file system mount points that exist below starting points enumerated in *path-name-list*. The mount point itself is visited, but entries below the mount point are not. Always true.

-mountstop Identical to **-xdev**. This primary is provided for backwards compatibility only. Use of **-xdev** is preferred over **-mountstop**.

-name *file* True if *file* matches the last component of the current file name. The matching is performed as specified by Pattern Matching Notation (see *regexp*(5)).

-path *file* Same as **-name** except the full path (as would be output by **-print**) is used instead of just the basename. Note that **/** characters are not treated as a special case. For example, ***/.profile** would match **./users/fred/.profile**.

-perm [-]*mode* In this primary, the argument *mode* is used to represent file mode bits. The argument is identical in format to the *mode* operand as described in *chmod*(1), with the exception that the first character must not be the **-** operator. When using the symbolic form of *mode*, the starting template is assumed to have all file mode bits cleared.

If the leading minus is omitted, this primary is true when the file permission bits exactly match the value of *mode*. Bits associated with the symbolic attributes **s** (setuid, setgid) and **t** (sticky-bit) are ignored when the minus is omitted.

If *mode* is preceded by a minus, this primary is true if all of the bits that are set in *mode* are also set in the file permission bits. In this case, the bits associated with the symbolic attributes `s` and `t` are significant.

`-fstype` *type* True if the file system to which the file belongs is of type *type*, where *type* is one of `hfs`, `cdfs`, or `nfs`.

`-type` *c* True if the type of the file is *c*, where *c* is:

`f`	regular file
`d`	directory
`b`	block special file
`c`	character special file
`p`	FIFO (named pipe)
`l`	symbolic link
`s`	socket
`n`	network special file
`M`	mount point
`H`	hidden directory (see *cdf*(4))

The use of `-type H` implies the `-hidden` primary (see above).

`-links` *n*
True if the file has *n* links.

`-user` *uname*
True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the `/etc/passwd` file, it is taken as a user ID. The *uname* operand can be preceded by a `+` or `-` to modify the comparison operation as described previously.

`-group` *gname*
True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the `/etc/group` file, it is taken as a group ID. The *gname* operand can be preceded by a `+` or `-` to modify the comparison operation as described previously.

`-devcid` *cname*
True if the file is a block or character special file whose *st_rcnode* value is equal to the cnode id of the cnode *cname*. If *cname* is numeric and does not appear as a cnode name in the `/etc/clusterconf` file, it is taken as a cnode ID . See *mknod*(2)) for a description of cnode-specific device files. The *cname* operand can be preceded by a `+` or `-` to modify the comparison operation as described previously.

`-nouser`
True if the file belongs to a user ID that is not listed in the password database. See *passwd*(4).

`-nogroup`
True if the file belongs to a group ID that is not listed in the group database. See *group*(4).

`-nodevcid`
True if the file is a block or character special file whose *st_rcnode* value is not listed in the `/etc/clusterconf` file. See *mknod*(2).

`-size` *n*[`c`]
True if the file is *n* blocks long. If *n* is followed by a `c`, the size is in bytes.

`-atime` *n*
True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by `find` itself.

`-mtime` *n*
True if the file has been modified in *n* days.

`-ctime` *n*
True if the file inode has been changed in *n* days.

`-newer` *file*
True if the current file has been modified more recently than the argument *file*.

`-newer[acm[acm]]` *file*
True if the indicated time value of the current file is newer than the indicated time of *file*. The time values are indicated from the set of characters `a`, `c`, and `m`, where:

`a`	The time the file was last accessed
`c`	The time the inode of the file was last modified
`m`	The time the file was last modified

The first [`acm`] specifies the time of the current file, the second [`acm`] specifies the time of *file*. If only one [`acm`] is specified, the second time specifier defaults to `m`.

`-inum` *n*
True if the file serial number (inode number) is *n*. Note that file serial numbers are unique only within a given file system. Therefore, matching file serial numbers does not guarantee that the referenced files are the same unless you restrict the search to a single file system.

`-linkedto` *path*
True if the file is the same physical file as the file specified by *path* (i.e. linked to *path*). This primary is similar to `-inum`, but correctly detects when a file is hard-linked to *path*, even when multiple file systems are searched.

`-print`
Causes the current path name to be printed. Always true.

`-exec` *cmd*
True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by a semicolon (semicolon is special to the shell and must be escaped). Any command argument `{}` is replaced by the current path name.

`-ok` *cmd*
Same as `-exec` except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing `y`.

`-cpio` *device*
Write the current file on *device* in *cpio*(4) format (5120-byte records). The use of `-cpio` implies `-depth`. Always true.

`-ncpio`
Same as `-cpio` but adds the `-c` option to `cpio`. The use of `-ncpio` implies `-depth`. Always true.

`-prune`
If the current entry is a directory, cause `find` to skip that directory. This can be useful to avoid walking certain directories, or to avoid recursive loops when using `cpio -p`. Note, however, that `-prune` is useless if the `-depth` option has also been given. See the description of `-only` and the EXAMPLES section, below, for more information. Always true.

`-only`
This is a positive-logic version of `-prune.` A `-prune` is performed after every directory, unless `-only` is successfully evaluated for that directory. As an example, the following three commands are equivalent:

```
find . -fsonly hfs -print
find . -print -fstype hfs -only
find . -print ! -fstype hfs -prune
```

Note, however, that `-only` is useless if the `-depth` option has also been given. Always true.

(*expression*)
True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

Primaries can be combined by using the following operators (in order of decreasing precedence):

`!` *expression*
Logical NOT operator. True if *expression* is not true.

expression `-a` *expression*
Logical AND operator. True if both of the *expression*s are true.

expression `-o` *expression*
Logical OR operator. True if either or both of the *expression*s are true.

If *expression* is omitted, or if none of `-print`, `-ok`, `-exec`, `-cpio`, or `-ncpio` are specified, `-print` is assumed.

Access Control Lists

The following primary enables the user to search for access control list entries:

`-acl` *aclpatt* True if the file's access control list matches an access control list pattern or contains optional access control list entries (see *acl*(5)). The `-acl` primary has three forms:

`-acl` *aclpatt* Match all files whose access control list includes all (zero or more) pattern entries specified by the *aclpatt* pattern.

`-acl` =*aclpatt*
Match a file only if its access control list includes all (zero or more) pattern entries specified by the *aclpatt* pattern, and every entry in its access control list is matched by at least one pattern entry specified in the *aclpatt* pattern.

`-acl opt` Match all files containing optional access control list entries.

The *aclpatt* string of the `-acl` primary can be given as an operator or short form pattern; see *acl*(5).

By default, `-acl` is true for files whose access control lists include all the (zero or more) access control list patterns in *aclpatt*. A file's access control list can also contain unmatched entries.

If *aclpatt* begins with =, the remainder of the string must match all entries in a file's access control list.

The *aclpatt* string (by default, or the part following =) can be either an access control list or an access control list pattern. However, if it is an access control list, *aclpatt* must include at least the three base entries ((*u*.%, mode), (%.*g*, mode), and (%.%, mode)).

As a special case, if *aclpatt* is the word `opt`, the primary is true for files with access control list entries.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name matching.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in pattern matching notation.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, `find` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Search the two directories `/example` and `/new/example` for files containing the string `Where are you` and print the names of the files:

```
find /example /new/example -exec grep -l 'Where are you' {} \;
```

Remove all files named `a.out` or `*.o` that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

Note that the spaces delimiting the escaped parentheses are required.

Print the names of all files in `/bin` that are context-dependent; that is, hidden directories:

```
find /bin -type H -print
```

Print the names of all files on this machine. Avoid walking `nfs` directories while still printing the `nfs` mount points:

```
find / -fsonly hfs -print
```

Copy the entire file system to a disk mounted on `/Disk`, avoiding the recursive copy problem. Both commands are equivalent (note the use of `-path` instead of `-name`):

```
cd /; find . ! -path ./Disk -only -print | cpio -pdxm /Disk

cd /; find . -path ./Disk -prune -o -print | cpio -pdxm /Disk
```

Copy the root disk to a disk mounted on `/Disk`, skipping all mounted file systems below `/`. Note that `-xdev` does not cause `/` to be skipped, even though it is a mount point. This is because `/` is the starting point and `-xdev` only affects entries `below` starting points.

```
cd /;  find . -xdev -print | cpio -pdm /Disk
```

Change permissions on all regular files in a directory subtree to mode 444, and permissions on all directories to 555:

```
find <pathname> -type f -print | xargs chmod 444
find <pathname> -type d -print | xargs chmod 555
```

Note that output from `find` was piped to *xargs*(1) instead of using the `-exec` primary. This is because when a large number of files or directories are to be processed by a single command, the `-exec` primary spawns a separate process for each file or directory, whereas *xargs* collects filenames or directory names into multiple arguments to a single *chmod* command, resulting in fewer processes and greater system efficiency.

Access Control List Examples

Find all files not owned by user "karl" that have access control lists with at least one entry associated with "karl", and one entry for no specific user in group "bin" with the read bit on and the write bit off:

```
find  /  ! -user karl -acl 'karl.*, %.bin+r-w' -print
```

Find all files that have a read bit set in any access control list entry:

```
find  /  -acl '*.*+r' -print
```

Find all files that have the write bit unset and execute bit set in every access control list entry:

```
find  /  -acl '=*.*-w+x' -print
```

Find all files that have optional access control list entries:

```
find  /  -acl opt -print
```

WARNINGS

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP_UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

DEPENDENCIES

NFS The `-acl` primary is always false for NFS files.

AUTHOR

`find` was developed by AT&T and HP.

FILES

/etc/clusterconf	cnode names
/etc/group	group names
/etc/passwd	user names
/etc/mnttab	mount points

SEE ALSO

chacl(1), chmod(1), cpio(1), sh(1), test(1), xargs(1) mknod(2), stat(2), cpio(4), fs(4), group(4), passwd(4), cdf(4), acl(5), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

find: SVID2, XPG2, XPG3, proposed POSIX.2 FIPS (June 1990)

STANDARDS CONFORMANCE

`find`: SVID2, XPG2, XPG3, POSIX.2

NAME

findmsg, dumpmsg - create message catalog file for modification

SYNOPSIS

findmsg [**-a**] *file* ...
dumpmsg *file* ...

DESCRIPTION

findmsg extracts messages from a C program source *file* and writes them to the standard output in a format suitable for input to **gencat** (see *gencat*(1)). If multiple input files are specified and the **-a** option is not used, the files are processed sequentially such that message-catalog comment lines identifying the input *file* are written before the output for each input *file*. **gencat** does not accept the resulting output if any two source files contain messages belonging to the same set number. The **-a** option causes **findmsg** to merge identically numbered sets from multiple input files so that **gencat** can process the **findmsg** output.

findmsg scans the source files for uncommented lines with one of the following three formats embedded within it:

```
catgets(any_var,NL_SETN,n,"message")

"message"              /* catgets n */

/* catgets n */        "message"
```

or any combination of these formats wholly contained on a single physical line. Any number of spaces or tabs can separate the **catgets** comment from the *message*. The digit *n*, which can be any valid message number (see *gencat*(1)), is combined with the *message* string to produce a message-catalog source line. The message source line is assigned to the set whose number is the current value of **NL_SETN** as set by the last **#define** directive encountered. If **NL_SETN** has not yet been defined when a message line is found, the message is output without a set number specification. If more than one message is found belonging to the same set and message number, the last message found is output; any others are silently discarded. Conditional compilation and **#include** instructions in the C source files are ignored.

dumpmsg extracts messages from a message catalog *file* created by **gencat**. Messages are written to standard output in a format suitable for editing and re-input to **gencat**.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of messages as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **findmsg** and **dumpmsg** behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

For historical reasons, **findmsg** also recognizes the formats:

```
nl_msg(n,"message")

"message"              /* nl_msg n */

/* nl_msg n */         "message"
```

Use of these formats is not recommended.

AUTHOR

findmsg was developed by HP.

SEE ALSO

findstr(1), gencat(1), insertmsg(1), catgets(3C).

NAME

findstr - find strings for inclusion in message catalogs

SYNOPSIS

`findstr` *file ...*

DESCRIPTION

`findstr` examines files of C source code for uncommented string constants which it places, along with the surrounding quotes, on the standard output, preceding each by the file name, start position, and length. This information is used by `insertmsg` (see *insertmsg*(1)). `findstr` does not output strings that are parameters of the `catgets()` routine (see *catgets*(3C)).

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of comments and string literals as single- and/or multi-byte characters.

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `findstr` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

`findstr` outputs initialization strings of static string variables. Calling `insertmsg` with these strings causes their replacement with a call to `catgets()` (see *catgets*(3C)). Since the initializer must be a string, this assignment results in an invalid C declaration. For example, the following line:

```
static char *x[] = "message"
```

is modified by `insertmsg` (see *insertmsg*(1)) to:

```
static char *x[] = (catgets(catd,NL_SETN,1,"""message"""))
```

These strings should be manually removed from `findstr` output before being input to `insertmsg`.

SEE ALSO

insertmsg(1).

NAME

finger - user information lookup program

SYNOPSIS

`finger` [*options*] *user_name* ...

DESCRIPTION

By default, `finger` lists for each *user_name* on the system:

- Login name,
- Full given name,
- Terminal write status (if write permission is denied),
- Idle time,
- Login time,
- User's home directory and login shell,
- Any plan the user has placed in file `.plan` in their home directory,
- Project on which they are working from the file `.project`, also in the home directory,
- office location and phone number (if known).

Idle time is in minutes if listed as a single integer, hours and minutes if a `:` is present, or days and hours if a `d` is present. Account names as well as first and last names of users are accepted.

`finger` can also be used to list users on a remote machine. The format for *user_name* is *user_name@host*. If *user_name* is not specified, the remote system (HP-UX or non-HP-UX) uses its default standard format for listing user information.

Options

`finger` recognizes the following options:

`-b`	Suppress printing the user's home directory and shell
`-f`	Suppress printing the header that is normally printed in a short-format printout.
`-f`	Suppress printing the `.project` file in a long-format printout.
`-i`	Force "idle" output format. Similar to short format except that only the login name, terminal, login time, and idle time are printed.
`-l`	Force long output format.
`-m`	Match arguments only on user name.
`-p`	Suppress printing of the `.plan` files
`-q`	Force quick output format. Similar to short format except that only the login name, terminal, and login time are printed.
`-R`	Print the user's host name.
`-s`	Force short output format.
`-w`	Suppress printing the full name in a short-format printout.

WARNINGS

Only the first line of the `.project` file is printed.

AUTHOR

`finger` was developed by the University of California, Berkeley.

FILES

`/etc/utmp`	who file
`/etc/wtmp`	last login file
`/etc/passwd`	for users names, offices, ...
`~/.plan`	plans
`~/.project`	projects

SEE ALSO

who(1).

NAME

fixman - fix manual pages for faster viewing with man(1)

SYNOPSIS

fixman

DESCRIPTION

fixman is a shell script that processes all ordinary files in default or specified **man*** and/or **cat*** directories. **fixman** unexpands spaces to tabs where possible and removes all character-backspace pairs (which usually exist to cause overstriking or underscoring for printer output). Removal of unnecessary character sequences improves the speed of *man*(1), and reduces disk space consumption. **fixman** should be run after using **catman** to create formatted, **cat**-able manual entries from unformatted, **nroff**-compatible source files (see *catman*(1M)).

By default, **fixman** searches for **cat*** subdirectories in the following parent directories in the order indicated:

- **/usr/man**
- **/usr/contrib/man**
- **/usr/local/man**

If the **MANPATH** environment variable is set, the directory paths specified by **MANPATH** are searched for **cat*** directories.

fixman does not remove duplicate blank lines. Thus, all files remain a multiple of one page (66 lines) long and can still be passed directly to **lp** (see *lp*(1)). (Note that *man*(1) normally uses **more -s** to accomplish this removal.)

To ensure success, the script should be run by a user who has appropriate privileges. It can take up to an hour or more to complete, depending on system speed. As a side-effect, file ownerships and permissions may be changed.

EXTERNAL INFLUENCES

Environment Variables

MANPATH, if set, defines the directories to be searched for **cat**-able manual entries.

WARNING

If the value of **MANPATH** is not the same while **fixman** is running as it was when **catman** was run or when manpage files were installed, some files may be missed and not processed (see *catman*(1M)).

FILES

/man/cat	Directories containing **nroff**-formatted versions of manual entries

AUTHOR

fixman was developed by HP.

SEE ALSO

catman(1M), chmod(1), expand(1), lp(1), man(1), mv(1), rmnl(1), sed(1), environ(5).

NAME

fold - fold long lines for finite width output device

SYNOPSIS

`fold` [`-s`] [`-w` *width*] [*file* ...]

Obsolete form:

`fold` [`-s`] [`-`*width*] [*file* ...]

DESCRIPTION

`fold` is a filter that folds the contents of the specified files, breaking the lines to have maximum width *width*. If no files are specified or if a *file* name of `-` is specified, the standard input is used.

If backspace or tab characters are encountered in the input, they are treated specially as follows:

Backspace	The current count of line width is decremented by one, although the count never becomes negative. Thus, the character sequence *character-backspace-character* counts as using one column position, assuming both characters each occupy a single column position.
Tab	Each tab character encountered advances the column position pointer to the next tab stop. Tab stops are set 8 columns apart at column positions 1, 9, 17, 25, 33, etc.

Options

`fold` recognizes the following options and command-line arguments:

`-s`	Break the line on the last blank character found before the specified number of column positions. If none are found, break the line at the specified line length.
`-w` *width* -*width*	Specify the maximum line length, in column positions. Default is 80. *width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using **`expand`** before processing by `fold` (see *expand*(1)). The -*width* option is obsolescent and may be removed in a future release.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text as single- and/or multi-byte characters.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `fold` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

SEE ALSO

expand(1).

STANDARDS CONFORMANCE

`fold`: POSIX.2

NAME
forder - convert file data order

SYNOPSIS
`forder [-a] [-l] [-n] [`*file* ...`]`

DESCRIPTION
The text orientation (mode) of a file can be right-to-left (non-Latin) or left-to-right (Latin). This text orientation can affect the way data is arranged in the file. The data arrangements that result are called screen order and keyboard order. `forder` converts the order of characters in the file from screen order to keyboard order or vice versa.

`forder` reads the concatenation of input files (or standard input if none are given) and produces on standard output a converted version of its input. If `-` appears as an input file name, `forder` reads standard input at that point (use `--` to delimit the end of options in such instances).

`forder` converts input files for all languages that are read from right-to-left. Unless the `-a` option is used, the command merely copies input files to standard output for languages that are read from left-to-right.

Options
`forder` recognizes the following options:

- `-a` Convert file data order for languages read from left-to-right.
- `-l` Identify the file as having been created in Latin mode.
- `-n` Identify the file as having been created in non-Latin mode.

EXTERNAL INFLUENCES
Environment Variables
The `LANGOPTS` environment variable determines the mode and order of the file. The syntax of `LANGOPTS` is:

[*mode*] [*_order*]

where *mode* describes the mode of a file: `l` represents Latin mode, and `n` represents non-Latin mode. Non-Latin mode is assumed for values other than `l` and `n`. The *order* describes the data order of a file: `k` is keyboard, and `s` is screen. Keyboard order is assumed for values other than `k` and `s`. Mode information in `LANGOPTS` can be overridden from the command line.

The `LC_ALL` environment variable determines the direction of a language (left-to-right or right-to-left).

The `LC_NUMERIC` environment variable determines whether a language has alternative numbers.

The `LANG` environment variable determines the language in which messages are displayed.

International Code Set Support
Single-byte character code sets are supported.

EXAMPLES
The following command begins with *file1*, which exists in screen order, converts it to keyboard order, sorts the keyboard-ordered output, converts it back to screen order, and redirects the output to *file2*. Note that `-n` is given to inform `forder` that *file1* was created in non-Latin mode.

```
forder -n file1 | sort | forder -n > file2
```

WARNINGS
It is the user's responsibility to ensure that the `LANGOPTS` environment variable accurately reflects the status of the file.

If present, alternative numbers always have a left-to-right orientation.

AUTHOR
`forder` was developed by HP.

SEE ALSO
environ(5), hpnls(5), strord(3C), nljust(1).

NAME
from - who is my mail from?

SYNOPSIS
`from` [`-s` *sender*] [*user*]

DESCRIPTION
`from` prints the mail header lines in your mailbox file to show who sent you mail. If *user* is specified, *user*'s mailbox is examined instead of your own. If the `-s` option is given, only headers of mail from *sender* are printed.

EXAMPLES
List header lines for all current mail in your mailbox that was sent by `ken`.

```
from -s ken
```

FILES
`/usr/mail/*`

AUTHOR
`from` was developed by the University of California, Berkeley.

SEE ALSO
biff(1), mail(1), prmail(1).

NAME

fsplit - split f77, ratfor, or efl files

SYNOPSIS

`fsplit` *options files*

DESCRIPTION

`fsplit` splits the named *file*(s) into separate files, each containing one procedure. A procedure includes `blockdata`, `function`, `main`, `program`, and `subroutine` program segments. Procedure *X* is put in file *X*`.f`, *X*`.r`, or *X*`.e`, depending on the language option chosen, with the following exceptions: `main` is put in the file *MAIN*`.[efr]` and unnamed `blockdata` segments in the files *blockdataN*`.[efr]` where *N* is a unique integer value for each file.

`fsplit` recognizes the following command-line options:

`-f`	(default) Input files are *f77*.
`-r`	Input files are `ratfor`.
`-e`	Input files are `Efl`.
`-s`	Strip `f77` input lines to 72 or fewer characters with trailing blanks removed.

WARNINGS

`fsplit` does not respect in-line compiler directives that may occur between functional units such as:

```
$ optimize assume_parm_types_matched
```

SEE ALSO

csplit(1), efl(1), f77(1), ratfor(1), split(1).

NAME

ftio - faster tape I/O

SYNOPSIS

`ftio -o` | `-O` [`achpvxAEHLM`] [`-B` *blksize*] [`-D` *type*] [`-K` *comment*] [`-L` *filelist*] [`-N` *datefile*] [`-S` *script*] [`-T` *tty*] [`-Z` *nobufs*] *tapedev* [*pathnames*] [`-F` *ignorenames*]

`ftio -i` | `-I` [`cdfmptuvxAEMPR`] [`-B` *blksize*] [`-S` *script*] [`-T` *tty*] [`-Z` *nobufs*] *tapedev* [*patterns*]

`ftio -g` [`v`] *tapedev* [*patterns*]

DESCRIPTION

`ftio` is a tool designed specifically for copying files to 9-track magnetic tape drives. It should perform faster than either `cpio` or `tar` in comparable situations (see *cpio*(1) and *tar*(1)). `ftio` uses multiple processes (to read/write the file system and to write/read the tape device), with large amounts of memory sharing between processes as well as a large block size for reading and writing to the tape.

`ftio` is compatible with `cpio` in that output from `cpio` is always readable by `ftio`, and output from `ftio` is readable by `cpio`, except as explained in the cpio Compatibility section below.

`ftio` must be invoked with exactly one of the options `-o`, `-O`, `-i`, `-I`, or `-g`. The option can be followed by modifiers which must appear immediately after the option with no spaces between the option and the modifier, as in `ftio -idxE` (see Modifiers section below).

tapedev specifies the name of an output file. A device on a remote machine can be specified in the form

machine **:** *device*

`ftio` creates a server, `/etc/rmt`, on the remote machine to access the tape device.

Options

`ftio` recognizes the following options:

`-o` Copy out files onto *tapedev* together with path name and status information. If *pathnames* are specified, `ftio` recursively descends *pathnames* looking for files, and copies those files onto *tapedev*. If *pathnames* are not specified, `ftio` reads the standard input to obtain a list of path names to copy. In addition to copying the files onto the tape set, `ftio` generates, for each tape in the tape set, a tape header containing the current tape number, machine node name and type, operating system name, release and version numbers (all from the `uname()` system call — see *uname*(2)), username of the backup initiator, time and date of the backup, number of consecutive times the current media has been used, a comment field, and other items used internally by `ftio`. The tape header is separated from the main body of the archive by an end of file mark. The tape header can be read by invoking `cat` with the device file name as the first argument (see *cat*(1)). Note that device files written with the `-o` option (such as `/dev/tty03`) are not transportable to other HP-UX implementations.

`-O` Copy out files in the same way as `ftio -ocva` when no modifiers are used with the `-O`. However, if the file `.ftiorc` exists in the user's home directory, `ftio` opens this file and scans for lines preceded by `O=`. Options defined on matching lines are passed to `ftio` as if they had been passed in the original command. See EXAMPLES section.

`-i` Extract, or copy in, files from *tapedev*, which is assumed to be the product of a previous `ftio -o` operation. Only files with names that match *patterns*, according to the rules of Pattern Matching Notation (see *regexp*(5)), are selected. In addition, a leading `!` within a pattern indicates that only those names should be selected that do *not* match the remainder of the pattern. Multiple *patterns* can be specified. If no *patterns* are specified, the default for *patterns* is `*` (that is, select all files). The extracted files are conditionally created and copied into the current directory tree, based upon the options described below. The permissions of the files are those of the previous `-o` operation.

`-I` Extract, or copy in, files in the same way as for `ftio -icdmv` when no modifiers are used with the `-I`. However, if the file `.ftiorc` exists in the user's home

	directory, `ftio` opens this file, and scans for lines preceded by `I=`. Options defined on matching lines are passed to `ftio` as if they had been passed in the original command. See EXAMPLES section.
`-g`	Read the file list on *tapedev*. If *patterns* is specified, only file names that match are printed. Note that file names are always preceded by the volume that `ftio` expected the file to be on when the file list was created; thus only the last volume is valid in this respect.
`-B` *blksize*	Specify the size (in bytes) of blocks written to tape. This number can end with `k`, which specifies multiplication by 1024. The use of larger blocks generally improves performance and tape usage. The maximum allowable block size is limited by the tape drive used. A default of 16 384 bytes is set because this is the maximum block size on most Hewlett-Packard tape drives.
`-D` *type*	Recursively descend a directory only if the file system to which it belongs is of type *type*, where *type* is either `hfs` or `nfs`.
`-F` *ignorenames*	Arguments following `-F` specify *patterns* that should not be copied to the tape. The same rules apply for *ignorenames* as do for *patterns*, see the previous description for `ftio -i`.
`-K` *comment*	Specify a comment to be placed in the `ftio` tape header.
`-L` *filelist*	If *pathnames* is specified, perform the file search and generate a list of files to back up prior to actually commencing the backup. This list is then appended to the tape header of each tape in the backup as a list of files that `ftio` attempted to fit onto this tape. The last tape in the backup contains a catalog of where the files are in the archive set. If *pathnames* is not specified, the file list is taken from standard input before the backup begins. *filelist* specifies the output file. In addition to generating file lists, the `-L` option implements tape checkpointing, allowing the backup to restart from a write failure on bad media.
`-M`	Do not generate or expect tape headers, and change the default block size to 5120 bytes. This provides full compatibility with `cpio` (see the cpio Compatibility section below).
`-N` *datefile*	Only files that are newer than the file specified in *datefile* are copied to tape.
`-R`	Automatically resynchronize when `ftio` goes out of phase. This is useful when restoring from a multi-tape backup on tapes other than the first. Default behavior is that `ftio` asks the user if resyncing is required.
`-S` *script*	Specify a command to be invoked every time a tape is completed in a multi-tape backup. The command is invoked by the Bourne shell (see *sh-bourne*(1) with the following arguments: *script tape_no user_name*. *script* is the string argument *script* specified with the `-S` option. *tape_no* is the number of the tape required, and *user_name* is the user who invoked `ftio`. Typically, the string *script* specifies a shell script which is used to notify the user that a tape change is required.
`-T` *tty*	Specify alternate to `/dev/tty`. Normally `/dev/tty` is opened by `ftio` when terminal interaction is required.
`-Z` *nobufs*	Specify the number of *blksize* chunks of memory to use as buffer space between the two processes, where *blksize* is the size of blocks written to the tape. The use of more chunks is usually better, but a point is reached where no improvement is gained, and in fact performance may deteriorate as buffer space is swapped out of main memory. A default value of 16 is set for *nobufs*, but the use of 32 or 64 may improve performance if your system is not heavily loaded. Best results are obtained when backups are performed with the system in single-user mode (see *shutdown*(1M)).

Modifiers

The following modifiers can be used with certain options as indicated in the SYNOPSIS:

- `a` After files are copied out to tape, reset the access time to appear as though the file was not accessed by `ftio`.
- `c` Write header information in ASCII character form for portability.
- `d` When restoring files, create directories as needed.
- `f` Copy in all files except those that match *patterns*.
- `h` Archive the files to which symbolic links point as if they were normal files or directories. Normally, `ftio` archives the link itself.
- `m` Retain previous file modification time and ownership of file. Restoring modification time does not apply to directories that are being restored.
- `p` At the end of the backup, print the number of blocks transferred, the total time taken (excluding tape rewind and reel-change time), and the effective transfer rate calculated from these figures. These values are printed at the end of each tape if `p` is given twice.
- `t` Print only a table of contents of the input. No files are created, read, or copied.
- `u` Copy unconditionally (normally, an older file does not replace a newer file with the same name).
- `v` Be verbose. Print a list of file names as well as tape headers. When used with the `t` modifier, the table of contents looks the same as the output of the `ls -l` (ell) command (see *ls*(1)).
- `x` Save or restore device special files. `ftio` uses *mknod*(2) to recreate these files during a restore operation. Thus, this modifier is restricted to users with appropriate privileges. This is intended for intrasystem (backup) use. Restoring device files onto a different system can be very dangerous.
- `A` If copying from tape (`-i` or `-I` option), print all file names found on the archive, noting which files have been restored. This is useful when the user restores selected files, but wants to know which (if any) files are on the tape.

  If copying to tape (`-o` or `-O` option), suppress warning messages regarding optional access control list entries. *ftio*(1) does not back up optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file that has optional access control list entries.
- `E` When archiving, store all files having absolute path names (pathname starts with `/`) with a path name that is relative to the root directory (in other words, remove the leading `/` from all files whose name starts with `/`). On restoration, any files in the archive that had an absolute path name before archiving (the leading `/` was removed from the path name) are restored relative to the current directory.
- `L` Same as the `-L` option, except that the file list is left in the current directory as the file `ftio.list`, instead of the file named in *filelist*.
- `H` Search hidden subdirectories (context-dependent files also called CDFs). Normally, only the CDF element matching the current context is archived, without expanding the path name to show the actual element. For more information on CDFs see *cdf*(4).
- `P` On restoration, use `prealloc()` to pre-allocate disk space for the file (see *prealloc*(2)). This vastly improves the localization of file fragments.

When end-of-tape is reached, `ftio` invokes *script* if the `-S` option was specified, rewinds the current tape, then asks the user to mount the next tape.

To pass one or more metacharacters to `ftio` without having the shell expand them, protect them either by preceding each of them with a backslash (as in `/usr\*`), or enclosing them in protective quotes (as in `'/usr*'`).

cpio Compatibility

`ftio` uses the same archive format as `cpio`. However, by default `ftio` creates tape headers and uses a tape block size of 16 Kbytes. `cpio` by default uses 512-byte blocks. When used with the `-B` option, `cpio` uses 5120 byte blocks. To achieve full compatibility with `cpio` in either input or output mode, the user should specify the **M** modifier. `ftio -oM` creates a single- or multi-tape archive that has no tape headers, and, by default, the same block size as `cpio -[o|i]B`. An archive created by a `cpio -oB` command can be restored using `ftio -iM`. If the **M** modifier of `ftio` is combined with a `-B 512` block-size specification, full compatibility with `cpio -[o|i]` (no **-B**) is achieved.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating pattern matching notation for file name generation.

`LC_CTYPE` determines the characters matched by character class expressions in pattern matching notation.

`LC_TIME` determines the format and contents of date and time strings.

LANG determines the language in which messages are displayed.

If `LC_COLLATE`, `LC_CTYPE`, or `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5))*isusedinstead*of **LANG**. If any internationalization variable contains an invalid setting, `ftio` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Copy the entire contents of the file system (including special files) onto tape drive `/dev/rmt/0h`:

```
ftio -ox /dev/rmt/0h /
```

Restore all the files on `/dev/rmt/0h`, relative to the current directory:

```
ftio -idxE /dev/rmt/0h
```

List the contents of a backup set created using `ftio -o`. Note that use of the `v` modifier gives a more detailed listing, and displays the contents of tape headers.

```
ftio -itv /dev/rmt/0h
```

Show how to use the `.ftiorc` file:

Assume a `.ftiorc` file exists in the user's home directory and contains the following:

```
# Example .ftiorc file.
I= cdmuvEpp -B 16k -S /usr/local/bin/ftio.change
O= cavEpp -Z 8 -B 16k -S /usr/local/bin/ftio.change
```

Invoked `ftio` with the following command line to back up the user's home directory and the system binary directory:

```
ftio -O /dev/rmt/0h /user/my_home /bin
```

Specifying the `-O` option causes `ftio` to check the `.ftiorc` file for additional options. In this case, character headers are generated, access times are reset, a listing of the files copied are printed to standard output, all file names are copied to `/dev/rmt/0h` with path names relative to `/`, performance data is printed when the backup is complete (and at every tape change), and, if the backup goes beyond one media the script, `/usr/local/bin/ftio.change` is invoked by `ftio` after each media is completed.

WARNINGS

`ftio` uses System V shared memory and semaphores for its operation. The resources committed to these functions are not automatically freed by the system when the process terminates. `ftio` does this only when it terminates normally, or when it terminates after receiving one the following signals: `SIGHUP`, `SIGINT`, `SIGTERM`. Any other signal is handled in the default manner described by *signal*(2). Note that the behavior for `SIGKILL` is to terminate the process without delay. Thus, if `ftio` receives a `SIGKILL` signal (as might be produced by the indiscriminate use of `kill -9` (see *kill*(1)), system resources used for shared memory and semaphores are not returned to the system. If it becomes necessary to terminate an invocation of `ftio`, use `kill -15`. Current system usage of shared memory and semaphores can be checked using the `ipcs` command (see *ipcs*(1)). Committed resources can be removed using `ipcrm` (see *ipcrm*(1)).

AUTHOR

`ftio` was developed by HP.

SEE ALSO

cpio(1), find(1), ipcs(1), ipcrm(1), kill(1), ls(1), rmt(1M), mknod(2), prealloc(2), signal(2), uname(2), cdf(4), acl(5), environ(5), lang(5), regexp(5), mt(7).

NAME

ftp - file transfer program

SYNOPSIS

`ftp` [`-g`] [`-i`] [`-n`] [`-v`] [*server-host*]

DESCRIPTION

`ftp` is a user interface to the File Transfer Protocol. `ftp` copies files over a network connection between the local "client" host and a remote "server" host. `ftp` runs on the client host.

Options

The `ftp` command supports the following options:

- `-g` Disable file name "globbing"; see the `glob` command, below. By default, when this option is not specified, globbing is enabled.
- `-i` Disable interactive prompting by multiple-file commands; see the `prompt` command, below. By default, when this option is not specified, prompting is enabled.
- `-n` Disable "auto-login"; see the `open` command, below. By default, when this option is not specified, auto-login is enabled.
- `-v` Enable verbose output; see the `verbose` command, below. If this option is not specified, `ftp` displays verbose output only if the standard input is associated with a terminal.

The name of the server host that `ftp` communicates with can be specified on the command line. If the server host is specified, `ftp` immediately opens a connection to the server host; see the `open` command, below. Otherwise, `ftp` waits for commands from the user.

File Transfer Protocol specifies file transfer parameters for *type*, *mode*, *form*, and *struct*. `ftp` supports the ASCII, `binary`, and `tenex` File Transfer Protocol *types*. `ascii` is the default FTP *type*. `ftp` supports only the default values for the file transfer parameters *mode* which defaults to `stream`, *form* which defaults to `non-print`, and *struct* which defaults to `file`.

COMMANDS

`ftp` supports the following commands. Command arguments with embedded spaces must be enclosed in quotes (for example, "argument with embedded spaces").

`!`[*command* [*args*]]
: Invoke a shell on the local host. The `SHELL` environment variable specifies which shell program to invoke. `ftp` invokes `/bin/sh` if `SHELL` is undefined. If *command* is specified, the shell executes it and returns to `ftp`. Otherwise, an interactive shell is invoked. When the shell terminates, it returns to `ftp`.

`$` *macro-name* [*args*]
: Execute the macro *macro-name* that was defined with the `macdef` command. Arguments are passed to the macro unglobbed.

`account` [*passwd*]
: Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user is prompted for an account password in a non-echoing input mode.

`append` *local-file* [*remote-file*]
: Copy *local-file* to the end of *remote-file*. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any *ntrans* or *nmap* setting.

`ascii`
: Set the file transfer *type* to network ASCII. This is the default type.

`bell`
: Sound a bell after each file transfer completes.

`binary`
: Set the file transfer *type* to `binary`.

`bye` Close the connection to the server host if a connection was open, and exit. Typing an end-of-file (EOF) character also terminates and exits the session.

`case`
: Toggle remote computer file name case mapping during `mget` commands. When `case` is on (the default is off), remote computer file names with all letters in uppercase are written in the local directory with the letters mapped to lowercase.

`cd` *remote-directory*
: Set the working directory on the server host to *remote-directory*.

`cdup`
: Set the working directory on the server host to the parent of the current remote working directory.

`chmod` *mode file-name*
: Change the permission modes the file *file-name* on the remote system to *mode*.

`close`
: Terminate the connection to the server host. The `close` command does not exit `ftp`. Any defined macros are erased.

`cr` Toggle carriage return stripping during `ascii` type file retrieval. Records are denoted by a carriage-return/line-feed sequence during `ascii` type file transfer. When `cr` is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single line-feed record delimiter. Records on non-UNIX remote systems may contain single line-feeds; when an `ascii` type transfer is made, these line-feeds can be distinguished from a record delimiter only when `cr` is off.

`delete` *remote-file*
: Delete *remote-file*. The *remote-file* can be an empty directory. No globbing is done.

`dir` [*remote-directory*] [*local-file*]
: Write a *remote-directory* listing to standard output or optionally to *local-file*. If neither *remote-directory* nor *local-file* is specified, list the remote working directory to standard output. If interactive prompting is on, `ftp` prompts the user to verify that the last argument is indeed the target file for `dir` output. Globbing characters are always expanded.

`disconnect`
: A synonym for `close`.

`form` *format*
: Set the file transfer *form* to *format*. The only supported format is `non-print`

`get` *remote-file* [*local-file*]
: Copy *remote-file* to *local-file*. If *local-file* is unspecified, `ftp` uses the specified *remote-file* name as the *local-file* name, subject to alteration by the current *case*, *ntrans*, and *nmap* settings.

`glob`
: Toggle file name globbing. When file name globbing is enabled, `ftp` expands *csh*(1) metacharacters in file and directory names. These characters are `*`, `?`, `[`, `]`, `~`, `{`, and `}`. The server host expands remote file and directory names. Globbing metacharacters are always expanded for the `ls` and `dir` commands. If globbing is enabled, metacharacters are also expanded for the multiple-file commands `mdelete`, `mdir`, `mget`, `mls`, and `mput`.

`hash`
: Toggle printing of a hash-sign (#) for each 1024 bytes transferred.

`help` [*command*]
: Print an informative message about the `ftp` command called *ftp-command*. If *ftp-command* is unspecified, print a list of all `ftp` commands.

`idle` [*seconds*]
: Set the inactivity timer on the remote server to *seconds* seconds. If *seconds* is omitted, `ftp` prints the current inactivity timer.

`lcd` [*local-directory*]
: Set the local working directory to *local-directory*. If *local-directory* is unspecified, set the local working directory to the user's local home directory.

`ls` [*remote-directory*] [*local-file*]
: Write a listing of *remote-directory* to *local-file*. The listing includes any system-dependent information

that the server chooses to include; for example, most UNIX systems produce output from the command `ls -l` (see also `nlist`). If neither *remote-directory* nor *local-file* is specified, list the remote working directory. If globbing is enabled, globbing metacharacters are expanded.

`macdef` *macro-name*
: Define a macro. Subsequent lines are stored as the macro *macro-name*; an empty input line terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a `close` command is executed. The macro processor interprets **$** and **** as special characters. A **$** followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A **$** followed by an *i* signals that macro processor that the executing macro is to be looped. On the first pass **$***i* is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A **** followed by any character is replaced by that character. Use the **** to prevent special treatment of the **$**.

`mdelete` [*remote-files*]
: Delete *remote-files*. If globbing is enabled, globbing metacharacters are expanded.

`mdir` *remote-files local-file*
: Write a listing of *remote-files* to *local-file*. If globbing is enabled, globbing metacharacters are expanded. If interactive prompting is on, `ftp` prompts the user to verify that the last argument is indeed the target local file for `mdir` output.

`mget` *remote-files*
: Copy *remote-files* to the local system. If globbing is enabled, globbing metacharacters are expanded. The resulting local file names are processed according to *case*, *ntrans*, and *nmap* settings.

`mkdir` *directory-name*
: Create remote *directory-name*.

`mls` *remote-files local-file*
: Write an abbreviated listing of *remote-files* to *local-file*. If globbing is enabled, globbing metacharacters are expanded. If interactive prompting is *on*, `ftp` prompts the user to verify that the last argument is indeed the target local file for `mls` output.

`mode` [*mode-name*]
: Set the FTP file transfer *mode* to *mode-name*. The only supported mode is `stream`.

`modtime` *remote-file*
: Show the last modification time of *remote-file*.

`mput` *local-files*
: Copy *local-files* from the local system to the remote system. The remote files have the same name as the local files processed according to *ntrans* and *nmap* settings. If globbing is enabled, globbing characters are expanded.

`newer` *file-name*
: Get the file only if the modification time of the remote file is more recent that the file on the current system. If the file does not exist on the current system, the remote file is considered *newer*. Otherwise, this command is identical to `get`.

`nlist` [*remote-directory*] [*local-file*]
: Write an abbreviated listing of *remote-directory* to *local-file*. If *remote-directory* is left unspecified, the current working directory is used. If interactive prompting is on, `ftp` prompts the user to verify that the last argument is indeed the target local file for `nlist` output.

`nmap` [*inpattern outpattern*]
: Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during `mput` commands and `put` commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during `mget` commands and `get` commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *inpattern* is a template for incoming filenames (which may have already been processed according to the `ntrans` and `case` settings). Variable templating is accomplished

by including the sequences **$1**, **$2**, ..., **$9** in *inpattern*. Use **** to prevent this special treatment of the **$** character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values. For example, given *inpattern* **$1.$2** and the remote file name **mydata.data**, **$1** would have the value **mydata**, and **$2** would have the value **data**. The *outpattern* determines the resulting mapped filename. The sequences **$1**, **$2**, ..., **$9** are replaced by any value resulting from the *inpattern* template. The sequence **$0** is replaced by the original filename. Additionally, the sequence **[***seq1* **,** *seq2* **]** is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command **nmap $1.$2.$3 [$1,$2].[$2,file]** would yield the output filename **myfile.data** for input filenames **myfile.data** and **myfile.data.old**, **myfile.file** for the input filename **myfile**, and **myfile.myfile** for the input filename **.myfile**. Spaces can be included in *outpattern*, as in the example: **nmap $1 | sed "s/ *$//" > $1** . Use the **** character to prevent special treatment of the **$**, **[**, **]**, and **,** characters.

ntrans | 0[*inchars* [*outchars*]]
: Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

open | 0*server-host* [*port-number*]
: Establish a connection to *server-host*, using *port-number* (if specified). If *auto-login* is enabled, **ftp** attempts to log into the server host.

prompt
: Toggle interactive prompting. By default, **ftp** prompts the user for a yes or no response for each output file during multiple-file commands. If interactive prompting is disabled, **ftp** performs the command for all specified files.

proxy *ftp-command*
: Execute an **ftp** command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command **proxy ?** to see other FTP commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** does not define new macros during the auto-login process, **close** does not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the FTP protocol **PASV** command by the server on the secondary control connection.

put *local-file* [*remote-file*]
: Copy *local-file* to *remote-file*. If *remote-file* is unspecified, **ftp** assigns the *local-file* name, processed according to any *ntrans* or *nmap* settings, to the *remote-file* name.

pwd Write the name of the remote working directory to *stdout*.

quit
: A synonym for **bye**.

quote *arguments*
: Send *arguments*, verbatim, to the server host. See *ftpd*(1M).

recv *remote-file* [*local-file*]
: A synonym for get.

reget *remote-file* [*local-file*]
: **reget** acts like **get**, except that if *local-file* exists and is smaller than *remote-file*, *local-file* is presumed to be a partially transferred copy of *remote-file* and the transfer is continued from the

apparent point of failure. This command is useful when transferring very large files over networks that tend to drop connections.

`rhelp` [*command-name*]
: Request help from the server host. If *command-name* is specified, supply it to the server. See *ftpd*(1M).

`rstatus` [*file-name*]
: With no arguments, show status of remote machine. If *file-name* is specified, show status of *file-name* on remote machine.

`rename` *remote-from remote-to*
: Rename *remote-from*, which can be either a file or a directory, to *remote-to*.

`reset`
: Clear reply queue. This command re-synchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

`restart` *marker*
: Restart the immediately following `get` or `put` at the indicated *marker*. On UNIX systems, marker is usually a byte offset into the file.

`rmdir` *remote-directory*
: Delete *remote-directory*. *remote-directory* must be an empty directory.

`runique`
: Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a `get` or `mget` command, a `.1` is appended to the name. If the resulting name matches another existing file, a `.2` is appended to the original name. If this process continues up to `.99`, an error message is printed, and the transfer does not take place. `ftp` reports the unique filename. Note that `runique` does not affect local files generated from a shell command (see below). The default value is `off`.

`send` *local-file* [*remote-file*]
: A synonym for `put`.

`sendport`
: Toggle the use of `PORT` commands. By default, `ftp` attempts to use a `PORT` command when establishing a connection for each data transfer. If the `PORT` command fails, `ftp` uses the default data port. When the use of `PORT` commands is disabled, `ftp` makes no attempt to use `PORT` commands for each data transfer. This is useful for certain FTP implementations that ignore `PORT` commands but (incorrectly) indicate that they've been accepted. See *ftpd*(1M). Turning `sendport` off may cause delays in the execution of commands.

`site` *arguments*
: Send *arguments*, verbatim, to the server host as a `SITE` command. See *ftpd*(1M).

`size` *remote-file*
: Show the size of *remote-file*.

`status`
: Show the current status of `ftp`.

`struct` [*struct-name*]
: Set the FTP file transfer *struct* to *struct-name*. The only supported *struct* is `file`.

`sunique`
: Toggle storing of files on remote machine under unique file names. The remote server reports the unique name. By default, `sunique` is `off`.

`system`
: Show the type of operating system running on the remote machine.

`tenex`
: Set the FTP file transfer *type* to `tenex`.

`type` [*type-name*]
: Set the FTP file transfer *type* to *type-name*. If *type-name* is unspecified, write the current *type* to *stdout*. **`Ascii`**, **`binary`**, and **`tenex`** are the *type*s currently supported.

`umask` [*newmask*]
: Set the default umask on the remote server to *newmask*. If *newmask* is ommitted, the current umask is printed.

`user` *user-name* [*password*] [*account*]
: Log into the server host on the current connection, which must already be open. A **`.netrc`** file in the user's local home directory can provide the *user-name*, *password*, and optionally the *account*; see *netrc*(4). Otherwise `ftp` prompts the user for this information. The HP-UX FTP server does not require an *account*. For security reasons, `ftp` always requires a password. It does not log into remote accounts that do not have a password.

`verbose`
: Toggle verbose output. If verbose output is enabled, `ftp` displays responses from the server host, and when a file transfer completes it reports statistics regarding the efficiency of the transfer.

`?` [*command*]
: A synonym for the `help` command. Prints the `help` information for the specified *command*.

Aborting A File Transfer

To abort a file transfer, use the terminal interrupt key (usually **Ctrl-C**). Sending transfers are halted immediately. `ftp` halts incoming (receive) transfers by first sending a FTP protocol **ABOR** command to the remote server, then discarding any further received data. The speed at which this is accomplished depends upon the remote server's support for **ABOR** processing. If the remote server does not support the **ABOR** command, an `ftp>` prompt does not appear until the remote server completes sending the requested file.

The terminal interrupt key sequence is ignored while `ftp` while awaits a reply from the remote server. A long delay in this mode may result from the **ABOR** processing described above, or from unexpected behavior by the remote server, including violations of the FTP protocol. If the delay results from unexpected remote server behavior, the local `ftp` program must be killed manually.

File Naming Conventions

Files specified as arguments to `ftp` commands are processed according to the following rules.

- If the file name `-` is specified, `ftp` uses the standard input (for reading) or standard output (for writing).
- If the first character of the file name is `|`, `ftp` interprets the remainder of the argument as a shell command. `ftp` forks a shell, using `popen()` (see *popen*(3S) with the supplied argument, and reads (writes) from standard output (standard input). If the shell command includes spaces, the argument must be quoted, as in:

  ```
  "| ls -lt".
  ```

  A particularly useful example of this mechanism is:

  ```
  "| dir . | more".
  ```

- Otherwise, if globbing is enabled, `ftp` expands local file names according to the rules used by the C shell (see *csh*(1)); see the `glob` command, below. If the `ftp` command expects a single local file (e.g. `put`), only the first filename generated by the globbing operation is used.
- For `mget` commands and `get` commands with unspecified local file names, the local filename is named the same as the remote filename, which may be altered by a **`case`**, **`ntrans`**, or **`nmap`** setting. The resulting filename may then be altered if **`runique`** is on.
- For `mput` commands and `put` commands with unspecified remote file names, the remote filename is named the same as the local filename, which may be altered by a **`ntrans`** or **`nmap`** setting. The resulting filename may then be altered by the remote server if **`sunique`** is on.

WARNINGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in FTP clients and servers based on the 4.2BSD implementation (specifically, any `ftp` or `ftpd` version released prior to HP-UX 8.0) has been corrected. This correction may result in incorrect transfers of binary files when using the `ascii` type. Avoid this problem by using the `binary` transfer type.

AUTHOR

`ftp` was developed by the University of California, Berkeley.

SEE ALSO

csh(1), rcp(1), ftpd(1M), netrc(4), ftpusers(4), hosts(4).

NAME

gencat - generate a formatted message catalog file

SYNOPSIS

`gencat` [`-1`] *catfile msgfile* ...

DESCRIPTION

Message catalogs allow a program to process input and produce output according to local customs and languages. For details, see *Native Language Support Users Guide*.

`gencat` merges each message source *msgfile* into a formatted message catalog *catfile* that can be accessed by `catgetmsg()` or `catgets()` (see *catgetmsg*(3C) and *catgets*(3C)). If *catfile* does not exist, it is created. If *catfile* exists, its messages are included in the new *catfile*. If set and message numbers collide, the new message text in *file* replaces the old message text in *catfile*. A *msgfile* consists of message, directive and comment lines (all without leading spaces or tabs) described below. Except as noted, fields are separated by one or more space or tab characters.

`$set` *s* [*comment*]
: A `$set` directive specifies the set *s*, of the messages that follow until the next `$set` or end-of-file appears. The set number *s* is an unsigned integer in the range 1 through **NL_SETMAX**. Any string following the set number is treated as a comment. If a `$set` directive is not specified, messages are put in the default set **NL_SETD**.

  Set numbers must be in ascending order within a *msgfile* but need not be contiguous.

`$delset` *s* [*comment*]
: A `$delset` directive deletes the message set identified by the set number *s*, from an existing message catalog. Any string following the set number is treated as a comment.

m message_text
: A message line specifies a message number *m*, and associated message text. The message number *m* is an unsigned integer in the range 1 through **NL_MSGMAX**. The *message_text* is a C string, including spaces, tabs and \ (backslash) escapes, but by default without surrounding quotes (see `$quote` directive below). The message number *m* is separated from the *message_text* by a single space or tab character. The *message_text* begins with the first character following the separator and ends at newline. Extra spaces or tabs (including any trailing spaces or tabs) are considered part of the *message_text*.

  The *message_text* of a message line is stored in *catfile* with message number *m* and set number *s* specified by the most recent `$set` directive.

  Message numbers must be in ascending order within a set but need not be contiguous.

  Note that the space or tab separator distinguishes insertion of a null message from deletion of a message. If a message line has a number and separator but no text, the message number and an associated null message string are stored in *catfile*. If a message line has a number but neither separator nor text, the message number and its associated message text are deleted from *catfile*.

`-1`
: If the `-1` option is specified, the length of *message_text* must be no more than **MAX_BUFLEN** – 1 bytes. If the `-1` option is not specified, the length of *message_text* must be no more than **NL_TEXTMAX** bytes. See *catgetmsg*(3C), *catgets*(3C), *catread*(3C), and *getmsg*(3C) for message length limits imposed by these routines.

`$quote` [*q comment*]
: A `$quote` directive specifies a quote character *q*, used to surround *message_text* and make leading and trailing space visible in a message line. Any string following the specified quote character *q* is treated as a comment. By default, or if a quote character *q* not is supplied, quoting of *message_text* is not recognized.

`$` *comment*
: A `$` followed by a space or tab is treated as a comment and can appear anywhere in a file. A line consisting of zero or more spaces or tabs is treated as a comment line.

NL_TEXTMAX, **NL_SETMAX**, and **NL_MSGMAX** are defined in <`limits.h`>. **NL_SETD** is defined in <`nl_types.h`>. **MAX_BUFLEN** is defined in <`msgcat.h`>.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of messages as single- and/or multi-byte characters.

Messages are issued in LANG if it is set to a valid language and **LANG** messages are available. Otherwise "C" locale messages will be issued.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **gencat** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

The **$quote** directive is not currently supported on the HP MPE and RTE operating systems.

AUTHOR

gencat was developed by HP and the X/Open Company, Ltd.

SEE ALSO

dumpmsg(1), findmsg(1), insertmsg(1), catgets(3C), catopen(3C).

Native Language Support Users Guide.

STANDARDS CONFORMANCE

gencat: XPG2, XPG3

NAME

get - get a version of an SCCS file

SYNOPSIS

get [**-r** *SID*] [**-c** *cutoff*] *[***-e***] [***-b***]* [**-i** *list*] [**-x** *list*] [**-k**] [**-l**[**p**]] [**-p**] [**-s**] [**-m**] [**-n**] [**-g**] [**-t**] [**-w** *string*] [**-a** *seq-number*] *file* ...

DESCRIPTION

get generates an ASCII text file from each named SCCS file according to the specifications given by its option arguments, which begin with **-**. The arguments can be specified in any order, but all option arguments apply to all named SCCS files. If a directory is named, **get** behaves as if each file in the directory was specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a file name of **-** is given, the standard input is read and each line of the standard input is assumed to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the **s.** prefix (see also FILES below).

Options

Explanation of the option arguments below is based on processing only one SCCS file. When processing multiple SCCS files, the effects of any option argument applies independently to each named file.

-r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by **delta** if the **-e** option is also used), as a function of the SID specified (see *delta*(1)).

-c*cutoff* *cutoff* date-time, in the form:

YY [*MM* [*DD* [*HH* [*MM* [*SS*]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **-c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters can separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: Note that this implies that one can use the **%E%** and **%U%** identification keywords (see below) for nested **get**s within, for example, a **send** command (see *send*(1)):

```
~!get
```

-e Indicates that the **get** is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of **delta**. The **-e** option used in a **get** for a particular version (SID) of the SCCS file prevents further **get**s for editing on the same SID until **delta** is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get -e** for different SIDs is always allowed. Note, however, that only one user is permitted to do a concurrent **get -e** (see *admin*(1)).

If the *g-file* generated by **get** with an **-e** option is accidentally ruined in the process of editing it, it can be regenerated by re-executing the **get** command with the **-k** option in place of the **-e** option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **-e** option is used.

-b Used with the **-e** option to indicate that the new delta should have an SID in a new branch as shown in Table 1. This option is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)

Note: A branch *delta* can always be created from a non-leaf *delta*.

-i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
list  ::= range | list, range
range ::= SID | SID - SID
```

SID, the SCCS Identification of a delta, can be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1. See WARNINGS below.

-x*list*
A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **-i** option for the *list* format.

-k
Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** option is implied by the **-e** option.

-l[p]
Causes a delta summary to be written into an *l-file*. If **-lp** is used, an *l-file* is not created; the delta summary is written on the standard output instead. See FILES for the format of the *l-file*. The user must have s-file read permission in order to use the **-l** option.

-p
Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output that normally goes to the standard output goes to file descriptor 2 (standard error) instead, unless the **-s** option is used, in which case it disappears.

-s
Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m
Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n
Causes each generated text line to be preceded with the **%M%** identification keyword value (see below). The format is: **%M%** value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** options are used, the format is: **%M%** value, followed by a horizontal tab, followed by the **-m** option-generated format.

-g
Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

-t
Used to access the most recently created ("top") delta in a given release (e.g., **-r1**), or release and level (e.g., **-r1.2**).

-w *string*
Substitute *string* for all occurrences of **@ @%M%** when **get**ting the file.

-a*seq-number*
The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(4)). This option is used by the **comb** command (see *comb*(1)); it is not a generally useful option, and should be avoided. If both the **-r** and **-a** options are specified, the **-a** option is used. Care should be taken when using the **-a** option in conjunction with the **-e** option, because the SID of the delta to be created may not be what one expects. The **-r** option can be used with the **-a** and **-e** options to control the naming of the SID of the delta to be created.

For each file processed, **get** responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** option is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** option is used included deltas are listed following the notation "Included"; if the **-x** option is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	−b Option Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	-	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

# Successor.

† The `-b` option is effective only if the `b` flag (see *admin*(1)) is present in the file. An entry of - means "irrelevant".

‡ This case applies if the `d` (default SID) flag is *not* present in the file. If the `d` flag *is* present in the file, then the SID obtained from the `d` flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

Identification Keywords

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords can be used in the text stored in an SCCS file:

Keyword	Value
%M%	Module name: either the value of the `m` flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading `s.` removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).

%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the `t` flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the `q` flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as **this should not have happened** type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string recognizable by **what** (see *what*(1)).
%W%	A shorthand notation for constructing *what*(1) strings for HP-UX system program files. **%W%=%Z%%M%***horizontal-tab***%I%**

%A%

Another shorthand notation for constructing *what*(1) strings for non-HP-UX system program files.
%A% = %Z%%Y% %M% %I%%Z%

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **get** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

FILES

Several auxiliary files can be created by **get**. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **-p** option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the **get**. It is owned by the real user. If the **-k** option is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **-l** option is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

1. A blank character if the delta was applied;
   ***** otherwise.
2. A blank character if the delta was applied or was not applied and ignored;
   ***** if the delta was not applied and was not ignored.
3. A code indicating a “special” reason why the delta was or was not applied:

   I: Included.
   X: Excluded.
   C: Cut off (by a **-c** option).

4. Blank.

5 SCCS identification (SID).

6. Tab character.
7. Creation date and time (in the form YY/MM/DD HH:MM:SS).
8. Blank.
9. Login name of person who created *delta*.

   The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a **get** with an **-e** option along to *delta*. Its contents are also used to prevent a subsequent execution of **get** with an **-e** option for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the **get** was executed, followed by a blank and the **-i** option argument if it was present, followed by a blank and the **-x** option argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., **get**) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of **get**. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file can be named when the **-e** option is used.

Unexpected results occur when using the **-i** option to merge changes into sections of a file that have been (perhaps inadvertently) deleted and subsequently re-inserted into a file.

An *l-file* cannot be generated when **-g** is used. In other words, **-g -l** does not work.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(4).

SCCS User's Guide, in *Programming on HP-UX*.

STANDARDS CONFORMANCE

get: SVID2, XPG2, XPG3

NAME

getaccess - list access rights to file(s)

SYNOPSIS

`getaccess` [`-u` *user*] [`-g` *user*] *group* [,*group*] ...] [`-n`] *file* ...
`getaccess -r` [`-n`] *file* ...

DESCRIPTION

`getaccess` lists for the specified files the effective access rights of the caller (that is, for their effective user ID, effective group ID, and supplementary groups list). By default, the command prints a symbolic representation of the user's access rights to the named file: `r` or `-` for read/no read, `w` or `-` for write/no write, and `x` or `-` for execute/no execute (for directories, search/no search), followed by the file name.

Options

`getaccess` recognizes the following options and command-line arguments:

`-u` *user* List access for the given user instead of the caller. A *user* can be a known user name, a valid ID number, or @, representing the file's owner ID. If information about more than one file is requested, the value of @ can differ for each.

This option sets the user ID only. The access check is made with the caller's effective group ID and supplementary group IDs unless `-g` is also specified.

`-g` *group* [,*group*] ...]
List access for the given group(s) instead of the caller's effective group ID and supplementary groups list. A *group* can be a known group name, a valid ID number, or @, representing the file's group ID. If information about more than one file is requested, the value of @ can differ for each.

`-r` List access using the caller's real user ID, group ID, and supplementary groups list, instead of effective ID values.

`-n` List access rights numerically (octal digits `0..7` instead of `rwx`) for each file requested. The bit values `R_OK`, `W_OK`, and `X_OK` are defined in the file `<unistd.h>`.

Checking access using access control lists is described in *acl*(5).

In addition, the write bit is cleared for files on read-only file systems or shared-text programs being executed. The execute bit is not turned off for shared-text programs open for writing because it is not possible to ascertain whether a file open for writing is a shared-text program.

Processes with appropriate privileges have read and write access to all files. However, write access is denied for files on read-only file systems or shared-text programs being executed. Execute access is allowed if and only if the file is not a regular file or the execute bit is set in any of the file's ACL entries.

To use `getaccess` successfully, the caller must have search access in every directory component of the path name of the *file*. `getaccess` verifies search access first by using the caller's effective IDs, regardless of the user and group IDs specified. This is distinct from the case in which the caller can search the path but the user for whom access is being checked does not have access to the file.

Note: a file name argument of `-` has no special meaning (such as standard input) to `getaccess`.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, `getaccess` behaves as if all internationalization variables are set to "C". See *environ*(5).

RETURN VALUE

`getaccess` returns one of the following values:

0 Successful completion.

1 `getaccess` was invoked incorrectly or encountered an unknown user or group name. An appropriate message is printed to standard error.

2. A file is nonexistent or unreachable (by the caller). `getaccess` prints an appropriate message to standard error, continues, then returns a value of 2 upon completion.

EXAMPLES

The following command prints the caller's access rights to *file1* using the file's group ID instead of the caller's effective group ID and groups list.

```
getaccess -g@ file1
```

Here's how to check access by user `ggd` in groups `red` and `19` to all files in the current directory, with access rights expressed as octal values.

```
getaccess -u ggd -g red,19 -n .* *
```

Here's how to list access rights for all files under `mydir`.

```
find mydir -print | sort | xargs getaccess
```

AUTHOR

`getaccess` was developed by HP.

FILES

```
/etc/passwd
/etc/group
```

SEE ALSO

chacl(1), lsacl(1), getaccess(2), glossary(9).

NAME

getconf - get system configuration values

SYNOPSIS

`getconf` [*parameter_name*] [*pathname*]

DESCRIPTION

getconf provides a program interface to the *confstr*(3C), *pathconf*(2), and *sysconf*(2) libraries.

The *parameter_name* argument specifies the configuration value desired in `confstr()`, `pathconf()`, or `sysconf()`. For *parameter_name*, operand values listed in the table with `pathconf()` as the underlying function, the *pathname* operand must also be supplied; otherwise, the *pathname* operand must not be supplied.

RETURN VALUE

The error codes returned by getconf are:

0 Success. A value corresponding to the operand was returned;
1 One or more missing or extra operands;
2 Operand was not recognized.
3 Pathname could not be accessed.

EXAMPLES

Request the number of intervals per second:

```
getconf CLK_TCK
```

Request the maximum value of a file's link count:

```
getconf LINK_MAX /etc/passwd
```

Some other supported inquiries include:

ARG_MAX	_POSIX_CHILD_MAX	POSIX2_C_DEV
BC_BASE_MAX	_POSIX_JOB_CONTROL	POSIX2_EXPR_NEST_MAX
BC_DIM_MAX	_POSIX_NGROUPS_MAX	POSIX2_FORT_DEV
BC_SCALE_MAX	_POSIX_OPEN_MAX	POSIX2_FORT_RUN
BC_STRING_MAX	_POSIX_SAVED_IDS	POSIX2_LINE_MAX
CHILD_MAX	_POSIX_SSIZE_MAX	POSIX2_LOCALEDEF
CLK_TCK	_POSIX_STREAM_MAX	POSIX2_RE_DUP_MAX
COLL_WEIGHTS_MAX	_POSIX_TZNAME_MAX	POSIX2_SW_DEV
CS_PATH	_POSIX_VERSION	POSIX2_VERSION
EXPR_NEST_MAX	POSIX2_BC_BASE_MAX	RE_DUP_MAX
LINE_MAX	POSIX2_BC_DIM_MAX	SC_PASS_MAX
NGROUPS_MAX	POSIX2_BC_SCALE_MAX	SC_XOPEN_VERSION
OPEN_MAX	POSIX2_BC_STRING_MAX	STREAM_MAX
PATH	POSIX2_COLL_WEIGHTS_MAX	TZNAME_MAX
_POSIX_ARG_MAX	POSIX2_C_BIND	

Some other supported inquires, which require the second parameter:

LINK_MAX	PIPE_BUF	_POSIX_NAME_MAX
MAX_CANON	_POSIX_CHOWN_RESTRICTED	_POSIX_NO_TRUNC
MAX_INPUT	_POSIX_LINK_MAX	_POSIX_PATH_MAX
NAME_MAX	_POSIX_MAX_CANON	_POSIX_PIPE_BUF
PATH_MAX	_POSIX_MAX_INPUT	_POSIX_VDISABLE

AUTHOR

`getconf` was developed by HP and POSIX.

SEE ALSO

confstr(3C), pathconf(2), sysconf(2).

STANDARDS CONFORMANCE

`getconf`: POSIX.2

NAME

getcontext - display current context

SYNOPSIS

`getcontext`

DESCRIPTION

`getcontext` displays the context of the invoking process. The context is displayed as a list of words separated by spaces, and is used when matching context-dependent files. See *cdf*(4).

AUTHOR

`getcontext` was developed by HP.

SEE ALSO

getcontext(2), cdf(4), context(5).

NAME

getopt - parse command options

SYNOPSIS

`getopt` *optstring args*

DESCRIPTION

`getopt` is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *optstring* is a string of recognized option letters (see *getopt*(3C)). If a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space.

The positional parameters ($1 $2 ...) of the shell are reset so that each option is preceded by a - and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

`getopt` recognizes two hyphens (- -) to delimit the end of the options. If absent, `getopt` places `--` at the end of the options.

The most common use of `getopt` is in the shell's `set` command (see the example below) where `getopt` converts the command line to a more easily parsed form. `getopt` writes the modified command line to the standard output.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `getopt` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

DIAGNOSTICS

`getopt` prints an error message on the standard error when it encounters an option letter not included in *optstring*.

EXAMPLES

The following code fragment processes the arguments for a command that can take the options `a` or `b`, and the option `o` which requires an argument:

```
set -- 'getopt abo: $*'

if [ $? -ne 0 ]; then
    echo $USAGE
    exit 2
fi

while [ $# -gt 0 ]; do
    case $1 in
    -a | -b)
        FLAG=$1
        shift
        ;;
    -o)
        OARG=$2
        shift 2
        ;;
    esac
done
```

This code accepts any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

WARNINGS

`getopt` option arguments must not be null strings nor contain embedded blanks.

SEE ALSO

sh(1), getopt(3C).

NAME

getopts - parse utility (command) options

SYNOPSIS

`getopts` *optstring name* [*arg* ...]

DESCRIPTION

`getopts` is used to retrieve options and option-arguments from a list of parameters.

Each time it is invoked, `getopts` places the value of the next option in the shall variable specified by the `name` operand and the index of the next argument to be processed in the shell variable `OPTIND`. Whenever the shell is invoked, `OPTIND` is initialized to 1.

When the option requires an option-argument, `getopts` places it in the shell variable `OPTARG`. If no option was found, or if the option that was found does not have an option-argument, `OPTARG` is unset.

If an option character not contained in the *optstring* operand is found where an option character is expected, the shell variable specified by *name* is set to the question-mark (`?`) character. In this case, if the first character in *optstring* is a colon (`:`), the shell variable `OPTARG` is set to the option character found, but no output is written to standard error; otherwise, the shell variable `OPTARG` is unset and a diagnostic message is written to standard error. This condition is considered to be an error detected in the way arguments were presented to the invoking application, but is not an error in `getopts` processing.

If an option-argument is missing:

- If the first character of *optstring* is a colon, the shell variable specified by *name* is set to the colon character and the shell variable `OPTARG` is set to the option character found.
- Otherwise, the shell variable specified by *name* is set to the question-mark character, the shell variable `OPTARG` is unset, and a diagnostic message is written to the standard error. This condition is considered to be an error detected in the way arguments are presented to the invoking application, but is not an error in `getopts` processing; a diagnostic message is written as stated, but the exit status is zero.

When the end of options is encountered, `getopts` exits with a return value greater than zero. The shell variable `OPTIND` is set to the index of the first nonoption-argument, where the first `--` argument is considered to be an option argument if there are no other non-option arguments appearing before it, or the value `$* + 1` if there are no nonoption-arguments; the *name* variable is set to the question-mark character. Any of the following identifies the end of options: the special option `--`, finding an argument that does not begin with a `-`, or encountering an error.

The shell variables `OPTIND` and `OPTARG` are local to the caller of `getopts` and are not exported by default.

The shell variable specified by the *name* operand, `OPTIND`, and `OPTARG` affect the current shell execution environment.

Operands

The following operands are supported:

optstring	A string containing the option characters recognized by the utility invoking *getopts*.
name	The name of a shell variable that is set by `getopts` to the option character that was found.

`getopts` by default parses positional parameters passed to the invoking shell procedures. If *args* are given, they are parsed instead of the positional parameters.

EXTERNAL INFLUENCES

The following environment variable affects the execution of getopts utility:

`OPTIND`	Used by `getopts` as the index of the next argument to be processed.

EXAMPLES

Since `getopts` affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment such as one of the following:

```
(getopts abc value "$@")
nohup getopts ...
```

```
        find -exec getopts ...;
```

it does not affect the shell variables in the caller's environment.

Note that shell functions share `OPTIND` with the calling shell even though the positional parameters are changed. Functions that use `getopts` to parse their arguments should save the value of `OPTIND` on entry and restore it before returning. However, there will be cases when a function must change `OPTIND` for the calling shell.

The following example script parses and displays its arguments:

```
aflag=
bflag=
while getopts ab: name
    do
        case $name in
        )
                aflag=1;;

                bflag=1
                bval="OPTARG";;
        ?)
                printf "Usage: %s: [-a] [-b value] args0 $0
                exit 2;;
        esac
    done
if [ ! -z "$aflag" ] ; then
        printf "Option -a specified0
fi
if [ ! -z "$bflag" ] ; then
        printf "Option -b "%s" specified0 "$bval"
fi
shift $(($OPTIND -1))
printf "Remaining arguments are: %s0 "$*"
```

SEE ALSO

getopt(1), ksh(1), sh-posix(1), sh(1), getopt(3C).

STANDARDS CONFORMANCE

`getopts`: POSIX.2

NAME

getprivgrp - get special attributes for group

SYNOPSIS

getprivgrp [**-g** | *group_name*]

DESCRIPTION

getprivgrp lists the access privileges of privileged groups set by **setprivgrp** (see *setprivgrp*(1M)). If *group_name* is supplied, access privileges are listed for that group only. If the caller is not a member of *group_name*, no information is displayed. If **-g** is used, **getprivgrp** lists access privileges that have been granted to all groups. Otherwise, access privileges are listed for all privileged groups to which the caller belongs.

The super-user is a member of all groups. Access privileges include **RTPRIO**, **MLOCK**, **CHOWN**, **LOCKRDONLY**, and **SETRUGID**.

In the HP Clustered environment, privilege groups are maintained separately for each member of the cluster. The CHOWN privilege from a cnode is determined by the privilege groups set up on the cluster server.

AUTHOR

getprivgrp was developed by HP.

SEE ALSO

getprivgrp(2), setprivgrp(1M), privgrp(4), privilege(5).

NAME

gprof - display call graph profile data

SYNOPSIS

`gprof` [*options*] [*a.out* [*gmon.out* ...]]

DESCRIPTION

`gprof` produces an execution profile of C, Pascal, and FORTRAN programs. The effect of called routines is incorporated into the profile of each caller. Profile data is taken from the call graph profile file (`gmon.out` default) that is created by programs compiled with the `-G` option of `cc`, `pc`, and `f77` (see *cc*(1), *pc*(1), and *f77*(1)). That option also links in versions of the library routines that are compiled for profiling. The symbol table in the named object file (`a.out` default) is read and correlated with the call graph profile file. If more than one profile file is specified, `gprof` output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by `prof` (see *prof*(1)). This listing gives the total execution times and call counts for each function in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. `gprof` discovers all cycles in the call graph. All calls made into the cycle share the time of that cycle. A second listing shows the functions sorted according to the time they represent< including the time of their call graph descendants. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how the time of this function and the time of its descendants are propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle, each with their contributions to the time and call counts of the cycle.

Options:

`gprof` recognizes the following options:

`-a`	Suppress printing statically declared functions. If this option is given, all relevant information about the static function (such as time samples, calls to other functions, and calls from other functions) belongs to the function loaded just before the static function in the `a.out` file.
`-b`	Suppress printing a description of each field in the profile.
`-c`	Identify the static call graph of the program using a heuristic that examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.
`-e` *name*	Suppress printing the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that are not suppressed). More than one `-e` option can be given. Only one *name* can be given with each `-e` option.
`-E` *name*	Suppress printing the graph profile entry for routine *name* (and its descendants) as `-e` above, and also exclude the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, `-E mcount -E mcleanup` is the default.)
`-f` *name*	Print only the graph profile entry of the specified routine *name* and its descendants. More than one `-f` option can be given. Only one *name* can be given with each `-f` option.
`-F` *name*	Print only the graph profile entry of the routine *name* and its descendants (as `-f` above) and also uses only the times of the printed routines in total time and percentage computations. More than one `-F` option can be given. Only one *name* can be given with each `-F` option. The `-F` option overrides the `-E` option.
`-s`	Produce a profile file `gmon.sum` that represents the sum of the profile information in all specified profile files. This summary profile file can be given to subsequent executions of `gprof` (probably also with a `-s` option) to accumulate profile data across several runs of an `a.out` file.
`-z`	Display routines that have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the `-c` option for discovering which routines

were never called.

WARNINGS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

Parents that are not profiled have the time of their profiled children propagated to them, but they appear to be spontaneously invoked in the call graph listing, and do not have their time propagated further. Similarly, signal catchers, even though profiled, appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call `exit()` (see *exit*(2)) or return normally, for the profiling information to be saved in the `gmon.out` file.

DEPENDENCIES

Series 700/800

The `-c` option is not supported.

AUTHOR

`gprof` was developed by The University of California, Berkeley.

FILES

`a.out*`	Default object file.
`gmon.out*`	Default dynamic call graph and profile
`gmon.sum*`	Summarized dynamic call graph and profile
`/usr/lib/gprof.callg*`	Call graph description
`/usr/lib/gprof.flat*`	Flat profile description

SEE ALSO

cc(1), f77(1), pc(1), prof(1), exit(2), profil(2), crt0(3), monitor(3C).

gprof: A Call Graph Execution Profiler; Graham, S.L., Kessler, P.B., McKusick, M.K.; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*; SIGPLAN Notices; Vol. 17, No. 6, pp. 120-126, June 1982.

NAME

grep, egrep, fgrep - search a file for a pattern

SYNOPSIS

Plain call with pattern
grep [**-E** | **-F**] [**-cilnqsvx**] *pattern* [*file* ...]

Call with (multiple) -e pattern
grep [**-E** | **-F**] [**-cilnqsvx**] **-e** *pattern*... [**-e** *pattern*] ... [*file* ...]

Call with -f file
grep [**-E** | **-F**] [**-cilnqsvx**] [**-f** *pattern_file*] [*file* ...]

Obsolescent:
egrep [**-cefilnsv**] [*expression*] [*file* ...]

fgrep [**-cefilnsvx**] [*strings*] [*file* ...]

DESCRIPTION

grep searches the input text *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *grep* supports the Basic Regular Expression syntax (see *regexp*(5)). The **-E** option (*egrep*) supports Extended Regular Expression (ERE) syntax (see *regexp*(5)). The **-F** option (*fgrep*) searches for fixed *strings* using the fast Boyer-Moore string searching algorithm. **-E** and **-F** options treat new-lines embedded in the *pattern* as alternation characters. A null expression or string matches every line.

The forms *egrep* and *fgrep* are maintained for backward compatibility; use of **-E** and **-F** options is recommended for portability.

Options

-E	Extended regular expressions. Each pattern specified is a sequence of one or more EREs. The EREs can be separated by new-line characters or given in separate **-e** *expression* options. A pattern matches an input line if any ERE in the sequence matches the contents of the input line without its trailing new-line character. The same functionality is obtained by using *egrep*.
-F	Fixed strings. Each pattern specified is a sequence of one or more strings. Strings can be separated by new-line characters or given in separate **-e** *expression* options. A pattern matches an input line if the line contains any of the strings in the sequence. The same functionality is obtained by using *fgrep*.
-b	Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context. Block numbers are calculated by dividing by 512 the number of bytes that have been read from the file and rounding down the result.
-c	Only a count of matching lines is printed.
-e *expression*	Same as a simple *expression* argument, but useful when the *expression* begins with a hyphen (-). Multiple **-e** options can be used to specify multiple patterns; an input line is selected if it matches any of the specified patterns.
-f *pattern_file*	The regular *expression* (*grep* and *grep* **-E**) or *strings* list (*grep* **-F**) is taken from the *pattern_file*.
-i	Ignore uppercase/lowercase distinctions during comparisons.
-l	Only the names of files with matching lines are listed (once), separated by new-lines. If standard input is searched, a pathname of - is listed.
-n	Each line is preceded by its relative line number in the file starting at 1. The line number is reset for each file searched. This option is ignored if **-c**, **-b**, **-l**, or **-q** is specified.
-q	(Quiet) Do not write anything to the standard output, regardless of matching lines. Exit with zero status upon finding the first matching line. Overrides any options that would produce output.
-s	Error messages produced for nonexistent or unreadable files are suppressed.

-v All lines but those matching are printed.

-x (eXact) Matches are recognized only when the entire input line matches the fixed string or regular expression.

In all cases in which output is generated, the file name is output if there is more than one input file. Care should be taken when using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes (**´...´**).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters, the classification of characters as letters, the case information for the *-i* option, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, the commands behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

Exit status is:

0	One or more matches found
1	No match found
2	Syntax error or inaccessible file (even if matches were found).

EXAMPLES

In the Bourne shell (*sh*(1)) the following example searches two files, finding all lines containing occurrences of any of four strings:

```
grep -F ´if
then
else
fi´ file1 file2
```

Note that the single quotes are necessary to tell *grep -F* when the strings have ended and the file names have begun.

For the C shell (*csh*(1)) the following command can be used:

```
grep -F ´if\
then\
else\
fi´ file1 file2
```

To search an **address** file with the following entries:

```
Ken   112 Warring St.  Apt. A
Judy  387 Bowditch  Apt. 12
Ann   429 Sixth St.
```

the command:

```
grep Judy address
```

prints:

```
Judy  387 Bowditch  Apt. 12
```

To search a file for lines that contain either a Dec or Nov, use the command:

```
grep -E '[Dd]ec | [Nn]ov' file
```

or

```
egrep -i 'dec | nov' file
```

Search all files in the current directory for the string **xyz**:

```
grep xyz *
```

Search all files in the current directory subtree for the string **xyz**, and ensure that no error occurs due to filename expansion exceeding system argument list limits:

```
find . -type f -print |xargs grep xyz
```

The previous example does not print the name of files where string **xyz** appears. To force **grep** to print file names, add a second argument to the **grep** command portion of the command line:

```
find . -type f -print |xargs grep xyz /dev/null
```

In this form, the first filename is that produced by `find`, and the second filename is the null file.

SEE ALSO

sed(1), sh(1), environ(5), lang(5), regcomp(3c), regexp(5).

STANDARDS CONFORMANCE

`grep`: SVID2, XPG2, XPG3, POSIX.2

`egrep`: SVID2, XPG2, XPG3, POSIX.2

`fgrep`: SVID2, XPG2, XPG3, POSIX.2

NAME

groups - show group memberships

SYNOPSIS

`groups` [`-p`] [`-g`] [`-l`] [*user*]

DESCRIPTION

`groups` shows the groups to which the caller or the optionally specified *user* belong. If invoked with no arguments, `groups` prints the current access list returned by `getgroups()` (see *getgroups*(2)).

Each user belongs to a group specified in the password file `/etc/passwd` and possibly to other groups as specified in the files `/etc/group` and `/etc/logingroup`. A user is granted the permissions of those groups specified in `/etc/passwd` and `/etc/logingroup` at login time. The permissions of the groups specified in `/etc/group` are normally available only with the use of `newgrp` (see *newgrp*(1)). If a user name is specified with no options, `groups` prints the union of all these groups.

The `-p`, `-g`, and `-l` options limit the printed list to those groups specified in `/etc/passwd`, `/etc/group`, and `/etc/logingroup`, respectively. If a user name is not specified with any of these options, `cuserid()` is called to determine the default user name (see *cuserid*(3S)).

The printed list of groups is sorted in ascending collation order (see Environment Variables below).

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the order in which the output is sorted.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `groups` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

EXAMPLES

Check file `/etc/logingroup` and display all groups to which user `tim` belongs:

```
groups -l tim
```

AUTHOR

`groups` was developed by the University of California, Berkeley.

FILES

```
/etc/group
/etc/logingroup
/etc/passwd
```

SEE ALSO

id(1), newgrp(1), getgroups(2), initgroups(3C), cuserid(3S), group(4).

NAME

head - give first few lines

SYNOPSIS

`head` [`-c`|`-l`] [`-n` *count*] [*file* ...]

Obsolescent:
`head` [`-`*count*] [*file* ...]

DESCRIPTION

`head` prints on standard output the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

If multiple *files* are specified, `head` outputs before each file a line of this form:

`==>` *file* `<==`

Options

`-c`	The quantity of output is measured in bytes.
`-`*count*	The number of units of output. This option is provided for backward compatibility (see `-n` below) and is mutually exclusive of all other options.
`-l`	The quantity of output is measured in lines; this is the default.
`-n` *count*	The number of lines (default) or bytes output. *count* is an unsigned decimal integer. If `-n` (or *-count*) is not given, the default quantity is 10. This option provides the same functionality as the *-count* option, but in a more standard way. Use of the `-n` option is recommended where portability between systems is important.

SEE ALSO

tail(1).

STANDARDS CONFORMANCE

`head`: POSIX.2

NAME
help - ask for help

SYNOPSIS
`help` [*args*]

DESCRIPTION
help finds information to explain a message from a command or explain the use of a command. Zero or more arguments can be supplied. If no arguments are given, *help* prompts for one.

The arguments can be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1	Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., `ge6`, for message 6 from the `get` command).
type 2	Does not contain numerics (as a command, such as `get`)
type 3	Is all numeric (e.g., `212`)

The response of the program is the explanatory information related to the argument, if there is any.

When all else fails, try `help stuck`.

EXTERNAL INFLUENCES
Environment Variables
`LC_CTYPE` determines the interpretation of text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, *help* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support
Single- and multi-byte character code sets are supported.

DIAGNOSTICS
Use the `help` command for explanations.

WARNINGS
Only SCCS and a very few other commands currently use `help`.

FILES
`/usr/lib/help`	directory containing files of message text.
`/usr/lib/help/helploc`	file containing locations of help files not in `/usr/lib/help`.

SEE ALSO
SCCS User's Guide, in *Programming on HP-UX.*

NAME

hostname - set or print name of current host system

SYNOPSIS

hostname [*name_of_host*]

DESCRIPTION

hostname prints the name of the current host, as given in the *gethostname*(2) system call. Users who have appropriate privileges can set the *hostname* by giving the argument *name_of_host*; this is usually done in the startup script **/etc/rc**. The *name_of_host argument is restricted to* **MAXHOSTNAMELEN** characters as defined in **<sys/param.h>**.

The system might be known by other names if networking products are supported. See the node manager's documentation supplied with your system for details.

WARNINGS

Many types of networking services are supported on HP-UX, each of which uses a different assigned system name and naming convention. To ensure predictable system behavior, it is essential that system names (also called host names or node names) be assigned in such a manner that they do not create conflicts when the various networking facilities interact with each other.

The system does not rely on a single system name in a specific location, partly because different services use dissimilar name formats as explained below. System names are assigned by using the **uname -S**, **hostname**, and **nodename** commands. In addition, the system name used in the HP Clustered Environment (called the **cnode name**) is assigned in the cluster configuration file **/etc/clusterconf**. System names are assigned as follows:

Nodename	Command/File	Format	Used By
NetIPC name	**nodename** *name*	*foo*[.*a*[.*b*]]	NS Services and NetIPC
Internet name	**hostname** *name*	*foo*[.*x*.*y*.*z*...]	ARPA and NFS Services
UUCP name	**uname -S** *name*	*foo*	UUCP and related programs
cnode name	**/etc/clusterconf**	*foo*	Cluster server and clients

where *foo* represents the assigned system name (it is *strongly* recommended that *foo* be identical for all commands and locations) and the optional .*x*.*y*.*z* or .*a*.*b* follow the specified notation for the particular ARPA/NFS or NS/NetIPC environment.

Internet names are also frequently called hostnames or domain names (not to be confused with NFS domain names). Refer to *hostname*(5) for more information about Internet naming conventions.

Whenever the system name is changed in any file or by use of any of the above commands, it should also be changed in all other locations as well. Other files or commands in addition to those above (such as **/usr/lib/uucp/Permissions** if used to circumvent **uname**, for example) may contain or alter system names. To ensure correct operation, they should also use the same system name.

System names are normally assigned by the **/etc/rc** script at start-up, and should not be altered elsewhere.

AUTHOR

hostname was developed by the University of California, Berkeley.

SEE ALSO

uname(1), gethostname(2), sethostname(2), uname(2), hostname(5).

NAME

hp - handle special functions of HP 2640 and HP 2621-series terminals

SYNOPSIS

`hp [-e] [-m]`

DESCRIPTION

hp supports special functions of the Hewlett-Packard HP 2640 and HP 2621 series of terminals, with the primary purpose of producing accurate representations of most `nroff` output. A typical use is:

`nroff -h` *files* ... `| hp`

Regardless of the hardware options on a given terminal, **hp** tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

Options

hp recognizes the following options:

`-e` Specify that your terminal has the "display enhancements" feature, to make maximal use of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, **hp** assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode; that is, dark-on-light, rather than light-on-dark.

`-m` Request minimization of output by removing new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; that is, any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

DIAGNOSTICS

`line too long`
The representation of a line exceeds 1,024 characters.

RETURN VALUE

hp returns zero for normal termination, and 2 for all errors.

WARNINGS

An "overstriking sequence" is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text "disappear"; in particular, tables generated by `tbl` that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to **hp** is piped through `col` (see *col*(1)).

Although some terminals do support numerical superscript characters, no attempt is made to display them.

SEE ALSO

col(1), neqn(1), nroff(1), tbl(1).

NAME
hyphen - find hyphenated words

SYNOPSIS
hyphen [*files*]

DESCRIPTION
hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, **hyphen** can be used as a filter.

EXAMPLES
Prepare an **nroff** hyphenation proofreading file for *textfile*.

```
mm textfile | hyphen
```

WARNINGS
hyphen cannot cope with hyphenated *italics* (i.e., underlined) words; it often misses them completely or mangles them.

hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

SEE ALSO
mm(1), nroff(1).

NAME

iconv - code set conversion

SYNOPSIS

`iconv -f` *fromcode* `-t` *tocode* [*file* ...]

DESCRIPTION

iconv converts the encoding of characters in the input files from the *fromcode* code set to the *tocode* code set, and writes the results on standard output. If no input files are given, **iconv** reads from standard input. If **-** appears as an input file name, **iconv** reads standard input at that point. **- -** can be used to delimit the end of options (see *getopt*(3C)).

Options

iconv recognizes the following options:

-f *fromcode*	Identify the code set corresponding to option argument *fromcode* as the code set that the input will be converted "from".
-t *tocode*	Identify the code set corresponding to option argument *tocode* as the code set that the input will be converted "to".

The *fromcode* and *tocode* names can be any length, but only the first four and the last letter are used to identify the code set. The names can contain any letter except a closing curly brace (**}**). HP-supplied *fromcode* and *tocode* names and their corresponding code sets include:

Names	Code Set
american_e	American EBCDIC
arabic8	HP ARABIC8
arabic_e	Arabic EBCDIC
big5	Traditional Chinese BIG5
bulgarian_e	Bulgarian EBCDIC
c-french_e	Canadian-French EBCDIC
chinese-t_e	Traditional Chinese EBCDIC
czech_e	Czech EBCDIC
danish_e	Danish EBCDIC
dutch_e	Dutch EBCDIC
english_e	English EBCDIC
finnish_e	Finnish EBCDIC
french_e	French EBCDIC
german_e	German EBCDIC
greek8	HP GREEK8
greek_e	Greek EBCDIC
hebrew8	HP HEBREW8
hebrew_e	Hebrew EBCDIC
hungarian_e	Hungarian EBCDIC
icelandic_e	Icelandic EBCDIC
iso8859_1	ISO 8859/1
iso8859_2	ISO 8859/2
iso8859_5	ISO 8859/5
iso8859_6	ISO 8859/6
iso8859_7	ISO 8859/7
iso8859_8	ISO 8859/8
iso8859_9	ISO 8859/9
italian_e	Italian EBCDIC
japanese_e	Japanese EBCDIC
japanese15	HP JAPANESE15
jis	Japanese Industrial Standard
sjis	Shifted-Japanese Industrial Standard
ujis	Japanese Extended UNIX Code
kana8	HP KATAKANA8
katakana_e	Katakana EBCDIC
korean15	HP KOREAN15

`korean_e`	Korean EBCDIC
`norwegian_e`	Norwegian EBCDIC
`polish_e`	Polish EBCDIC
`portuguese_e`	Portuguese EBCDIC
`roc15`	HP Traditional CHINESE15
`roman8`	HP ROMAN8
`rumanian_e`	Rumanian EBCDIC
`russian_e`	Russian EBCDIC
`serbocroation_e`	Serbocroation EBCDIC
`slovene_e`	Slovene EBCDIC
`spanish_e`	Spanish EBCDIC
`swedish_e`	Swedish EBCDIC
`thai8`	HP THAI8
`thai_e`	Thai EBCDIC
`turkish8`	HP TURKISH8
`turkish_e`	Turkish EBCDIC

EXTERNAL INFLUENCES

Environment Variables

The LANG environment variable determines the language in which messages are displayed.

International Code Set Support

Single and multi-byte character code sets are supported.

WARNINGS

If an input character does not have a valid equivalent in the code set selected by the `-t` option (the "to" code set), it is mapped to as system-defined default character.

If an input character does not belong to the code set selected by the `-f` option (the "from" code set), the command terminates.

EXAMPLES

Convert the contents of file `foo` from code set Roman8 to ISO 8859/1 and store the results in file `bar`.

```
iconv -f roman8 -t iso8859_1 foo > bar
```

AUTHOR

`iconv` was developed by HP.

SEE ALSO

getopt(3C)

STANDARDS CONFORMANCE

`iconv`: XPG2, XPG3

NAME

id - print user and group IDs and names

SYNOPSIS

`id` [`-g`|`-u`] [`-nr`] [*user*]

DESCRIPTION

`id` writes a message on the standard output, giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

If the *user* operand is specified and the process is a user with appropriate privileges, the user and group IDs of the selected user are written. In this case, effective IDs are assumed to be identical to real IDs.

If the process has supplementary group affiliations (see *groups*(1)), the supplementary group affiliations are also written.

Options

`-g` Output only the group ID in decimal format. To modify, use the `-n` option. The default group ID is the effective group ID. To modify, use the `-r` option. If the invoking process has supplementary group affiliations that are different from the effective group ID, output each such affiliation in decimal format on the same line.

`-n` With `-u` or `-g`, output the name in character string format instead of the numeric ID.

`-r` With `-u` or `-g`, output the real ID instead of the effective ID.

`-u` Output only the user ID in decimal format. To modify, use the `-n` option. The default group ID is the effective group ID. To modify, use the `-r` option.

AUTHOR

`id` was developed by HP and AT&T.

SEE ALSO

groups(1), logname(1), getuid(2).

STANDARDS CONFORMANCE

`id`: SVID2, XPG2, XPG3, POSIX.2

NAME

ident - identify files in RCS

SYNOPSIS

`ident` *file* ...

DESCRIPTION

`ident` searches the named files for all occurrences of the pattern *$keyword*:...$, where *keyword* is one of the following:

```
Author      Log
Date        Revision
Header      Source
Locker      State
```

These patterns are normally inserted automatically by the RCS `co` command, but can also be inserted manually (see *co*(1)).

`ident` works on text files as well as object files. For example, if the C program in file `f.c` contains:

```
char rcsid[] = "$Header:  Header information $" ;
```

and `f.c` is compiled into `f.o`, the command:

```
ident f.c f.o
```

prints:

```
f.c:
           $Header: Header information $
f.o:
           $Header: Header information $
```

AUTHOR

`ident` was developed by Walter F. Tichy.

SEE ALSO

ci(1), co(1), rcs(1), rcsdiff(1), rcsintro(5), rcsmerge(1), rlog(1), rcsfile(4).

NAME

ied - input editor and command history for interactive programs

SYNOPSIS

`ied` [`-dirt`] [`-h` *file*] [`-s` *size*] [`-p` *prompt*] [`-k` *charmap*] *utility* [*arguments* ...]

DESCRIPTION

`ied` is a utility command that is intended to act as an interface between the user and an interactive program such as bc, bs, or Bourne shell, providing most of the line editing and history functionality found in the Korn shell. `ied` interprets the *utility* name as the command to be executed, and passes *arguments* as the arguments to the utility. Subsequent input to *utility* then has access to editing and history functions very similar to those provided by *ksh*.

`ied` monitors the state of the pty it uses to run the command, and, whenever the application it is running, changes the state from the state of the tty when `ied` started, `ied` becomes "transparent". This allows programs to do shell escapes to screen-smart programs. In general, `ied` should not in any way interfere with any action taken by any program for which it provides a front end. This includes Korn shell itself: in this case `ied` would provide history for any application that was run by `ksh`, and `ksh` would provide its own independent history. In a useful extreme case, `ied` can be used as a front end to the login shell (which might be `ksh` or `csh`). In this case, all applications that use normal line editing gain line editing and history, sharing a single history. The shell would continue to have its own independent history if it provides such a mechanism.

When `ied` is in its transparent mode, no history is saved. In particular the `ex` mode of `vi` does not use normal line editing (rather, it simulates it) and `ied` cannot provide history in this case. The `Subject:` and address line editing of mailx also cannot be edited with `ied`.

Options

Several options and command-line arguments control `ied`'s operation:

`-d` Debug mode. Print information about the operation of the program. It is best used to determine if a program puts `ied` into transparent mode unexpectedly.

`-h` *filename* Keep the history in a file named *filename*. If a file of that name already exists and is a history file, the latter part of it (the last *size* lines as specified by the `-s` option) is used as the initial value of the history. If the `-h` option is not used, the environment variable `IEDHISTFILE` is used to supply the name. If neither are present an unnamed temporary file is used, and no initial value is provided.

`-i` Force interactive mode. Normally `ied` simply `exec`s the command to which it is asked to be a front end when the standard input is not a tty (this allows aliases to be used for commands used in shells without interfering with their operation). This option forces `ied` to remain as a front end, and all editing functions are in place. This permits a utility that behaves differently in interactive and batch modes to be driven from a pipe or file in interactive mode. This is particularly useful in testing commands that make this distinction.

`-k` *charmap* *charmap* is a file of 256 or fewer lines. The line number in the file is the ordinal of a character as seen as input by `ied`, and the character on the line is the character generated as output (and also used as editing characters). This allows remapping of (ordinary) keys such as for a Dvorak keyboard. Characters must start in column one of each line, and be represented as 1-4 characters followed by a space or the new-line character for the next line. Characters after the space are ignored as comments. Single-character entries represent themselves. Two-character entries where the first character is a circumflex (^) converts the second character to the corresponding control character. Two-character sequences where the first character is backslash (\) use the C language conventions:

`\n`	newline	`\s`	space
`\\`	escape	`\0`	null
`\r`	return	`\f`	form feed
`\t`	tab	`\v`	vertical tab
`\b`	backspace		

Three- and four-character sequences must be `\`*nn* or `\`*nnn*, giving the octal value for the character. If *charmap* is less than 256 lines long, the remaining characters are mapped to themselves.

-p *prompt* Many commands do not prompt when ready for input. **ied** approximates a prompting mechanism for such commands. This is not always perfectly successful, but for many commands it helps. In the worst case, the prompt is interspersed with output in the wrong location. *prompt* is a string as used in the format argument to *printf*(3S). The only **%** conversions that can be included are up to one instance of **%d** which is converted to the sequential number of the command, and any number of occurrences of **%%** which is treated as a literal **%** character. Prompting is suppressed when **ied** is operating in transparent mode.

-r This sets "non-raw" mode. Normally **ied** uses its own editing capabilities when reading simple text. This causes **ied** to use tty line discipline most of the time. The disadvantage of the default mode is that more context switches and general processing are required. The advantage is that **ied** is more transparent. For example, to specifically send an end-of-file in the non-raw mode requires that the end-of-file character (usually Ctrl-D) be followed by a carriage return. Similarly the "literal next" function (Ctrl-V) cannot escape the line-erase and line-kill functions in non-raw mode.

-s *size* This option specifies the size of the history buffer. When **ied** is started with an existing history file, approximately the last *size* lines are available to the history mechanism (the number is not guaranteed to be exactly *size*). Other lines in the file are retained until such time as **ied** is started on that history file and it exceeds approximately 4K bytes in size, at which time **ied** discards older entries at the beginning of the file until it is near 4 Kbytes in size. Since this occurs only at startup, history files can grow to be quite large between restarts. Larger values of *size* make the process image larger.

If **-s** is not specified, the value of the environment variable **IEDHISTSIZE** is used. If neither is specified, a default is used.

-t Set transparent mode. This forces **ied** to permanently be in transparent mode (as discussed above). It is primarily useful with **-i** for some classes of automated processing. In particular, it is useful for driving a command if the command takes as input what **ied** would interpret as editing characters. Thus with the appropriate combinations of **-i** and **-t**, it is possible to drive an editor such as **vi** or a screen-smart application from a batch file.

Should something go wrong with **ied**, the **SIGQUIT** signal, repeated 3 times, usually aborts **ied**. The exception is the case of a fully transparent application, where **ied** must be killed from another window or terminal. This is really relevant only when there is no way to direct the serviced process to terminate itself.

The editing capabilities of **ied** are essentially those found in **ksh**. Only those that differ from **ksh** are described below. As in **ksh**, the style of editing is determined from the environment variable **VISUAL**, or from **EDITOR** if **VISUAL** is not specified. The value examined should end in **vi**, **emacs**, or **gmacs** to specify an editor type. If it does not, **ied** does no editing, and history is not accessible.

In vi mode:

J Join lines. Considering the most recently edited line (which is empty immediately after a line is sent to the application) to be the "last line" of the history, the current line being displayed from the history is appended to the end of the last line, and the position in the history is reset to be at the last line which is then displayed. A space is inserted between the old and new text on the last line. The cursor is left on that space. Because **ied**'s understanding of line

	continuation is minimal, this is useful for editing long statements.
v	Not supported.
V	Not supported.
#	Sends nothing to the application, but inserts the line in the history (useful for adding comments to history file).
<esc> **,** ***** **,** **=**	(Filename expansion). Not supported.
@	Macro expansion. Not supported.
	Note however that `ksh` has a rarely-used function _ that substitutes words from the previous line (this is not the macro **$_**, but rather an editor command). If a preceding *count* is given, it uses the *count*th word of the last line. This is much more useful with `ied`.

In emacs/gmacs mode:

M-*, **M-=**, **M-***<esc>*	(filename expansion) Not supported.
	Note that the command **M-.** (and it's synonym **M-_**) provide the same functionality as the vi mode _ command.
Macro expansion.	Not supported.
`^O`	Although supported, it may not always appear correctly on the screen. The **^L** command can be used to redraw the line. See below for the discussion on prompting.

EXAMPLES

Add interactive editing to the `bc` command:

```
ied bc
```

Execute `vi` on `testfile` using comands taken from `script`:

```
cat script | ied -i -t vi testfile
```

Note that without the use of `ied`, `vi` would misbehave because its standard input would not be a terminal device. In this case the `-t` is not required because `vi` puts itself in raw mode, but for an application that does not, `-t` might be required.

The command line

```
ied -i -t grep '^x:' data_file | tee x_lines
```

searches the file `data_file` for lines begining with `x:`, sending one copy to the terminal and a second to file `x_lines`, just like the command line

```
grep '^x:' data_file | tee x_lines
```

The difference is that in the command line without `ied`, `grep` writes directly to a pipe, and thus buffers its output. If `data_file` is very large and not many lines match the pattern, output to the terminal is delayed. By using `ied`, the output of `grep` goes to a pty instead, which causes `grep` to output each line as it is ready.

WARNINGS

Since `ied` cannot know everything about every application, it is possible that it can become confused, with either the timing or the prompt being out of phase with the application. Since the use of `ied` is never required, it is the user's choice to determine whether the application is more usable with or without `ied`. In general, however, programs that do not confuse `ied` are usually also the most likely to benefit from its use.

`ied` tries to intuit the currently active prompt when it is not providing one itself. However, this is not always successful. Even when it is successful, the timing of `ied` and the serviced command may occasionally confuse the output. The `^L` commands in both emacs and vi modes redraw the edit line in a consistent fashion that can be used to create the next command.

AUTHOR

`ied` was developed by HP.

SEE ALSO

ksh(1).

NAME

imageview - display bit-mapped image files on an X11 display

SYNOPSIS

imageview [**filename**] [-**d** *host* **:** *number*] [-**g** *geometry*] [-**o** *orientation*]

DESCRIPTION

imageview displays bit-mapped image files on an X11 display. Use **imageview**'s graphical user interface (based on OSF/Motif) to display an image then manipulate the image on the screen, using various rotate, zoom, and resize controls.

The following table lists image file types that can be displayed, what these files contain, the filename suffix (extension) VUE uses to display the correct icon, and the file version supported.

File Type	VUE File Extension	File Contents	Version Supported
TIFF	`.tif`	PC, scanned, and FAX images	5.0 (6.0 for TIFF JPEG)
JFIF	`.jpg .jpeg`	JPEG-compressed images	8-R8
GIF	`.gif`	xv and xgif images	87a
XWD	`.xwd`	Pixmap images from xwd (Z format)	X11
XBM	`.xbm .bm`	Bitonal X bitmap images	X11
XPM	`.xpm .pm`	Color X pixmap images	3.0
BMF	`.bmf`	Starbase bitmap images (Z format)	1

TIFF images can be in uncompressed format, or any of the following compressed formats: JPEG, LZW, G3, G4, or Packbits.

The *filename* argument is an absolute or relative pathname of a file containing one or more images in a supported format, and can contain ~, *, and ? pattern specifiers. If *filename* resolves to more than one file, **imageview** displays the first file found.

Options

imageview recognizes the following options:

-**d** *host* **:** *number* Specifies the display to use. Arguments are:

host	Name of the host machine.
number	Number of the display.

If the -**d** option is omitted, **imageview** uses the value of the **DISPLAY** environment variable.

-**g** *geometry*
Specifies the position and maximum size of the viewer window. The viewer window consists of an image window framed by a menu and control panel. The syntax for *geometry* is the standard X geometry syntax; see *X*(1) in the reference section of *Using the X Window System*.

If the -**g** option is omitted, **imageview** creates a window that matches the image size. If the image is too large for the display, **imageview** creates the largest window possible without changing the ratio of image height to image width.

-**o** *orientation*
Sets the initial orientation of the image. *orientation* is one of the following:

landscape	Rotate the image 90 degrees clockwise before displaying.
portrait	Do not rotate the image.

If the -**o** option is omitted, -**o portrait** is used.

EXAMPLES

Display the file **rgb.tif** in the current directory and rotate the first image found 90 degrees clockwise.

```
        imageview rgb.tif -o landscape
```

AUTHOR

`imageview` was developed by HP.

SEE ALSO

X(1) in *Using the X Window System*.

NAME

insertmsg - use findstr(1) output to insert calls to catgets(3C)

SYNOPSIS

`insertmsg` [`-h`] [`-n` *number*] [`-i` *amount*] [`-s` *number*] *stringlist*

DESCRIPTION

`insertmsg` examines the file *stringlist*, which is assumed to be the output of `findstr` after subsequent editing to remove any strings that do not need to be localized (see *findstr*(1)). If the `-h` option is specified, `insertmsg` places the following lines at the beginning of each file named in *stringlist:*

```
#ifndef NLS
#define catgets(i,sn,mn,s) (s)
#else NLS
#defineNL_SETN number
#include <nl_types.h>
#endif NLS
```

where *number* is a set number defined by the `-s` option; the default is `1`. For each string in *stringlist*, `insertmsg` surrounds the string in the corresponding file with an expression of the form:

`(catgets(catd,NL_SETN,`*msg_num*`,"`*default string*`"))`

The *default string* is the original string referenced by the line in *stringlist*, and *msg_num* is replaced by the message number assigned to that string. The assigned message numbers begin with the number defined by the `-n` option and are incremented by the amount defined by the `-i` option. The default is `1` for both the starting message number and the increment. If *name*`.c` is the file to be modified, as specified within the *stringlist* file, *insertmsg* places the modified source in `nl_`*name*`.c`. The user must then manually edit the file `nl_`*name*`.c` to insert the following statements:

```
nl_catd catd;
catd = catopen(" appropriate message catalog ",0);
```

The data type `nl_catd` is defined in `<nl_types.h>` and `catd` is a parameter to the calls to *catgets*, which are inserted for each string from *stringlist*.

`insertmsg` also sends to the standard output a file that can be used as input to `gencat` (see *gencat*(1)).

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text as single- and/or multi-byte characters.

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `insertmsg` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

If `insertmsg` does not find opening or closing double quotes where required in the strings file, it prints `insertmsg exiting : lost in strings file` and aborts. If this happens, check the strings file to ensure that the lines that have been kept there have not been altered.

WARNINGS

Previous implementations of `insertmsg` inserted a call to `nl_msg` rather than `catgets`. `nl_msg` outputs *default string* if the message retrieval is unsuccessful for any reason, whereas `catgets` reverts to *default string* only when it is unable to open the message catalog. Any other failed attempt to retrieve the message results in the output of a null string.

If the `-h` option is not used, it may be necessary to manually add the following statement to the file created by `insertmsg`:

```
#include <nl_types.h>
```

`insertmsg` inserts a pointer to a static area that is overwritten on each call.

AUTHOR

`insertmsg` was developed by HP.

SEE ALSO

findstr(1), gencat(1), catgets(3C), catopen(3C).

NAME

iostat - report I/O statistics

SYNOPSIS

`iostat` [`-t`] [*interval* [*count*]]

DESCRIPTION

`iostat` iteratively reports I/O statistics for each active disk on the system. Disk data is arranged in a three-column format:

Column Heading	Interpretation
`bps`	Kbytes transferred per second
`sps`	Number of seeks per second
`msps`	Milliseconds per average seek

If two or more disks are present, data for successive disks is presented side-by-side across the page.

To compute this information, seeks, data transfer completions, and the number of words transferred are counted for each disk. Also, the state of each disk is examined **HZ** times per second (as defined in <`sys/param.h`>) and a tally is made if the disk is active. These numbers can be combined with the transfer rates of each device to determine average seek times for each device.

Options

`iostat` recognizes the following options and command-line arguments:

`-t` Report terminal statistics as well as disk statistics. Terminal statistics include:

`tin`	Number of characters read from terminals.
`tout`	Number of characters written to terminals.
`us`	Percentage of time system has spent in user mode.
`ni`	Percentage of time system has spent in user mode running low-priority (*nice* d) processes.
`sy`	Percentage of time system has spent in system mode.
`id`	Percentage of time system has spent idling.

interval
Display successive lines which are summaries over the last *interval* seconds. The first line reported is for the time since a reboot and each subsequent line is for the last interval only.

count
Repeat the statistics *count* times.

EXAMPLES

Show current I/O statistics for all disks:

```
iostat
```

Display I/O statistics for all disks every 10 seconds until INTERRUPT or QUIT is pressed:

```
iostat 10
```

Display I/O statistics for all disks every 10 seconds and terminate after 5 successive readings:

```
iostat 10 5
```

Same except also show terminal and processor statistics:

```
iostat -t 10 5
```

DEPENDENCIES

Series 300/400

`iostat` does not yet report disk information for SCSI disks.

AUTHOR

`iostat` was developed by the University of California, Berkeley, and HP.

SEE ALSO

vmstat(1).

NAME

ipcrm - remove a message queue, semaphore set or shared memory id

SYNOPSIS

`ipcrm` [*options*]

DESCRIPTION

`ipcrm` removes one or more specified messages, semaphores or shared memory identifiers. The identifiers are specified by the following *options*:

`-q` *msqid*	Remove the message queue identifier *msqid* from the system and destroy the message queue and data structure associated with it.
`-m` *shmid*	Remove the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
`-s` *semid*	Remove the semaphore identifier *semid* from the system and destroy the set of semaphores and data structure associated with it.
`-Q` *msgkey*	Remove the message queue identifier, created with key *msgkey*, from the system and destroy the message queue and data structure associated with it.
`-M` *shmkey*	Remove the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
`-S` *semkey*	Remove the semaphore identifier, created with key *semkey*, from the system and destroy the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys can be found by using `ipcs` (see *ipcs*(1)).

In the HP Clustered environment, messages, semaphores, and shared memory are not global across members of the cluster. Therefore, `ipcrm` can be used to remove only local identifiers.

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

STANDARDS CONFORMANCE

`ipcrm`: SVID2

NAME

ipcs - report inter-process communication facilities status

SYNOPSIS

`ipcs` [*options*]

DESCRIPTION

`ipcs` prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information displayed is controlled by the following *options*:

`-q`	Print information about active message queues.
`-m`	Print information about active shared memory segments.
`-s`	Print information about active semaphores.

If any of the options `-q`, `-m`, or `-s` are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

`-b`	Print largest-allowable-size information (maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores). See below for the meanings of columns in a listing.
`-c`	Print creator's login name and group name. See below.
`-o`	Print information on outstanding usage (number of messages on queue and total number of bytes in messages on queue for message queues, and number of processes attached to shared memory segments).
`-p`	Print process number information (process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments). See below.
`-t`	Print time information (time of the last control operation that changed access permissions for all facilities; time of last `msgsnd()` and last `msgrcv()` on message queues, last `shmat()` and last `shmdt()` on shared memory, last `semop()` on semaphores (see *semop*(2)). See below.
`-a`	Use all print *options* (this is a shorthand notation for `-b`, `-c`, `-o`, `-p`, and `-t`).
`-C` *corefile*	Use file *corefile* in place of `/dev/kmem`.
`-N` *namelist*	The argument will be taken as the name of an alternate *namelist* (`/hp-ux` is the default).

The column headings and the meaning of the columns in an `ipcs` listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

`T` `(all)`
Facility type:

`q`	message queue;
`m`	shared memory segment;
`s`	semaphore.

`ID` (all)
The identifier for the facility entry.

`KEY` (all)
The key used as an argument to `msgget()`, `semget()`, or `shmget()` to create the facility entry. (Note: The key of a shared memory segment is changed to `IPC_PRIVATE` when the segment has been removed until all processes attached to the segment detach it.)

`MODE` `(all)`
The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

`R`	if a process is waiting on a `msgrcv()`;
`S`	if a process is waiting on a `msgsnd()`;
`D`	if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
`C`	if the associated shared memory segment is to be cleared when the first attach is executed;
`-`	if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.
The permissions are indicated as follows:

`r`	if read permission is granted;
`w`	if write permission is granted;
`a`	if alter permission is granted;
`-`	if the indicated permission is *not* granted.

`OWNER` (all)
The login name of the owner of the facility entry.

`GROUP` (all)
The group name of the group of the owner of the facility entry.

`CREATOR` (a,c)
The login name of the creator of the facility entry.

`CGROUP` (a,c)
The group name of the group of the creator of the facility entry.

`CBYTES` (a,o)
The number of bytes in messages currently outstanding on the associated message queue.

`QNUM` (a,o)
The number of messages currently outstanding on the associated message queue.

`QBYTES` (a,b)
The maximum number of bytes allowed in messages outstanding on the associated message queue.

`LSPID` (a,p)
The process ID of the last process to send a message to the associated queue.

`LRPID` (a,p)
The process ID of the last process to receive a message from the associated queue.

`STIME` (a,t)
The time the last message was sent to the associated queue.

`RTIME` (a,t)
The time the last message was received from the associated queue.

`CTIME` (a,t)
The time when the associated entry was created or changed.

`NATTCH` (a,o)
The number of processes attached to the associated shared memory segment.

`SEGSZ` (a,b)
The size of the associated shared memory segment.

`CPID` (a,p)
The process ID of the creator of the shared memory entry.

LPID (a,p)
The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,t)
The time the last attach was completed to the associated shared memory segment.

DTIME (a,t)
The time the last detach was completed on the associated shared memory segment.

NSEMS (a,b)
The number of semaphores in the set associated with the semaphore entry.

OTIME (a,t)
The time the last semaphore operation was completed on the set associated with the semaphore entry.

WARNINGS

`ipcs` produces only an approximate indication of actual system status because system processes are continually changing while `ipcs` is acquiring the requested information.

FILES

`/etc/group`	group names
`/hp-ux`	system namelist
`/dev/kmem`	memory
`/etc/passwd`	user names

SEE ALSO

msgop(2), semop(2), shmop(2).

STANDARDS CONFORMANCE

`ipcs`: SVID2

(Requires Optional ALLBASE/SQL Software)

NAME

isql - ALLBASE/SQL interactive SQL interface

SYNOPSIS

`isql`

REMARKS

The ALLBASE/SQL product must be installed on the system before `isql` can be used.

DESCRIPTION

`isql` invokes the Interactive SQL interface for defining and accessing an ALLBASE/SQL relational database environment (DBEnvironment). There are no options available with this command. `isql` can be executed by all system users.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

AUTHOR

`isql` was developed by HP.

FILES

`/usr/lib/sqldaemon`	Cleanup daemon program file.
`/usr/lib/hpsqlproc`	ALLBASE/SQL program file.
`/usr/bin/isql`	Interactive SQL program file.
`/usr/bin/sqlutil`	SQL utility program file.
`/usr/lib/hpsqlcat`	ALLBASE/SQL message catalog file.
`/usr/lib/nls/$LANG/hpsqlcat`	Localized message catalog file.
`/usr/lib/isqlwel`	Interactive SQL welcome banner file.
`/usr/lib/nls/$LANG/isqlwel`	Localized welcome banner file.

SEE ALSO

ALLBASE/ISQL Reference Manual.

NAME

join - relational database operator

SYNOPSIS

`join` [*options*] *file1 file2*

DESCRIPTION

`join` forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* or *file2* is `-`, the standard input is used.

file1 and *file2* must be sorted in increasing collating sequence (see Environment Variables below) on the fields on which they are to be joined; normally the first in each line.

The output contains one line for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field followed by the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are space, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a space.

Some of the below options use the argument *n*. This argument should be a `1` or a `2` referring to either *file1* or *file2*, respectively.

Options

`-a` *n*	In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is `1` or `2`.
`-e` *s*	Replace empty output fields by string *s*.
`-j` *m*	Join on field *m* of both files. The argument *m* must be delimited by space characters. This option and the following two are provided for backward compatibility. Use of the `-1` and `-2` options (see below) is recommended for portability.
`-j1` *m*	Join on field *m* of *file1*.
`-j2` m	Join on field *m* of *file2*.
`-o` *list*	Each output line comprises the fields specified in *list*, each element of which has the form *n*.*m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
`-t` *c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.
`-v` *file_number*	Instead of the default output, produce a line only for each unpairable line in *file_number*, where *file_number* is `1` or `2`.
`-1` f	Join on field *f* of file 1. Fields are numbered starting with 1.
`-2` *f*	Join on field *f* of file 2. Fields are numbered starting with 1.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence `join` expects from input files.

`LC_CTYPE` determines the alternative blank character as an input field separator, and the interpretation of data within files as single and/or multi-byte characters. `LC_CTYPE` also determines whether the separator defined through the `-t` option is a single- or multi-byte character.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `join` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

EXAMPLES

The following command line joins the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name, and the login directory. It is assumed that the files have been sorted in the collating sequence defined by the `LC_COLLATE` or `LANG` environment variable on the group ID fields.

```
join -1 4 -2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

The following command produces an output consisting all possible combinations of lines that have identical first fields in the two sorted files *sf1* and *sf2*, with each line consisting of the first and third fields from `sorted_file1` and the second and fourth fields from `sorted_file2`:

```
join -j1 1 -j2 1 -o 1.1, 2.2, 1.3, 2.4 sorted_file1 sorted_file2
```

WARNINGS

With default field separation, the collating sequence is that of `sort -b`; with `-t`, the sequence is that of a plain sort.

The conventions of `join`, `sort`, `comm`, `uniq`, and `awk` are incongruous.

Numeric filenames may cause conflict when the `-o` option is used immediately before listing filenames.

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

STANDARDS CONFORMANCE

`join`: SVID2, XPG2, XPG3, POSIX.2

NAME

kermit - kermit file transfer

SYNOPSIS

kermit [*options*] [*file*]

DESCRIPTION

kermit is a file transfer program for moving files between many machines of different operating systems and architectures.

Arguments are optional. If **kermit** is executed without arguments, it enters command mode. Otherwise, **kermit** parses the arguments from the command line and interprets them.

The following terms are used in the command descriptions that follow:

fn	HP-UX file specification, possibly containing the "wildcard" characters ***** or **?** (***** matches all character strings, **?** matches any single character).
fn1	HP-UX file specification which must not contain ***** or **?**.
rfn	Remote file specification in the remote system's own syntax. May denote a single file or a group of files.
rfn1	Remote file specification which should denote only a single file.
n	Decimal number between 0 and 94.
c	Decimal number between 0 and 127 representing the value of an ASCII character.
cc	Decimal number ranging from 0 through 31 or exactly 127, which represents the value of an ASCII control character.
[ ]	Any field in square braces is optional.
{*x* | *y* | *z*}	Alternatives are listed in curly braces.

kermit command line options can specify either actions or settings. If **kermit** is invoked with a command line that specifies no actions, it issues a prompt and begins interactive dialogue. Action options specify either protocol transactions or terminal connection.

The command line must contain no more than one protocol action option.

Action Options

kermit recognizes the following action options:

-s *fn* Send the specified file or files. If *fn* contains wildcard (meta) characters the HP-UX shell expands it into a list. If *fn* is **-**, **kermit** sends from standard input, which must come from a file:

```
kermit -s - < foo.bar
```

or a pipeline:

```
ls -l | kermit -s -
```

This mechanism cannot be used to send terminal type-in. To send a file whose name is - precede the file name with a path name, as in

```
kermit -s ./-
```

-r
Receive a file or files. Wait passively for files to arrive.

-k
Receive (passively) a file or files, sending them to standard output. This option can be used in several ways:

Display incoming files on your terminal (can be used only in "local mode" — see below):

```
          kermit -k
```

Send incoming file or files to a named file. If more than one file arrives, all are concatenated together into the single file *fn1*:

```
     kermit -k > fn1
```

Pipe incoming data (single or multiple files) to the indicated command:

```
     kermit -k | command
```

For example:

```
     kermit -k | sort > sorted.stuff
```

-a *fn1* If you have specified a file transfer option, you can also specify an alternate name for a single ~~file by using the **-a** option. For example,~~

```
          kermit -s foo -a bar
```

sends the file *foo* telling the receiver that its name is *bar*. If more than one file arrives or is sent, only the first file is affected by the **-a** option. To store the first incoming file under the name **baz**:

```
     kermit -ra baz
```

-x
Begin server operation. Can be used in either local or remote mode.

Setting Options

kermit recognizes the following setting options:

-l *dev* Line – Specify a terminal line to use for file transfer and terminal connection, as in

```
          kermit -l /dev/tty1p4
```

The **-l** option places **kermit** in "local" mode (see below).

The following setting options are sometimes helpful when setting up successful communication with the remote system:

-b *n* Set Baud Rate – Specify the baud rate for the line given in the **-l** option, as in

```
          kermit -l /dev/tty1p4 -b 9600
```

This option should always be included with the **-l** option, since the speed of an external line is not necessarily what you expect.

-p *x*
Set Parity – **e**, **o**, **m**, **s**, **n** (even, odd, mark, space, or none). If parity is other than none, the 8th-bit prefixing mechanism is used for transferring 8-bit binary data, provided the opposite **kermit** agrees. The default parity is none.

-t
Specifies half duplex, line turnaround with XON as the handshake character.

The following setting options can be used only with a **kermit** which is local — either by default or because the **-l** option has been specified:

-g *rfn* Actively request a remote server to send the named file or files; *rfn* is a file specification in the remote host's own syntax. If *rfn* happens to contain any special shell characters, such as *, they must be quoted, as in:

```
          kermit -g x\*.\?
```

-f
Send a "finish" command to a remote server.

-c
Establish a terminal connection over the specified or default communication line before any protocol

transaction takes place. Control can be returned to the local system by typing the escape character (normally Control-Backslash) followed by the letter `c`.

`-n`
Like `-c`, but connect after a protocol transaction takes place. `-n` and `-c` are used in the following manner:

Connect `kermit` through `/dev/tty0p4` at 9600 baud:

```
kermit -l /dev/tty0p4 -b 9600 -c
```

Tell `kermit` to wait for a file to be received, then connect through `/dev/tty0p4` at 9600 baud:

```
kermit -l /dev/tty0p4 -b 9600 -rn
```

On a timesharing system, the `-l` and `-b` options must also be included with the `-r`,`-k`, or `-s` options if the other `kermit` is on a remote system.

If `kermit` is in local mode, the standard output is continuously updated to show the progress of the file transfer. A dot (`.`) is printed for every four data packets; other packets are shown by type (e.g. `S` for Send-Init), `T` is printed when a timeout occurs, and `%` is printed for each retransmission. In addition, you can type certain "interrupt" commands during file transfer. These commands must be preceded by the escape character (by default, Ctrl-\).

Ctrl-F:	Interrupt the current File, and go on to the next (if any).
Ctrl-B:	Interrupt the entire Batch of files, terminate the transaction.
Ctrl-R:	Resend the current packet
Ctrl-A:	Display a status report for the current transaction.

Other Command-Line Options
Several other command-line options are provided:

`-i`	Send or receive files exactly "as is" with no conversions. This option is necessary for transmitting binary files. It can also be used to slightly boost efficiency in HP-UX-to-HP-UX transfers of text files by eliminating CR-LF/newline conversion.
`-w`	Write-Protect – Avoid filename collisions for incoming files.
`-q`	Quiet – Suppress screen update during file transfer; for example to allow a file transfer to proceed in the background.
`-d`	Debug – Record debugging information in the file `debug.log` in the current directory.
`-h`	Help – Display a brief synopsis of the command line options.

Local vs. Remote Mode
`kermit` is "local" if it is running on a PC or workstation that you are using directly, or if it is running on a multiuser system and transferring files over an external communication line – not your job's controlling terminal or console. `kermit` is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line connected to your PC or workstation.

If you are running `kermit` on a PC, it is in local mode by default, with the "back port" designated for file transfer and terminal connection. If you are running `kermit` on a multiuser (timesharing) system, it is in remote mode unless you explicitly point it at an external line for file transfer or terminal connection with the `-l` option.

Interactive operation
`kermit`'s interactive command prompt is "C-Kermit>". In response to this prompt, you can type any valid command. `kermit` executes the command then prompts you for another command. The process continues until you instruct the program to terminate.

Commands begin with a keyword, normally a verb, such as "send". You can omit trailing characters from any keyword, as long as you specify sufficient characters to distinguish it from any other keyword valid in that field. Certain commonly-used keywords (such as `send`, `receive`, `connect`) have special non-unique abbreviations (`s` for "send", `r` for "receive", `c` for "connect").

Certain characters have special functions in interactive commands:

?	Question mark, typed at any point in a command, produces a message explaining what is possible or expected at that point. Depending on the context, the message may be a brief phrase, a menu of keywords, or a list of files.
ESC	(The Escape key) – Request completion of the current keyword or filename, or insertion of a default value. The result is a beep if the requested operation fails due to multiple matches.
DEL,	BS (The Delete or Rubout key, Backspace or control-H) – Delete the previous character from the command.
`^W`	(Control-W) – Erase the right-most word from the command line.
`^U`	(Control-U) – Erase the entire command.
`^R`	(Control-R) – Redisplay the current command.
SP,	TAB (Space, Horizontal tab) – Delimits fields (keywords, filenames, numbers) within a command.
CR,	LF, FF (Carriage Return, Line-feed, Form-feed) – Enters the command for execution.
\	(Backslash) – Enter any of the above characters into the command as a literal. To enter a backslash, type two backslashes in a row (\\).

You can type the editing characters (DEL, `^W`, etc.) repeatedly, to delete all the way back to the prompt. No action is performed until the command is entered by typing carriage-return, line-feed, or form-feed. If you make any mistakes , **kermit** displays an informative error message followed by a new prompt. Make liberal use of **?** and ESC to feel your way through the commands. One important command is **help** – you should use it the first time you run **kermit**.

Interactive **kermit** accepts commands from files as well as from the keyboard. When you enter interactive mode, **kermit** looks for the file **.kermrc** in your home or current directory (first it looks in the home directory, then in the current one) and executes any commands it finds there. These commands must be in interactive format, not HP-UX command-line format. A **take** command is also provided to execute commands from a file at any time during an interactive session. Command files can be nested to any reasonable depth.

Here is a brief list of **kermit** interactive commands:

! *command* Execute the HP-UX shell command *command*.

bye Terminate and log out a remote **kermit** server.

close *log_file* Close a log file. *log_file* is one of the following: debugging, packets, session, or transactions.

connect Establish a terminal connection to a remote system as if it were a local terminal to that computer. The connection is made through the device specified in the most recent **set line** command. All characters you type at your keyboard are sent out the communication line, all characters arriving at the communication port are displayed on your terminal. Current settings of speed, parity, duplex, and flow control are honored. If you have issued a **log session** command, everything you see on your terminal is also recorded to your session log. This provides a way to "capture" files from systems that do not have **kermit** programs available.

To get back to your own system, type the escape character, which is FS (control-\, ASCII 28) unless you have changed it with the **set escape** command, followed by a single character command, such as **c** for "close connection". The recognized single character commands are:

c	Close the connection.
b	Send a BREAK signal to remote machine.
0	(zero) Send a NUL character.

`s`	Give a status report about the connection.
`^e`	Send Ctrl-\ itself (or whatever you have defined the escape character to be, typed twice in a row sends one copy of it).

`cwd` *dir*
Change Working Directory. Changes the current working directory to *dir*.

`dial` *number*
Dial a telephone number. Tells the modem to dial the number *num*. `kermit` must be told what type of modem is being used (see `set` command below).

`directory` *dir* `...`
Display a directory listing.

`echo` *arg* `...`
Display arguments literally.

`exit`
Exit from the program, closing any open logs.

`finish`
Instruct a remote `kermit` server to exit, but not log out.

`get` [*file* [*dname*]]
Get files from a remote `kermit` server. *file* may contain wildcard characters. If *dname* is specified, the first incoming file is stored under that name.

Since a remote file specification (or list) might contain spaces, which normally delimit fields of a `kermit` command, an alternate form of the command is provided to allow the inbound file to be given a new name: enter `get` alone on a line, and `kermit` prompts separately for the remote and local file specifications. For example:

```
C-Kermit>get
Remote file specification:  foo
Local name to store it under:  bar
```

`help` [*topic*]
Display a help message for a given command.

`log` *log_file file_name*
Open a log file with name *file_name*. *log_file* must be one of the following: debugging, packet, session, or transaction.

`quit`
Same as `exit`.

`receive` [*dname*]
Passively wait for files to arrive. If *dname* is specified, `kermit` stores the first incoming file under that name. The `receive` command may be abbreviated to `r`.

`remote`
Issue file management commands to a remote `kermit` server. The valid *remote* commands are:

`cwd` *dir*	Change remote working directory.
`delete` *file* ...	Delete remote files.
`directory` *dir* ...	Display a listing of remote file names.
`help` [*topic*]	Request help from a remote server.
`host` *command*	Issue a command to the remote host in its own command language.
`space`	Display current disk space usage on remote system.
`type` *file*	Display a remote file on your screen.

`who` [*user*] Display who's logged in, or get information about a user.

`script` *text*
Log in on the remote system using the text provided. The login script is intended to operate similarly to UUCP `Systems` entries. A login script is a sequence of the form:

expect send [*expect send*] ...

where *expect* is a prompt or message to be issued by the remote site, and *send* is the data to return to the remote host. *send* can also be EOT to send Ctrl-D, or BREAK to send a break. Letters in *send* can be prefixed by `~` to send special characters. These are: `~b` for backspace, `~s` for space, `~q` for "?", `~n` for linefeed, `~r` for return, `~c` for don't append a return, and *~ooo* (where *ooo* are octal digits) for the octal equivalent of a character. As with some UUCP systems, sent strings are followed by `~r` unless they end with `~c`.

Only the last seven characters in each *expect* are matched. A null *expect*, e.g., `~0` or two adjacent dashes, causes a short delay. If you expect that a sequence might not arrive, as with UUCP, conditional sequences can be expressed in the form:

–send-expect [*–send-expect* [...]]

where dashed sequences are followed as long as previous expects fail.

`send` *file* [*dname*]
Send files. Send the file of files specified by *file* to the other `kermit`, which should be running as a server, or which should be given the `receive` command. Each file is sent under its own name (or converted as specified by the `set names` command). The `send` command can be abbreviated to `s` even though `s` is not a unique abbreviation for a top-level `kermit` command.

The wildcard characters `*` and `?` are accepted in *file*. If `?` is to be included, it must be prefixed by `\` to override its normal function of providing help. `*` matches any string; `?` matches any single character. Other notations for file groups, such as `[a-z]og`, are not available in interactive commands. When *file* contains wildcard characters, there is a limit to the number of files that can be matched, which varies from system to system. If you get the message `Too many files match` , try again, using a more judicious selection.

`kermit` does not skip over "invisible" files that match the file specification. HP-UX systems usually treat files whose names start with a dot (such as `.login`, `.profile`, and `.kermrc`) as invisible.

If *dname* is specified, *file* must not contain any wildcard characters and *dname* specifies the name to send it under.

Note: `kermit` sends only from the current or specified directory. It does not traverse directory trees. If the source directory contains subdirectories, they are skipped. Conversely, `kermit` does not create directories when receiving files. If you need to do create directories, pipe *tar*(1) through `kermit`. For example, on the origination system, type:

```
tar cf - /usr/src | kermit -is -
```

This causes *tar* to send the directory `/usr/src` (and all files in its subdirectories) to standard output. `kermit` receives the standard input and sends it as a binary file. On the receiving system type:

```
kermit -il /dev/tty1p3 -b 9600 -k | tar xf -
```

This causes `kermit` to receive the *tar* archive and sends it via standard output to its own copy of *tar*, which extracts from it a replica of the original directory tree.

`server`
Begin server operation. Places the local `kermit` into server mode. All further commands must arrive as valid `kermit` packets from the `kermit` on the other end of the line. The HP-UX `kermit` server can respond to the following commands:

`get`	Send files.
`send`	Receive files.
`bye`	Attempt to log out.
`finish`	Exit the server mode.
`remote directory`	Send directory listing.
`remote delete`	Remove files.
`remote cwd`	Change working directory.
`remote type`	Send files to your screen.
`remote space`	Report about `kermit` disk usage.
`remote who`	Shows who's logged in.
`remote host`	Executes an HP-UX shell command.
`remote help`	Lists these capabilities.

`set`
Set various parameters. The "set" parameters are:

`block-check { 1 | 2 | 3 }`
Level of packet error detection.

1	Single-character 6-bit checksum, folded to included the values of all bits from each character.
2	is a two-character 12-bit checksum.
3	Three-character 16-bit cyclic redundancy check (CRC).

The higher the block check, the better the error detection and correction and the higher the resulting overhead. Type `1` is most commonly used; it is supported by all `kermit` implementations, and has proven adequate in most circumstances. Types `2` or `3` would be used to advantage when transferring 8-bit binary files over noisy lines.

`delay` *n*
How long to wait before sending first packet. *n* is specified in seconds.

`duplex { full | half }`
Specify which side echoes during "connect". `full` means the other side, `half` means the local `kermit` must echo typein itself.

`escape-character` *cc*
Character to prefix "escape commands" during "connect". The escape character is also used to prefix interrupt commands during file transfers. The default value is 28 (control backslash).

`file` *param*
Set various file parameters. Valid values for *param* are:

`display { on | off }`
Normally `on`. When in local mode, display progress of file transfers to the standard output, and listen to the standard input for interruptions. If off (also settable by `-p` on the `kermit` invocation line), none of this is done.

`names { converted | literal }`
Normally `converted`, which means that outbound file names have path specifications stripped, lowercase letters converted to uppercase, tildes and extra periods changed to **X**s and an **X** inserted in front of any name that starts with period. Incoming files have uppercase letters shifted to lowercase. `literal` means none of these conversions are done. When

using `literal` naming, the sender should not use path names in the file specification unless the same path exists on the target system and is writable.

`type { binary | text }`
: Normally `text`, which means that conversion is done between HP-UX newline characters and the carriage-return/line-feed sequences required by the canonical `kermit` file transmission format, and in common use on non-HP-UX systems. `binary` means to transmit file contents without conversion. `binary` is necessary for binary file transmission and is desirable in all HP-UX-to-HP-UX transactions to cut down on overhead.

`warning { on | off }`
: Normally `off`, which means that incoming files silently overwrite existing files of the same name. When `on`, `kermit` checks whether an arriving file would overwrite an existing file. If so, it constructs a new name for the arriving file of the form *file~n*, where *file* is the name they share and *n* is a generation number. For example if `foo` exists, the new file is called `foo~1`. If `foo` and `foo~1` exist, the new file becomes `foo~2`, and so on.

`flow-control { none | xon/xoff }`
: Communication line full-duplex flow control. Normally XON/XOFF for full duplex flow control. Should be set to `none` if the other system cannot do XON/XOFF flow control.

`handshake { xon | xoff | cr | lf | bell | esc | none }`
: Normally `none`. Otherwise half duplex communication line turnaround handshaking is done, which means `kermit` does not reply to a packet until it has received the indicated handshake character or has timed out waiting for it.

`line` [*device*]
: Communication line device name. If you specify a device name, `kermit` operates in local mode and you should remember to issue any other necessary `set` commands, such as `set speed`. If you omit the device name, `kermit` reverts to its default mode of operation.

`modem-dialer { direct | hayes | ventel }`
: Type of modem-dialer on communication line. `direct` indicates either there is no dial-out modem, or that if the line requires carrier detection to open, then `set line` hangs waiting for an incoming call. `hayes` and `ventel` indicate that the subsequent `set line` will prepare for a subsequent `dial` command for Hayes and Ventel dialers, respectively.

`parity { even | odd | mark | space | none }`
: Communication line character parity. If other than `none`, `kermit` seeks to use the 8th-bit prefixing mechanism for transferring 8-bit binary data, which can be used successfully only if the other `kermit` agrees.

`prompt` [*string*]
: Change the `kermit` program's prompt. If *string* is given, the prompt is set to *string*. If *string* is omitted, the prompt reverts to the default `C-Kermit>`.

`receive` *parameter value*
: Parameters to request or inspect for incoming packets, as follows:

  `end-of-line` *cc*
  : Normally *carriage return* (15) by default.

  `packet-length` *number*
  : Maximum length packet for the other side to send; a decimal number between 10 and 94. Shorter packet lengths can be used on noisy lines, or with systems, front ends, or networks that have small buffers. The shorter the packet, the higher the overhead, but the lower the chance of a packet being corrupted by noise, and the less time to retransmit the corrupted packets.

  `timeout` *number*
  : How many seconds the other Kermit should wait for a packet before asking for retransmission.

  `pad-character` *cc*
  : Character to use for inter-packet padding. Normally `kermit` does not need to

have incoming packets preceded by pad characters. This command allows **kermit** to request the other **kermit** to use *cc* as a pad character. Default of NUL (ASCII 0). No **kermits** are known to need padding, and if one did, it would request it without your having to tell it to do so.

padding *number*
How many padding characters to request before each incoming packet.

start-of-packet *cc*
Set control character to mark beginning of incoming packets. The **kermit** packet prefix character is SOH (control-A). The only reasons it would ever be changed would be: Some piece of equipment somewhere between the two **kermit** programs cannot pass SOH, or some piece of equipment similarly placed is echoing its input. In the latter case, the recipient of such an echo can change the packet prefix for outbound packets, so that the echoed packets will be ignored. The opposite **kermit** must also be told to change the prefix for its inbound packets.

send *parameter***value**
Establish parameters for outgoing packets. This command is generally used to override negotiated values, or to establish values before negotiation takes place.

end-of-line *cc*
The ASCII character to be used as a line terminator for outbound packets if one is required by the other system. Normally *carriage return* (15) by default. This command is necessary only for systems requiring a line terminator other than carriage return.

packet-length *number*
Maximum length packet to send, decimal number, between 10 and 94, decimal. Shorter packet lengths can be used on noisy lines, or with systems, front ends, or networks that have small buffers. The shorter the packet, the higher the overhead, but the lower the chance of a packet being corrupted by noise, and the less time to retransmit the corrupted packets. This command can be used to specify a shorter length than the one requested by the other **kermit**, but not a longer one.

timeout *number*
How many seconds to wait for a packet before asking for trying again. A value of zero means do not time out (wait forever).

pad-character *cc*
Character to use for inter-packet padding. Default of NUL (ASCII 0).

padding *number*
How many padding characters to send before a packet. Defaults are no padding.

start-of-packet *cc*
Set control character to mark start of packets. The **kermit** packet prefix character is SOH (control-A). The only reasons it would ever be changed would be: Some piece of equipment somewhere between the two **kermit** programs does not pass SOH; or some piece of equipment similarly placed is echoing its input. The opposite **kermit** must also be told to change the prefix for its inbound packets.

speed { 0 | 110 | 300 | 600 | 1200 | 1800 | 2400 | 4800 | 9600 }
Set communication line speed. This command cannot be used to change the speed of your own console terminal. Normally, you must use this command after a **set line** before you can use the line.

show { parameters | versions }
In response to **show parameters**, **kermit** displays the current values of all **set** parameters described above. In response to **show versions**, **kermit** displays the version numbers and dates of all its interval modules.

space
Display current disk space usage.

statistics
Display statistics about most recent **kermit** protocol transaction including file and communication line

I/O, as well as what encoding options were in effect (such as 8th-bit prefixing, repeat-count compression, etc.).

`take` *file*
Execute commands from *file*. *file* can contain any interactive `kermit` commands, including `take`. Command files can be nested to any reasonable depth. The `echo` command can be used within command files to issue greetings, announce progress, etc.

Command files are in exactly the same syntax as interactive commands. Note that this implies that if you want to include special characters such as `?` or `\` you must quote these characters the same way as when typing interactive commands.

Command files can be used in lieu of command macros (which have not been implemented in this version of `kermit`). For example, if you commonly connect to a system called `B` that is connected to `ttylp3` at 4800 baud, you could create a file called `b` containing the commands:

```
set line /dev/ttylp3
set speed 4800
echo Connecting to System B...
connect
```

then simply type `take b` (or `t b` since no other commands begin with the letter `t`) whenever you want to connect to system B.

An implicit `take` command is executed upon your `.kermrc` file upon `kermit`'s initial entry into interactive dialog. The `.kermrc` file should contain `set` or other commands you want to be in effect at all times. For instance, you might want to override the default action when incoming files have the same names as existing files. In that case, put the command

```
set file warning on
```

in your `.kermrc` file.

DIAGNOSTICS
The diagnostics produced by `kermit` itself are intended to be self-explanatory.

WARNINGS
File renaming: When filename collision avoidance (`set file warning`) is selected, `kermit` constructs unique names by appending a generation number to the end of the file name. Currently, no checking is done to ensure that the result is still within the maximum length for a file name. Consequently, if the existing file name is already the maximum allowable length, the file can be overwritten.

UUCP line locking: `kermit` locks lines when it first opens them, to prevent UUCP and multiuser conflicts. This occurs either when a `set line` is issued, or if the `-l` argument is used when the first `dial`, `connect`, or protocol operation occurs. The lock is released if another `set line` is issued, or if the program quits, exits, or is terminated by SIGINT. If a user connects and returns to the shell command level (for example to initiate `kermit` by piped commands) the line lock is released when returning to the shell. Locking is not needed or used if communication occur over the local terminal line (e.g. `/dev/tty`). In that case, there is no difficulty with piped operations releasing locks and lines.

Removing stale lock files: For various reasons, lock files sometimes get left after UUCP or `kermit` activities terminate. The most common reason is that the UUCP or `kermit` activity was killed by a shell command. UUCP supports a function called `uuclean` which is customarily used to remove these files after a predetermined age. If in doubt about a lock file on the dial-out line you need, contact your system administrator.

Modem controls: If a connection is made over a communication line (rather than on the controlling terminal line), and that line has modem controls (e.g. data terminal ready and carrier detect implementations), returning to the shell disconnects the conversation. In such cases, use interactive mode commands to avoid use of piped shell-level operations.

Resetting the terminal after abnormal termination of kill: When `kermit` terminates abnormally (such as when a `kill` command is issued by the operator – see *kill*(1)), it may be necessary to reset the terminal state. If commands do not seem to be accepted at the shell prompt, try typing `^J` Reset `^J` to take the terminal out of raw mode if it was stuck there.

AUTHOR

The `kermit` protocol was developed by Columbia University. `kermit` is available for many systems for a nominal fee from Columbia and various user groups.

FILES

`/usr/bin/kermit`
`$HOME/.kermrc`

SEE ALSO

cu(1), tar(1), uucp(1).

NAME

keysh - context-sensitive softkey shell

SYNOPSIS

keysh

DESCRIPTION

keysh is an extension of the standard Korn-shell (for a description of the basic Korn-shell functionality, see *ksh*(1)).

keysh uses hierarchical softkey menus and context-sensitive help to aid users in building command-lines, combining the power of the Korn-shell with the ease-of-use of a menu system.

And **keysh** is entirely data-driven, allowing its menus and help to be easily extended as needed.

Note that during **keysh** invocation, the environment variable **$TERM** must specify the terminal type, as defined in the *terminfo*(4) database (see ENVIRONMENT VARIABLES below).

COMMAND ENTRY

keysh continually parses the command-line and always presents the user with an appropriate set of *current choices* on the softkey labels.

The user can select these softkeys to create readable *softkey commands* on the command-line. **keysh** automatically translates these softkey commands into equivalent *HP-UX commands* prior to executing them.

Alternatively, the user can ignore the softkeys altogether in favor of entering the traditional HP-UX commands directly, as when using the Korn-shell.

During command entry, **keysh** ordinarily displays a *status-line* near the bottom of the screen. This status-line contains information such as the host name, current directory, and time and date.

Whenever the user *must* perform an action to complete the current softkey command, **keysh** temporarily displays a *prompt message* in place of the status-line. This message briefly describes the required action.

Softkey Types

keysh presents four basic softkey types:

--Help--	Selecting the **--Help--** softkey causes **keysh** to display help information associated with the next selected softkey, rather than actually performing its action.
--More--	If there are more current choices than there are softkeys, **keysh** breaks the choices into banks and displays a special **--More--** softkey along with the first bank. Selecting the **--More--** softkey causes **keysh** to display the next bank of softkeys in sequence, eventually cycling back to the first.
<param>	*parameter* softkeys are displayed as a name enclosed between a pair of less-than and greater-than symbols. They indicate that the user-supplied text (such as a file name) should be entered into the command-line at that point, rather than actually selecting the softkey. (Actually selecting the softkey only causes **keysh** to display a hint message on the status line; the command-line remains unchanged.)
option	All other softkeys are *option* softkeys that can be used to insert the corresponding command or option name into the command-line.

Softkeys can be selected from left to right.

Editing The Command-Line

keysh supports the normal Korn-shell command-line editing modes. In addition, **keysh** also recognizes the cursor movement and editing keys found on most terminals, as defined in the *terminfo*(4) database. These include:

<Clear display>	Clear the screen and command-line. If the screen is scrolled, clear only from the cursor position to the end of scrolling memory.
<Clear line>	Clear from the cursor position to the end of the command-line.
<Delete line>	Clear the entire command-line.
<Insert line>	Translate any softkey commands in the current command-line and then edit the result.

<Delete char>	Delete the character under the cursor.
<Insert char>	Toggle between insert and overwrite modes.
<Up/Down arrow>	Recall the previous/next command from the history buffer.
<Left/Right arrow>	Move the cursor left/right.
<Home up/down>	Move the cursor to the beginning/end of the command-line.
<Tab>	If no <Insert line> key is present, perform the <Insert line> function (see above). Otherwise, if no **--Help--** softkey is present, perform the **--Help--** function (also see above). Otherwise, perform the normal tab function.
<Backtab>	Move the cursor to the beginning of the previous word.
<Ctrl-L>	Redraw the lower lines of the screen and restore any necessary terminal modes.

Visible Softkey Commands

If the **visibles** configuration option is enabled (see CONFIGURATION below), **keysh** displays a list of configured softkey commands on the softkey labels whenever it is expecting a new command. This is the the top-level softkey menu.

If the user selects one of these softkey commands, **keysh** inserts its command name into the command-line then displays a sub-menu listing the command's major parameters and/or options.

The user can then (from left to right) select option softkeys and/or enter text in place of parameter softkeys. **keysh** automatically navigates the hierarchical softkey menu, always presenting the user with an appropriate set of current choices on the softkey labels.

Note that **keysh** automatically redisplays the top-level softkey menu when it detects that a command separator (such as a pipe or semi-colon) has been entered, thus allowing the user to use softkeys for subsequent commands on the command-line as well as the first.

Invisible Softkey Commands

If the **invisibles** configuration option is enabled (see CONFIGURATION below) and **keysh** recognizes a traditional HP-UX command being entered, it gives the user one last chance to use the softkeys by again presenting an appropriate set of current choices on the softkey labels. As with the top-level softkey menu options, the user can choose to ignore the softkeys in favor of entering the traditional HP-UX options directly.

Backup Softkeys

If the **backups** configuration option is enabled (see CONFIGURATION below), **keysh** displays the *backup softkeys* and programs the terminal function keys appropriately whenever it has no other softkeys to display (such as when a command is running). These provide the traditional static softkey control which many users may be used to.

Traditional HP-UX Commands

If the user enters a traditional HP-UX command when **keysh** is displaying its top-level softkey menu, **keysh** simply displays the backup softkeys and allows the user to proceed.

If **keysh** subsequently detects a command separator, it again redisplays the top-level softkey menu.

Softkey Command Syntax Errors

Many softkey commands present the user with a set of softkey options from which exactly one (or at least one) *must* be selected. If the user fails to do this, **keysh** treats it as a syntax error, displaying an error message and not accepting the command until the error has been corrected.

Similarly, many softkey commands require that the user enter one or more softkey parameters before the command is semantically complete. If the user fails to do this, **keysh** again treats it as a syntax error.

Softkey Command Redirections

The user can append redirection symbols (such as a less-than or greater-than symbol followed by a file name) following a softkey command. These are appended *verbatim* to the translated HP-UX command.

USING KEYSH WITH TERMINAL SESSION MANAGER

When operating under the Terminal Session Manager (see *tsm*(1)), **keysh** displays the **tsm** softkeys instead of the backup softkeys. If desired, this interaction can be overridden by setting the **$KEYTSM** environment variable (see ENVIRONMENT VARIABLES below).

When operating under `tsm`, **`keysh`** also automatically displays the `tsm` window number in the status-line.

CONFIGURATION

All **`keysh`** configuration functions are accessed through the top-level **`Keysh_config`** softkey command or `kc` built-in command. These functions include:

- adding, placing, and deleting softkeys,
- specifying backup softkeys,
- selecting global options,
- selecting status-line items,
- restarting keysh,
- writing configuration changes, and
- undoing other configuration changes.

Each time the user changes **`keysh`**'s configuration, **`keysh`** automatically updates the user's **`$HOME/.keyshrc`** file. Upon subsequent invocations, **`keysh`** automatically reconfigures itself as configured previously.

Adding, Placing, And Deleting Softkeys

Any of the standard softkeys (see STANDARD SOFTKEY DEFINITIONS below) can be added to the top-level softkey menu using the **`kc softkey add`** command. If desired, an alternate softkey label may be specified (usually in place of a cryptic HP-UX command name) using the **`with_label`** option.

By default, added softkeys are placed at the end of the last **`--More--`** bank of the top-level softkey menu. This placement can be overridden using the **`and_place`** option of the **`kc softkey add`** command or using the **`kc softkey move`** command.

In addition to the standard softkeys, custom softkeys can also be added from custom softkey files using the **`from_user`** or **`from_file`** options. For a description of the softkey file format, see *softkeys*(4).

Note that any time a softkey is added from a particular softkey file, all of the remaining softkeys from that file are automatically loaded for use as invisible softkey commands. All softkeys from a file can also be loaded for use as invisible softkey commands using the **`kc softkey add invisibles`** command.

Any of the softkeys in the top-level softkey menu can be deleted using the **`kc softkey delete`** command.

Specifying Backup Softkeys

Backup softkeys are typically specified in the user's **`$HOME/.softkeys`** file. The basic backup softkey definition line resembles:

`backup softkey "`*<softkey>***`" literal " `***<string>***`";`**

Where *<softkey>* is the softkey label to display and *<string>* is the text string to program the terminal function key with. A maximum of eight backup softkeys can be specified.

Note that backup softkeys must be explicitly added using the **`kc softkey add backups`** command before **`keysh`** can program them.

Selecting Global Options

Various global options can be configured using the **`kc option`** command, including:

`backups`	Enable or disable the programming of the backup softkeys.
`help`	Enable or disable the **`--Help--`** softkey.
`invisibles`	Enable or disable the recognition of invisible softkey commands.
`prompts`	Enable or disable the automatic generation of prompt messages. When enabled, **`keysh`** displays a prompt message whenever the user *must* perform an action to complete the current softkey command. This message briefly describes the required action.
`selectors`	Enable or disable the use of keyboard selectors. When enabled, **`keysh`** displays an upper-case selector character in each softkey label. Typing the unquoted (upper-case) character selects the softkey just as if its corresponding function key had been pressed. Quoting the selector character in any way restores its traditional meaning.

Selector keys are intended to be used on terminals that do not support a sufficient number of softkeys.

`translations`
: Enable or disable the display of HP-UX command translations.

`visibles` Enable or disable the presentation and recognition of visible softkey commands.

Selecting Status-Line Items

Various information items can be configured into the status-line displayed at the bottom of the screen using the `kc status_line` command, including:

`host_name`	The host name.
`user_name`	The user name.
`current_dir`	The current directory.
`mail_status`	The mail status based on the `$MAIL` environment variable (i.e., `No mail`, `You have mail`, or `You have new mail`).
`date`	The date.
`time`	The time of day.

In addition, the `$KEYSH` environment variable, if set, is always displayed first in the status-line.

Restarting Keysh

`keysh` can be forced to reread the `$HOME/.keyshrc` file with the `kc restart` command. This command is typically used to update a `keysh` to a new configuration specified in another window.

`keysh` can also be forced to remove the `$HOME/.keyshrc` file and restart from the default user configuration with the `kc restart default` command.

Writing Configuration Changes

`keysh` can be forced to rewrite the `$HOME/.keyshrc` file with the `kc write` command.

Undoing Other Configuration Changes

`keysh` can also be forced to rewrite the `$HOME/.keyshrc` file with its original contents, thus undoing all configuration changes made since `keysh` was invoked, using the `kc undo` command.

Scaling Keysh Functionalities

`keysh` provides a scalable set of functionalities which can be tailored to suit personal preferences.

For users who are familiar with the HP-UX command names (though not necessarily with the command options) or for users who prefer to usually have the `tsm` softkeys visible, the command `kc options visibles off` prevents `keysh` from displaying its top-level softkey menu while waiting for a command; instead, it displays the backup softkeys or `tsm` softkeys, as appropriate. (`keysh` start-up time can then be decreased significantly by editing the `$HOME/.keyshrc` file and removing the lines which add visible softkeys.)

For users who are also familiar with the HP-UX command options, the command `kc options invisibles off` prevents `keysh` from displaying softkey menus for invisible softkey commands, also.

And for users who have no need for the backup softkeys, the command `kc options backups off` prevents `keysh` from ever programming the backup softkeys.

Note that if `visibles`, `invisibles`, and `backups` are all turned off, `keysh` performs *no* softkey processing at all. `keysh` effectively transforms into a Korn-shell which displays a status-line and recognizes the cursor movement and editing keys.

EXAMPLES

To add the `od` (see *od*(1)) softkey to the end of the top-level softkey menu and label it `Octal_dump`,

```
kc softkey add od with_label Octal_dump
```

To add the *paste*(1) softkey to the beginning of the top-level softkey menu and label it `Paste`,

```
kc softkey add paste and_place as_first_softkey
```

To add the custom emacs softkey from the file **~rpt/.softkeys** to the top-level softkey menu immediately before the **ls** (see *ls*(1)) softkey,

```
kc softkey add emacs from_user rpt and_place before_softkey ls
```

To add all invisible softkeys from the file **~rpt/.softkeys**,

```
kc softkey add invisibles from_user rpt
```

To add the backup softkeys from the file **$HOME/.softkeys**,

```
kc softkey add backups
```

To delete the **Edit_file** softkey from the top-level softkey menu,

```
kc softkey delete Edit_file
```

To disable the **--Help--** softkey,

```
kc options help off
```

To configure the user name into the status-line,

```
kc status_line user_name on
```

To configure the exit-value of the last command executed into the status-line,

```
KEYSH="\${?#0}"
```

To list the ten largest files in the current directory,

```
ls long_format | Sort_lines numerically reverse_order
starting_at_field 5 | head
```

STANDARD SOFTKEY DEFINITIONS

Copy_files, **Move_files**, **Print_files**, **Set_file_attribs**, **Switch**.

adjust, **ar**, **bdf**, **cal**, **cancel**, **cat**, **cd**, **cdb**, **chatr**, **chgrp**, **chmod**, **chown**, **cmp**, **col**, **comm**, **cpio**, **cut**, **dd**, **df**, **diff**, **dircmp**, **disable**, **du**, **elm**, **enable**, **exit**, **find**, **fold**, **grep**, **head**, **jobs**, **kill**, **lp**, **lpstat**, **ls**, **mailx**, **make**, **man**, **mkdir**, **more**, **nm**, **nroff**, **od**, **paste**, **pg**, **pr**, **ps**, **remsh**, **rlogin**, **rm**, **rmdir**, **sdiff**, **set**, **shar**, **sort**, **tail**, **tar**, **tcio**, **tee**, **touch**, **tr**, **umask**, **uname**, **vi**, **wc**, **who**, **write**, **xd**, **xdb**.

ENVIRONMENT VARIABLES

TERM	Specifies the terminal type, as defined in the *terminfo*(4) database. This variable must be either part of **keysh**'s invocation environment or it must be set within one of the standard Korn-shell start-up files.
COLUMNS	Specifies the number of columns in the terminal screen if different than the *terminfo*(4) default.
LINES	Specifies the number of lines in the terminal screen if not the same as the *terminfo*(4) default.
PAGER	Specifies the preferred pager to be used to display help. The default is **more** (see **more**(1)).
TZ	Specifies the time-zone to be used for time and date representations on the status-line. The default is **american**.
KEYBEL	Specifies the character sequence sent to the terminal by **keysh** to ring the bell. The default is **^G**.
KEYENV	Specifies an alternate **keysh** configuration file. The default is **$HOME/.keyshrc**.
KEYESC	Specifies the maximum allowable delay between characters (in milliseconds) if they are to be treated as part of a terminal escape sequence. The default is 350 ms.
KEYKSH	If set, specifies that **keysh** should mimic the behavior of the Korn-shell as closely as possible. No softkeys or status-line are displayed. This mode is particularly useful over slow modem lines.

`KEYLOC`	If set, specifies that **`keysh`** should leave the terminal keypad in local mode while commands are being entered. This mimics the behavior of the Korn-shell.
`KEYPS1`	If set, specifies that **`keysh`** should not reset the initial values of **`$PS1`**, **`$PS2`**, and **`$PS3`**. Note that **`$PS1`** must be a constant character string in order for **`keysh`** to recognize it and provide subsequent softkey assistance.
`KEYSH`	Specifies arbitrary text to be included in the **`keysh`** status-line.
`KEYSIM`	If set, specifies that **`keysh`** should always simulate softkey labels and not use the built-in labels on HP terminals.
`KEYTSM`	If set, specifies that **`keysh`** should *not* use the **`tsm`** softkeys when **`tsm`** is running. In this case, the user can either use the **`tsm hotkey`**, the backup softkeys, or the **`Switch`** softkey command (see STANDARD SOFTKEY DEFINITIONS above) to switch **`tsm`** windows.

KSH DIFFERENCES

`keysh` is an extension of *ksh*(1) with the following exceptions:

Screen Updates

`keysh` optimizes its display output to take advantage of available terminal capabilities. Unlike the Korn-shell which often has to redraw large portions of the command-line, **`keysh`** can simply insert or delete characters at the appropriate screen position.

This makes **`keysh`** significantly faster over slow modem lines, especially if the **`$KEYKSH`** environment variable is set (see ENVIRONMENT VARIABLES above).

Emacs-Mode Editing

The new **`<ESC>v`** command performs the function of the vi-mode **`v`** command.

An initial **`^N`** command recalls the history line *following* the history line executed as the previous command. This provides an easy mechanism to repeat a sequence of history commands.

`gmacs` editing mode is not supported; **`emacs`** editing mode follows the GNU emacs (18.54) definition of **`^T`**.

The **`^@`** and <ESC>**`n`** **`^K`** commands are not supported.

The **`M-`**<*letter*> and **`M-]`**<*letter*> alias functions are not supported (in lieu of true softkey support).

Vi-Mode Editing

The new `o` command performs the function of the emacs-mode **`^O`** command.

An initial `j` command recalls the history line *following* the history line executed as the previous command. This provides an easy mechanism to repeat a sequence of history commands.

The `|` command is not supported.

The @<*letter*> alias function is not supported (in lieu of true softkey support).

The `u` command performs an emacs-style nested undo; `u`<space> performs a traditional vi-style undo.

WARNINGS

`keysh` requires that the **`$TERM`** environment variable be set appropriately in your **`$HOME/.profile`** file. It also requires that **`$LINES`** and **`$COLUMNS`** be set appropriately if running on a non-standard size terminal. Otherwise, an error message or a garbled screen display results.

`keysh` requires that option softkeys be selected from left to right. When editing a command-line, it is possible to back up and insert a softkey out-of-order -- resulting in a command error.

`keysh` initializes **`$PS1`**, **`$PS2`**, and **`$PS3`** and types them *read-only* — do not change them. Instead, use **`$KEYSH`** to display additional status information.

`keysh` normally maintains the **`$HOME/.keyshrc`** file without user intervention; however, start-up errors may occasionally occur and persist. In this case, either execute the command **`kc restart default`** (to remove the file and revert to the default user configuration) or execute the command **`kc write`** (to rewrite the file with the current configuration).

`keysh` assumes that HP-UX commands are not heavily aliased; otherwise unexpected command translations may occur.

keysh neglects the effects of the Korn-shell expansion mechanisms when counting command-line parameters, causing it to occasionally underestimate the true number of parameters specified. The <ESC>* emacs-mode or vi-mode editing command can often be used to pre-expand these parameters.

The <ESC>**v** emacs-mode editing command and **v** vi-mode editing command cannot be used to edit (pre-translated) softkey commands, since no subsequent command translation can occur.

Adding a large number of softkeys can cause **keysh** to overflow a 1-Mbyte Korn-shell data size limitation, causing disconcerting behavior.

keysh can only program the function keys on terminals whose *terminfo*(4) entry defines the **pfkey** capability; similarly, it can only use hardware softkey labels on terminals whose *terminfo*(4) entry defines the **pln** capability (along with specifying **lh** equal to 2).

The default value for **$KEYESC** was chosen to provide reasonable response in both local and networked environments. If keysh misinterprets quickly typed emacs-mode or vi-mode editing commands as terminal escape sequences, it may be necessary to decrease this value.

Specifying a **\n** (new-line) in the literal key sequence for a backup softkey causes undesired results on HP terminals; use a **\r** (carriage-return) instead.

keysh does not display **tsm** softkeys when simulating softkey labels.

A limited number of environment variables and arguments are exported to the pager when displaying help.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which softkeys and messages are displayed.

LC_TIME determines the format and contents of date and time strings in the status-line.

International Code Set Support

Single-byte character code sets are supported.

AUTHOR

keysh was developed by HP and AT&T.

FILES

/usr/bin/keysh	main executable
/usr/keysh/builtins	**Keysh_config** softkey definition file
/usr/keysh/$LANG/softkeys	standard softkey definitions file
/usr/keysh/$LANG/keyshrc	default user configuration file
/usr/lib/nls/$LANG/keysh.cat	message catalog
$HOME/.keyshrc	user configuration file
$HOME/.softkeys	user softkey definitions file

SEE ALSO

ksh(1), tsm(1), softkeys(4), terminfo(4).

NAME

kill - terminate a process

SYNOPSIS

`kill -s` *signal_name pid ...*
`kill -l`

Obsolescent Versions:

`kill` *- signal_number*] *pid ...*
`kill` *- signal_name*] *pid ...*

DESCRIPTION

`kill` utility sends a signal to the process(es) specified by each *pid* operand.

By default, `kill` sends signal `SIGTERM` to the specified processes. This normally kills processes that do not catch or ignore the signal.

The process number of each asynchronous process started with `&` is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using the `ps` command (see *ps*(1)).

The killed process must belong to the current user unless the current user is a user with appropriate privileges.

If a signal number or a signal name preceded by `-` is given as the first argument, that signal is sent instead of `SIGTERM` (see *signal*(5)). In particular, `kill -KILL` ... is a sure kill. If the first argument is a negative integer, it is interpreted as a signal number, not as a negative *pid* (process group).

A signal name can be any of the signal names listed in *signal*(5) with or without the `SIG` prefix. In addition, `SIGNULL` is recognized and represents the signal value `0`. Uppercase and lowercase letters in signal names are treated as equivalent.

For each *pid* operand, `kill` performs actions equivalent to the `kill()` function called with the following arguments:

- The *pid* argument to the `kill()` function is taken from the *pid* argument to the `kill` command. For example, if pid `0` is specified, all processes in the process group are signaled.
- The *sig* argument to `kill()` is the value specified by the `-s` option, *-signal_number* option, the *-signal_name* option, or by `SIGTERM` if none of these options is specified.

OPTIONS

`kill` recognizes the following options:

`-l` (ell) Write all values of signal_name supported by the implementation. No signals are sent with this option. When `-l` option is specified, the symbolic name of each signal is written in the following format:

`%s%c`, *signal_name*, *separator*

where *signal_name* is in uppercase without the `SIG` prefix, and the separator is either a new-line or a space character. For the last signal written, *separator* is a new-line character.

`-s`*signal_name*
Specify the signal to send. Values of `signal_name` are recognized in a uppercase/lowercase-independent fashion, without the `SIG` prefix. These values can be obtained by using the `-l` option. In addition, the symbolic name `0` is recognized, representing the signal value zero. The corresponding signal is sent instead of `SIGTERM`.

-signal_name
(Obsolescent) Equivalent to `-s`*signal_name*.

-signal_number
(Obsolescent) Specify a non-negative decimal integer, signal_number, representing the signal to be used instead of `SIGTERM`, as the sig argument in the effective call to `kill()`:

signal_number	*sig* **Value**
0	`0`
1	`SIGHUP`
2	`SIGINT`
3	`SIGQUIT`
6	`SIGABRT`
9	`SIGKILL`
4	`SIGALRM`
5	`SIGTERM`

RETURN VALUE

Upon completion, `kill` returns with one of the following values:

0 At least one matching process was found for each *pid* operand, and the specified signal was successfully processed for at least one matching process.

>0 An error occurred.

EXAMPLES

The command:

```
kill 6135
```

signals process number 6135 to terminate (assuming you own the process). This gives the process an opportunity to exit gracefully (removing temporary files, etc.).

The commands:

```
kill -9 6135
kill -SIGKILL 6135
kill -KILL 6135
kill -s KILL 6135
kill -s kill 6135
```

terminate process number 6135 by sending a `SIGKILL` signal to the process (assuming you own the process). This tells the kernel to remove the process immediately.

WARNINGS

If a process hangs during some operation (such as I/O) so that it is never scheduled, it cannot die until it is allowed to run. Thus, such a process may never go away after the kill. Similarly, defunct processes (see *ps*(1)) may have already finished executing, but remain on the system until their parent reaps them (see *wait*(2)). Using kill to send signals to them has no effect.

Some non-HP-UX implementations provide `kill` only as a shell built-in utility.

SEE ALSO

csh(1), ksh(1), ps(1), sh-posix(1), sh(1), kill(2), wait(2), signal(5).

STANDARDS CONFORMANCE

`kill`: SVID2, XPG2, XPG3, POSIX.2

NAME

ksh, rksh - shell, the standard/restricted command programming language

SYNOPSIS

ksh [±aefhikmnoprstuvx] [±o *option*] ... [- c *string*] [*arg* ...]
rksh [±aefhikmnoprstuvx] [±o *option*] ... [- c *string*] [*arg* ...]

DESCRIPTION

ksh is a command programming language that executes commands read from a terminal or a file. **rksh** is a restricted version of the command interpreter **ksh**, used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invoking ksh* and *Special Commands* sections later in this entry for details about command line options and arguments, particularly the **set** command.

Definitions

metacharacter
One of the following characters:

; & ( ) | < > *new-line space tab*

blank	A tab or space character.
identifier	A sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for **functions** and **named parameters**.
word	A sequence of characters separated by one or more non-quoted metacharacters .
command	A sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action, either directly or by invoking separate utilities.

special command
A command that is carried out by the shell without creating a separate process. Often called "built-in commands". Except for documented side effects, most special commands can be implemented as separate utilities.

The # character is interpreted as the beginning of a comment. See *Quoting* below.

Commands

A **simple-command** is a sequence of blank-separated words that can be preceded by a parameter assignment list. (See *Environment* below). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The **value** of a simple-command is its exit status if it terminates normally, or (octal) 200+**status** if it terminates abnormally (see *signal*(5) for a list of status values).

A **pipeline** is a sequence of one or more **commands** separated by |. The standard output of each command except the last is connected by a pipe (see *pipe*(2)) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command in the pipeline.

A **list** is a sequence of one or more pipelines separated by ;, &, &&, or | |, and optionally terminated by ;, &, or |&. Of these five symbols, ;, &, and |& have equal precedence. && and | | have a higher but also equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (that is, the shell does not wait for that pipeline to finish). The symbol |& causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell (known as a co-process). The standard input and output of the spawned command can be written to and read from by the parent shell using the **-p** option of the special commands **read** and **print** described later. The symbol && (| |) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines can appear in a *list*, instead of semicolons, to delimit commands.

A **command** is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *identifier* [**in** *word* ...] **do** *list* **done**
Each time **for** is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, **for** executes the **do** *list* once for each positional parameter set

(see *Parameter Substitution* below). Execution ends when there are no more words in the list.

`select` *identifier* [`in` *word*...] `do` *list* `done`
: A `select` command prints on standard error (file descriptor 2), the set of *word*s, each preceded by a number. If `in` *word* ... is omitted, the positional parameters are used instead (see *Parameter Substitution* below). The `PS3` prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed *word*s, the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty, the selection list is printed again. Otherwise the value of the parameter *identifier* is set to null. The contents of the line read from standard input is saved in the parameter **REPLY**. The *list* is executed for each selection until a `break` or end-of-file (*eof*) is encountered.

`case` *word* `in` [[`(`]*pattern* [`|`*pattern*] ... `)` *list* `;;`] ... `esac`
: A `case` command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is identical to that used for file name generation (see *File Name Generation* below).

`if` *list* `then` *list* [`elif` *list* `then` *list*] ... [`else` *list*]`fi`
: The *list* following `if` is executed and, if it returns a zero exit status, the *list* following the first `then` is executed. Otherwise, the *list* following `elif` is executed and, if its value is zero, the *list* following the next `then` is executed. Failing that, the `else` *list* is executed. If no `else` *list* or `then` *list* is executed, `if` returns a zero exit status.

`while` *list* `do` *list* `done`
`until` *list* `do` *list* `done`
: A `while` command repeatedly executes the `while` *list*, and if the exit status of the last command in the list is zero, executes the `do` *list*; otherwise the loop terminates. If no commands in the `do` *list* are executed, `while` returns a zero exit status; `until` can be used in place of `while` to negate the loop termination test.

`(` *list* `)`
: Execute *list* in a separate environment. If two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

`{` *list* `;}`
: Execute *list*, but not in a separate environment. Note that `{` is a keyword and requires a trailing blank to be recognized.

`[[` *expression* `]]`
: Evaluates *expression* and returns a zero exit status when *expression* is true. See *Conditional Expressions* below, for a description of *expression*. Note that `[[` and `]]` are keywords and require blanks between them and *expression*.

`function` *identifier* `{` *list* `;}`
identifier `()` `{` *list* `;}`
: Define a function referred to by *identifier*. The body of the function is the *list* of commands between `{` and `}` (see *Functions* below).

`time` *pipeline*
: *pipeline* is executed and the elapsed time, user time, and system time are printed on standard error.

The following keywords are recognized only as the first word of a command and when not quoted:

```
    if then else elif fi case esac for while until do done { } function
    select time [[ ]]
```

Comments

A word beginning with `#` causes that word and all subsequent characters up to a new-line to be ignored.

Aliasing

The first word of each command is replaced by the text of an **alias**, if an alias for this word has been defined. An **alias name** consists of any number of characters excluding metacharacters, quoting characters, file expansion characters, parameter and command substitution characters, and `=`. The replacement string can contain any valid shell script, including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, is tested for additional aliases. If the last character of the alias value is a *blank*, the word following the alias is also checked for alias substitution. Aliases can be used to redefine special built-in commands, but cannot be used to redefine the keywords listed above. Aliases can be created, listed, and exported with the `alias` command and can be removed with the `unalias` command. Exported aliases remain in effect for subshells but must be reinitialized for separate invocations of the shell (see *Invoking ksh* below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for it to take effect, `alias` must be executed before the command referring to the alias is read.

Aliases are frequently used as a shorthand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full path name of the corresponding command. These aliases are called *tracked aliases*. The value of a tracked alias is defined the first time the identifier is read and becomes undefined each time the PATH variable is reset. These aliases remain tracked so that the next reference redefines the value. Several tracked aliases are compiled into the shell. The `-h` option of the `set` command converts each command name that is an *identifier* into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t -'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
stop='kill -STOP'
suspend='kill -STOP $$'
true=':'
type='whence -v'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, the word up to a `/` is checked to see if it matches a user name in the `/etc/passwd` file. If a match is found, the ~ and the matched login name are replaced by the login directory of the matched user. This is called a tilde substitution. If no match is found, the original text is left unchanged. A ~, alone or before a `/`, is replaced by the value of the HOME parameter. A ~ followed by a `+` or `-` is replaced by the value of the parameter PWD and OLDPWD, respectively. In addition, tilde substitution is attempted when the value of a parameter assignment begins with a ~.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign (`$(`*command*`)`) or a pair of back single quotes (accent grave) (`` ` ``*command*`` ` ``) can be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed (see *Quoting* below). The command substitution `$(cat file)` can be replaced by the equivalent but faster `$(<file)`. Command substitution of most special commands (built-ins) that do not perform I/O redirection are carried out without creating a separate process. However, command substitution of a function creates a separate process to execute the function and all commands (built-in or otherwise) in that function.

An arithmetic expression enclosed in double parenthesis preceded by a dollar sign (`$((`*expression*`))`) is replaced by the value of the arithmetic expression within the double parenthesis (see *Arithmetic Evaluation* below for a description of arithmetic expressions).

Parameter Substitution

A **parameter** is an **identifier**, one or more digits, or any of the characters `*`, `@`, `#`, `?`, `-`, `$`, and `!`. A **named parameter** (a parameter denoted by an identifier) has a value and zero or more attributes. Named parameters can be assigned values and attributes by using the `typeset` special command. Attributes supported by `ksh` are described later with the `typeset` special command. Exported parameters pass values and attributes to the environment.

The shell supports a limited one-dimensional array facility. An element of an array parameter is referenced by a subscript. A subscript is denoted by a `[` followed by an arithmetic expression (see *Arithmetic Evaluation* below) followed by a `]`. To assign values to an array, use `set -A` *name value* .... The value of all subscripts must be in the range of `0` through `1023`. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array is created if necessary. Referencing an array without a subscript is equivalent to referencing the first element.

The value of a named parameter can also be assigned by writing:

name=*value* [*name*=*value*] ...

If the `-i` integer attribute is set for *name*, the *value* is subject to arithmetic evaluation as described below.

Positional parameters, parameters denoted by a number, can be assigned values with the `set` special command. Parameter `$0` is set from argument zero when the shell is invoked.

The character `$` is used to introduce substitutable *parameters*.

`${`*parameter*`}` Substitute the value of the parameter, if any. Braces are required when *parameter* is followed by a letter, digit, or underscore that should not be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits, it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is `*` or `@` all the positional parameters, starting with `$1`, are substituted (separated by a field separator character). If an array *identifier* with subscript `*` or `@` is used, the value for each element is substituted (separated by a field separator character). The shell reads all the characters from `${` to the matching `}` as part of the same word even if it contains braces or metacharacters.

`${#`*parameter*`}` If *parameter* is `*` or `@`, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

`${#`*identifier*`[*]}` Substitute the number of elements in the array *identifier*.

`${`*parameter*`:-`*word*`}`
If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

`${`*parameter*`:=`*word*`}`
If *parameter* is not set or is null, set it to *word*; then substitute the value of the parameter. Positional parameters cannot be assigned in this way.

`${`*parameter*`:?`*word*`}`
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

`${`*parameter*`:+`*word*`}`
If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

`${`*parameter*`#`*pattern*`}`
`${`*parameter*`##`*pattern*`}`
If the shell *pattern* matches the beginning of the value of *parameter*, the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise the value of this *parameter* is substituted. In the former case, the smallest matching pattern is deleted; in the latter case, the largest matching pattern is deleted.

`${`*parameter*`%`*pattern*`}`
`${`*parameter*`%%`*pattern*`}`
If the shell *pattern* matches the end of the value of *parameter*, the value of *parameter* with the matched part is deleted; otherwise substitute the value of *parameter*. In the former, the smallest matching pattern is deleted; in the latter, the largest matching pattern is deleted.

In the above, *word* is not evaluated unless it is used as the substituted string. Thus, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:-$(pwd)}
```

If the colon (`:`) is omitted from the above expressions, the shell only checks to determine whether or not *parameter* is set.

The following parameters are set automatically by the shell:

#	The number of positional parameters in decimal.
-	Options supplied to the shell on invocation or by the `set` command.
?	The decimal value returned by the last executed command.
$	The process number of this shell.
_	Initially, the value of _ is an absolute pathname of the shell or script being executed as passed in the *environment*. Subsequently it is assigned the last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching `MAIL` file when checking for mail.
!	The process number of the last background command invoked.
COLUMNS	If this variable is set, its value is used to define the width of the edit window for the shell edit modes and for printing `select` lists. In a windowed environment, if the shell detects that the window size has changed, the shell updates the value of COLUMNS unless COLUMNS is unset.
ERRNO	The value of `errno` as set by the most recently failed system call. This value is system dependent and is intended for debugging purposes.
LINENO	The line number of the current line within the script or function being executed.
LINES	If this variable is set, the value is used to determine the column length for printing `select` lists. `select` lists print vertically until about two-thirds of `LINES` lines are filled. In a windowed environment, if the shell detects that the window size has changed, the shell updates the value of `LINES` unless `LINES` is unset.
OLDPWD	The previous working directory set by the `cd` command.
OPTARG	The value of the last option argument processed by the `getopts` special command.
OPTIND	The index of the last option argument processed by the `getopts` special command.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the `cd` command.
RANDOM	Each time this parameter is evaluated, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a numeric value to RANDOM.
REPLY	This parameter is set by the `select` statement and by the `read` special command when no arguments are supplied.
SECONDS	Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, the value returned upon reference is the value that was assigned plus the number of seconds since the assignment.

The following parameters are used by the shell:

CDPATH	The search path for the `cd` command.
EDITOR	If the value of this variable ends in `emacs`, `gmacs`, or `vi` and the `VISUAL` variable is not set, the corresponding option is turned on (see `set` in *Special Commands* below).
ENV	If this parameter is set, parameter substitution is performed on the value to generate the path name of the script to be executed when the shell is invoked (see *Invoking ksh* below). This file is typically used for *alias* and *function* definitions.
FCEDIT	The default editor name for the `fc` command.
FPATH	The search path for function definitions. This path is searched when a function with the `-u` attribute is referenced and when a command is not found. If an executable file is found, then it is read and executed in the current environment.
IFS	Internal field separators, normally *space*, *tab*, and *new-line* that are used to separate command words resulting from command or parameter substitution, and for separating words with the special command `read`. The first character of the `IFS` parameter is used to separate arguments for the `"$*"` substitution (see *Quoting* below).
HISTFILE	If this parameter is set when the shell is invoked, its value is the path name of the file that is used to store the command history. The default value is `$HOME/.sh_history`. If the user has appropriate privileges and no `HISTFILE` is given, then no history file is used (see *Command Re-entry* below).
HISTSIZE	If this parameter is set when the shell is invoked, the number of previously entered commands accessible to this shell will be greater than or equal to this number. The

	default is `128`.
`HOME`	The default argument (home directory) for the `cd` command.
`MAIL`	If this parameter is set to the name of a mail file and the `MAILPATH` parameter is not set, the shell informs the user of arrival of mail in the specified file.
`MAILCHECK`	This variable specifies how often (in seconds) the shell checks for changes in the modification time of any of the files specified by the `MAILPATH` or `MAIL` parameters. The default value is `600` seconds. When the time has elapsed the shell checks before issuing the next prompt.
`MAILPATH`	A list of file names separated by colons (:). If this parameter is set, the shell informs the user of any modifications to the specified files that have occurred within the last `MAILCHECK` seconds. Each file name can be followed by a `?` and a message to be printed, in which case the message undergoes parameter and command substitution with the parameter `$_` defined as the name of the changed file. The default message is `you have mail in $_`.
`PATH`	The search path for commands (see *Execution* below). The user cannot change `PATH` if executing `rksh` (except in the `.profile` file).
`PS1`	The value of this parameter is expanded for parameter substitution, to define the primary prompt string which, by default, is `$` followed by a space character. The character `!` in the primary prompt string is replaced by the command number (see *Command Re-entry* below). To include a `!` in the prompt, use `!!`.
`PS2`	Secondary prompt string, by default `>` followed by a space character.
`PS3`	Selection prompt string used within a `select` loop, by default `#?` followed by a space character.
`PS4`	The value of this variable is expanded for parameter substitution and precedes each line of an execution trace. If `PS4` is unset, the execution trace prompt is `+` followed by a space character.
`SHELL`	The path name of the shell is kept in the environment. When invoked, the shell is restricted if the value of this variable contains an `r` in the basename.
`TMOUT`	If set to a value greater than zero, the shell terminates if a command is not entered within the prescribed number of seconds after issuing the `PS1` prompt.
`VISUAL`	Invokes the corresponding option when the value of this variable ends in *emacs*, *gmacs*, or *vi* (see `set` in *Special Commands* below).

The shell gives default values to `PATH`, `PS1`, `PS2`, `MAILCHECK`, `TMOUT`, and `IFS`. `HOME`, `SHELL`, `ENV`, and `MAIL` are never set automatically by the shell (although `HOME`, `SHELL`, and `MAIL` are set by *login*(1)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for field separator characters (found in `IFS`), and split into distinct arguments where such characters are found. `ksh` retains explicit null arguments (or `' '`) but removes implicit null arguments (those resulting from *parameters* that have no values).

File Name Generation

Following substitution, each command *word* is processed as a pattern for file name expansion unless the `-f` option has been `set`. The form of the patterns is the Pattern Matching Notation defined by *regexp*(5). The word is replaced with sorted file names matching the pattern. If no file name is found that matches the pattern, the word is left unchanged.

In addition to the notation described in *regexp*(5), `ksh` recognizes composite patterns made up of one or more pattern lists separated from each other with a `|`. Composite patterns can be formed with one or more of the following:

?(*pattern-list*)	Optionally matches any one of the given patterns.
*(*pattern-list*)	Matches zero or more occurrences of the given patterns.
+(*pattern-list*)	Matches one or more occurrences of the given patterns.
@(*pattern-list*)	Matches exactly one of the given patterns.
!(*pattern-list*)	Matches anything, except one of the given patterns.

Quoting

Each of the *metacharacters* listed above (See *Definitions* above) has a special meaning to the shell and causes termination of a word unless quoted. A character can be *quoted* (i.e., made to stand for itself) by preceding it with a `\`. The pair `\`*new-line* is ignored. All characters enclosed between a pair of single quote marks (`' '`), are quoted. A single quote cannot appear within single quotes. Inside double quote marks (`" "`), parameter and command substitution occurs and `\` quotes the characters `\`, `` ` ``, `"`, and `$`. `$*` and `@` have identical meanings when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, `"$*"` is equivalent to `"$1`*d* `$2`*d*...`"`, where *d* is the first character of the `IFS` parameter, whereas `$@` is equivalent to `"$1"` `"$2"` .... Inside back single quote (accent grave) marks (`` ` ` ``) `\` quotes the characters `\`, `` ` ``, and `$`. If the back single quotes occur within double quotes, `\` also quotes the character `"`.

The special meaning of keywords or aliases can be removed by quoting any character of the keyword. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

The ability to perform integer arithmetic is provided with the special command `let`. Evaluations are performed using long arithmetic. Constants take the form [*base*#]*n*, where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted, base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression of the C language. All the integral operators, other than `++`, `--`, `?:`, and `,` are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a *variable* can be specified with the `-i` option of the `typeset` special command. Arithmetic evaluation is performed on the value of each assignment to a variable with the `-i` attribute. If you do not specify an arithmetic base, the first assignment to the variable determines the arithmetic base. This base is used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the `let` command is provided. For any command beginning with `((`, all characters until the matching `))` are treated as a quoted expression. More precisely, `((`...`))` is equivalent to `let "`...`"`.

Prompting

When used interactively, the shell prompts with the value of `PS1` before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (the value of `PS2`) is issued.

Conditional Expressions.

A *conditional expression* is used with the `[[` compound command to test attributes of files and to compare strings. Word splitting and file name generation are not performed on the words between `[[` and `]]`. Each expression can be constructed from one or more of the following unary or binary expressions:

`-a` *file*	True if *file* exists.
`-b` *file*	True if *file* exists and is a block special file.
`-c` *file*	True if *file* exists and is a character special file.
`-d` *file*	True if *file* exists and is a directory.
`-f` *file*	True if *file* exists and is an ordinary file.
`-g` *file*	True if *file* exists and is has its setgid bit set.
`-h` *file*	True if *file* exists and is a a symbolic link.
`-k` *file*	True if *file* exists and is has its sticky bit set.
`-n` *string*	True if length of *string* is non-zero.
`-o` *option*	True if option named *option* is on.
`-p` *file*	True if *file* exists and is a fifo special file or a pipe.
`-r` *file*	True if *file* exists and is readable by current process.
`-s` *file*	True if *file* exists and has size greater than zero.
`-t` *fildes*	True if file descriptor number *fildes* is open and associated with a terminal device.
`-u` *file*	True if *file* exists and is has its setuid bit set.
`-w` *file*	True if *file* exists and is writable by current process.

`-x` *file*	True if *file* exists and is executable by current process. If *file* exists and is a directory, the current process has permission to search in the directory.
`-z` *string*	True if length of *string* is zero.
`-H` *file*	True if *file* exists and is a hidden directory (see *cdf*(4)).
`-L` *file*	True if *file* exists and is a symbolic link.
`-O` *file*	True if *file* exists and is owned by the effective user ID of this process.
`-G` *file*	True if *file* exists and its group matches the effective group ID of this process.
`-S` *file*	True if *file* exists and is a socket.
file1 `-nt` *file2*	True if *file1* exists and is newer than *file2*.
file1 `-ot` *file2*	True if *file1* exists and is older than *file2*.
file1 `-ef` *file2*	True if *file1* and *file2* exist and refer to the same file.
string = *pattern*	True if *string* matches *pattern*.
string != *pattern*	True if *string* does not match *pattern*.
string1 < *string2*	True if *string1* comes before *string2* based on ASCII value of their characters.
string1 > *string2*	True if *string1* comes after *string2* based on ASCII value of their characters.
exp1 `-eq` *exp2*	True if *exp1* is equal to *exp2*.
exp1 `-ne` *exp2*	True if *exp1* is not equal to *exp2*.
exp1 `-lt` *exp2*	True if *exp1* is less than *exp2*.
exp1 `-gt` *exp2*	True if *exp1* is greater than *exp2*.
exp1 `-le` *exp2*	True if *exp1* is less than or equal to *exp2*.
exp1 `-ge` *exp2*	True if *exp1* is greater than or equal to *exp2*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

(*expression*)	True, if *expression* is true. Used to group expressions.
! *expression*	True if *expression* is false.
expression1 && *expression2*	True, if *expression1* and *expression2* are both true.
expression1 || *expression2*	True, if either *expression1* or *expression2* is true.

Input/Output

Before a command is executed, its input and output can be redirected using a special notation interpreted by the shell. The following can appear anywhere in a simple-command or can precede or follow a command and are not passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used, except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<*word*	Use file *word* as standard input (file descriptor `0`).
>*word*	Use file *word* as standard output (file descriptor `1`). If the file does not exist, it is created. If the file exists, and the `noclobber` option is on, an error occurs; otherwise, the file is truncated to zero length.
>|*word*	Sames as >, except that it overrides the `noclobber` option.
>>*word*	Use file *word* as standard output. If the file exists, output is appended to it (by first searching for the end-of-file); otherwise, the file is created.
<>*word*	Open file *word* for reading and writing as standard input. If the file does not exist it is created.
<<[-]*word*	The shell input is read up to a line that matches *word*, or to an end-of-file. No parameter substitution, command substitution, or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document. Otherwise, parameter and command substitution occurs, \new-line is ignored, and \\ must be used to quote the characters \\, $, \`, and the first character of *word*. If - is appended to <<, all leading tabs are stripped from *word* and from the document.
<&*digit*	The standard input is duplicated from file descriptor *digit* (see *dup*(2)).
>&*digit*	The standard output is duplicated to file descriptor *digit* (see *dup*(2)).

`<&-`	The standard input is closed.
`>&-`	The standard output is closed.
`<&p`	The input from the co-process is moved to standard input.
`>&p`	The output to the co-process is moved to standard output.

If one of the above is preceded by a digit, the file descriptor number cited is that specified by the digit (instead of the default `0` or `1`). For example:

... `2>&1`

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

Redirection order is significant because the shell evaluates redirections referencing file descriptors in terms of the currently open file associated with the specified file descriptor at the time of evaluation. For example:

... `1>`*fname* `2>&1`

first assigns file descriptor 1 (standard output) to file *fname*, then assigns file descriptor 2 (standard error) to the file assigned to file descriptor 1; i.e., *fname*. On the other hand, if the order of redirection is reversed as follows:

... `2>&1 1>`*fname*

file descriptor 2 is assigned to the current standard output (user terminal unless a different assignment is inherited). File descriptor 1 is then reassigned to file *fname* without changing the assignment of file descriptor 2.

The input and output of a *co-process* can be moved to a numbered file descriptor allowing other commands to write to them and read from them using the above redirection operators. If the input of the current *co-process* is moved to a numbered file descriptor, another *co-process* can be started.

If a command is followed by `&` and job control is inactive, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The **environment** (see *environ*(5)) is a list of name-value pairs passed to an executed program much like a normal argument list. The names must be **identifiers** and the values are character strings. The shell interacts with the environment in several ways. When invoked, the shell scans the environment and creates a parameter for each name found, gives it the corresponding value, and marks it *export*. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones by using the `export` or `typeset -x` commands, the values become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell whose values can be modified by the current shell, plus any additions which must be noted in `export` or `typeset -x` commands.

The environment for any *simple-command* or function can be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument takes the form *identifier*=*value*. For example,

`TERM=450` *cmd args*

and

`(export TERM; TERM=450;` *cmd args*`)`

are equivalent (as far as the above execution of *cmd* is concerned except for special commands listed below that are preceded by a dagger).

If the `-k` option is set, all parameter assignment arguments are placed in the environment, even if they occur after the command name. The following echo statement prints `a=b c`. After the `-k` option is set, the second echo statement prints only `c`:

```
echo a=b c
set -k
echo a=b c
```

This feature is intended for use with scripts written for early versions of the shell, and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions

The `function` keyword (described in the *Commands* section above) is used to define shell functions. Shell functions are read and stored internally. Alias names are resolved when the function is read. Functions are executed like commands, with the arguments passed as positional parameters (see *Execution* below).

Functions execute in the same process as the caller except that command substitution of a function creates a new process. Functions share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. If a function does not catch or specifically ignore a trap condition, the function terminates and the condition is passed on to the caller. A trap on **EXIT** set inside a function is executed after the function completes in the environment of the caller. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the **+f** option of the **typeset** special command. Function identifiers and the associated text of the functions can be listed with the **-f** option. Functions can be undefined with the **-f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **-xf** option of the **typeset** command allows a function to be exported to scripts that are executed without reinvoking the shell. Functions that must be defined across separate invocations of the shell should be placed in the **ENV** file.

Jobs

If the **monitor** option of the **set** command is turned on, an interactive shell associates a **job** with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with &, the shell prints a line resembling:

```
[1] 1234
```

indicating that job number 1 was started asynchronously and had one (top-level) process whose process ID was 1234.

If you are running a job and want to do something else, type the suspend character (usually **^Z** (Ctrl-Z)) to send a STOP signal to the current job. The shell then indicates that the job has been 'Stopped', and prints another prompt. The state of this job can be manipulated by using the **bg** command to put it in the background, running other commands (while it is stopped or running in the background), and eventually restarting or returning the job to the foreground by using the **fg** command. A **^Z** takes effect immediately and resembles an interrupt, since pending output and unread input are discarded when **^Z** is typed.

A job run in the background stops if it tries to read from the terminal. Background jobs normally are allowed to produce output, but can be disabled by giving the **stty tostop** command. If the user sets this tty option, background jobs stop when trying to produce output.

There are several ways to refer to jobs in the shell. A job can be referred to by the process ID of any process in the job or by one of the following:

%*number*	The job with the given number.
%*string*	Any job whose command line begins with *string*.
%?*string*	Any job whose command line contains *string*.
%%	Current job.
%+	Equivalent to %%.
%-	Previous job.

The shell learns immediately when a process changes state. It informs the user when a job is blocked and prevented from further progress, but only just before it prints a prompt.

When the monitor mode is on, each background job that completes triggers any trap set for CHLD.

Attempting to leave the shell while jobs are running or stopped produces the warning, **You have stopped (running) jobs**. Use the **jobs** command to identify them. An immediate attempt to exit again terminates the stopped jobs; the shell does not produce a warning the second time.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by `&` and the `monitor` option is off. Otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the `trap` command below).

Execution

Substitutions are made each time a command is executed. If the command name matches one of the *Special Commands* listed below, it is executed within the current shell process. Next, `ksh` checks the command name to determine whether it matches one of the user-defined functions. If it does, `ksh` saves the positional parameters and then sets them to the arguments of the *function* call. When the *function* completes or issues a `return`, `ksh` restores the positional parameter list and executes any trap set on **EXIT** within the function. The value of a *function* is the value of the last command executed. A function is executed in the current shell process. If a command name is not a *special command* or a user-defined *function*, `ksh` creates a process and attempts to execute the command using `exec` (see *exec*(2)).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (`:`). The default path is `/bin:/usr/bin:` (specifying `/bin`, `/usr/bin`, and the current directory in that order). Note that the current directory is specified by a null path name which can appear immediately after the equals sign, between colon delimiters, or at the end of the path list. The search path is not used if the command name contains a `/`. Otherwise, each directory in the path is searched for an executable file. If the file has execute permissions but is not a directory or an executable object code file, it is assumed to be a script file, which is a file of data for an interpreter. If the first two characters of the script file are `#!`, `exec` (see *exec*(2)) expects an interpreter path name to follow. `exec` then attempts to execute the specified interpreter as a separate process to read the entire script file. If a call to `exec` fails, `/bin/ksh` is spawned to interpret the script file. All non-exported aliases, functions, and named parameters are removed in this case. If the shell command file does not have read permission, or if the *setuid* and/or *setgid* bits are set on the file, the shell executes an agent to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is also executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or writable. A shell can access the commands of all *interactive* shells that use the same named **HISTFILE**. The special command `fc` is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If no editor program is specified as an argument to `fc`, the value of the **FCEDIT** parameter is used. If **FCEDIT** is not defined, `/bin/ed` is used. The edited command is printed and re-executed upon leaving the editor. The editor name `-` is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if `r` is aliased to `fc -e -`, typing `r bad=good c` re-executes the most recent command that starts with the letter `c` and replaces the first occurrence of the string `bad` with the string `good`.

Special Commands

The following simple-commands are executed in the shell process. They permit input/output redirection. Unless otherwise indicated, file descriptor 1 is the default output location and the exit status, when there are no syntax errors, is zero. Commands that are preceded by † or †† are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words following a command preceded by †† that are in the format of a variable assignment are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the = sign and word splitting and file name generation are not performed.

† `:` [*arg* ...] The command only expands parameters. A zero exit code is returned.

† `.` *file* [*arg* ...]
Read and execute commands from *file* and return. The commands are executed in the

current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed. It is not necessary that the execute permission bit be set for *file*.

†† **alias** [**-tx**] [*name*[*=value*] ...]
: **alias** with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **-t** option is used to set and list tracked aliases. The value of a tracked alias is the full path name corresponding to the given *name*. The value of a tracked alias becomes undefined when the value of **PATH** is reset, but the alias remains tracked. Without the **-t** option, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **-x** option is used to set or print exported aliases. An exported alias is defined across sub-shell environments. Alias returns true unless a *name* is given for which no alias has been defined.

bg [*job* ...] Puts the specified *jobs* into the background. The current job is put in the background if *job* is unspecified. See *Jobs* for a description of the format of *job*.

† **break** [*n*] Exit from the enclosing **for**, **while**, **until**, or **select** loop, if any. If *n* is specified, break *n* levels.

† **continue** [*n*]
: Resume the next iteration of the enclosing **for**, **while**, **until**, or **select** loop. If *n* is specified, resume at the *n*-th enclosing loop.

cd [**-L**|**-P**] [*arg*]
cd *old new* This command can take either of two forms. In the first form it changes the current directory to *arg*. If *arg* is **-** the directory is changed to the previous directory. The **-L** option (default) preserves logical naming when treating symbolic links. **cd -L ..** moves the current directory one path component closer to the root directory. The **-P** option preserves the physical path when treating symbolic links. **cd -P ..** changes the working directory to the parent directory of the current directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). If **CDPATH** is null or undefined, the default value is the current directory. Note that the current directory is specified by a null path name which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a **/**, the search path is not used. Otherwise, each directory in the path is searched for *arg*.

The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.

The **cd** command cannot be executed by **rksh**.

echo [*arg* ...]
: See *echo*(1) for usage and description.

† **eval** [*arg* ...]
: Reads the arguments as input to the shell and executes the resulting command(s).

† **exec** [*arg* ...]
: Parameter assignments remain in effect after the command completes. If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments can appear and affect the current process. If no arguments are given, the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 opened with this mechanism are closed when invoking another program.

† **exit** [*n*] Causes the shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end-of-file also causes the shell to exit, except when a shell has the *ignoreeof* option set (see **set** below).

†† **export** [*name* [=*value*] ...]
The given *name*s are marked for automatic export to the *environment* of subsequently executed commands.

fc [**-e***ename*][**-nlr**] [*first* [*last*]]
fc -e - [*old*=*new*][*command*]
In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands typed at the terminal. The arguments *first* and *last* can be specified as a number or string. A given string is used to locate the most recent command. A negative number is used to offset the current command number. The **-l** option causes the commands to be listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, the value of the parameter **FCEDIT** (default **/bin/ed**) is used as the editor. Once editing has ended, the commands (if any) are executed. If *last* is omitted, only the command specified by *first* is used. If *first* is not specified, the default is the previous command for editing and −16 for listing. The **-r** option reverses the order of the commands and the **-n** option suppresses command numbers when listing. In the latter, the *command* is re-executed after the substitution *old*=*new* is performed.

fg [*job* ...] Brings each *job* into the foreground in the order specified. If no *job* is specified, the current job is brought into the foreground. See *Jobs* for a description of the format of *job*.

getopts *optstring name* [*arg* ...]
Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a **+** or a **-**. An option not beginning with **+** or **-**, or the argument **--** ends the options. *optstring* contains the letters that *getopts* recognizes. If a letter is followed by a **:**, that option is expected to have an argument. The options can be separated from the argument by blanks.

getopts places the next option letter it finds inside variable *name* each time it is invoked with a **+** preceding it when *arg* begins with a **+**. The index of the next *arg* is stored in **OPTIND**. The option argument, if any, gets stored in **OPTARG**.

A leading **:** in *optstring* causes **getopts** to store the letter of an invalid option in **OPTARG**, and to set *name* to **?** for an unknown option and to **:** when a required option is missing. Otherwise, **getopts** prints an error message. The exit status is non-zero when there are no more options.

jobs [**-lnp**][*job* ...]
Lists information about each given job; or all active jobs if *job* is omitted. The **-l** option lists process ids in addition to the normal information. The **-n** option only displays jobs that have stopped or exited since last notified. The **-p** option causes only the process group to be listed. See *Jobs* for a description of the format of *job*.

kill [**-***sig*] *process* ...
Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are given either by number or name (as given in *signal*(5), stripped of the prefix **SIG**). The signal names are listed by **kill -l**. No default exists; merely typing **kill** does not affect the current job. If the signal being sent is **TERM** (terminate) or **HUP** (hangup), the job or process is sent a **CONT** (continue) signal when stopped. The *process* argument can be either a process ID or job. If the first argument to **kill** is a negative integer, it is interpreted as a *sig* argument and not as a process group.

let *arg* ... Each *arg* is a separate *arithmetic expression* to be evaluated. See *Arithmetic Evaluation* above, for a description of arithmetic expression evaluation. The exit status is 0 if the value of the last expression is non-zero, and 1 otherwise.

† **newgrp** [*arg* ...]
Equivalent to **exec newgrp** *arg* ....

print[**-Rnprsu**[*n*]] [*arg* ...]
The shell output mechanism. With no options or with option **-** or **--** the arguments are printed on standard output as described by *echo*(1). Raw mode, **-R** or **-r**, ignores the escape conventions of *echo*. The **-R** option prints all subsequent arguments and options

other than **-n**. The **-p** option causes the arguments to be written onto the pipe of the process spawned with **|&** instead of standard output. The **-s** option causes the arguments to be written onto the history file instead of standard output. The **-u** option can be used to specify a one-digit file descriptor unit number *n* on which the output is to be placed. The default is 1. If the option **-n** is used, no new-line character is added to the output.

pwd [**-L**|**-P**] With no arguments prints the current working directory (equivalent to **print -r - $PWD**). The **-L** option (default) preserves the logical meaning of the current directory and **-P** preserves the physical meaning of the current directory if it is a symbolic link (see **cd** and *ln*(1)).

read [**-prsu**[*n*]] [*name*]**?***prompt*] [*name* ...]
The shell input mechanism. One line is read and broken up into words using the characters in **IFS** as separators. In **-r** raw mode, \ at the end of a line does not signify line continuation. The first word is assigned to the first *name*, the second word to the second *name*, etc., with remaining words assigned to the last *name*. The **-p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **-s** option is present, the input is saved as a command in the history file. The option **-u** can be used to specify a one-digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is **0**. If *name* is omitted, **REPLY** is used as the default *name*. The return code is **0**, unless an end-of-file is encountered. An end-of-file with the **-p** option causes cleanup for this process so that another process can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* when the shell is interactive. If the given file descriptor is open for writing and is a terminal device, the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2. The return code is **0**, unless an end-of-file is encountered.

†† **readonly** [*name*[=*value*] ...]
The given *names* are marked read-only and these names cannot be changed by subsequent assignment.

† **return** [*n*]
Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted, the return status is that of the last command executed. Only the low 8 bits of *n* are passed back to the caller. If **return** is invoked while not in a *function* or executing a script by the **.** (dot) built-in command, it has the same effect as an **exit** command.

set [**±aefhkmnopstuvx** | **±o** *option*] ... [**±A** *name*] [*arg* ...]
The following options are used for this command:

- **-A** Array assignment. Unset the variable *name* and assign values sequentially from the list *arg*. If **+A** is used, the variable *name* is not unset first.
- **-a** All subsequent defined parameters are automatically exported.
- **-e** If the shell is non-interactive and if a command fails, execute the **ERR** trap, if set, and exit immediately. This mode is disabled while reading profiles.
- **-f** Disables file name generation.
- **-h** Each command whose name is an *identifier* becomes a tracked alias when first encountered.
- **-k** All parameter assignment arguments (not just those that precede the command name) are placed in the environment for a command.
- **-m** Background jobs are run in a separate process group and a line is printed upon completion. The exit status of background jobs is reported in a completion message. This option is turned on automatically for interactive shells.
- **-n** Read commands and check them for syntax errors, but do not execute them. The **-n** option is ignored for interactive shells.
- **-o** The **-o** argument takes any of several *option* names, but only one *option* can be specified with each **-o** option. If none is supplied, the current option settings are printed. The **-o** argument *option* names follow:

`allexport`	Same as `-a`.
`bgnice`	All background jobs are run at a lower priority.
`errexit`	Same as `-e`.
`emacs`	Activates an `emacs`-style in-line editor for command entry.
`gmacs`	Activates a `gmacs`-style in-line editor for command entry.
`ignoreeof`	The shell does not exit on end-of-file. The command `exit` must be used.
`keyword`	Same as `-k`.
`markdirs`	All directory names resulting from file name generation have a trailing `/` appended.
`monitor`	Same as `-m`.
`noclobber`	Prevents redirection `>` from truncating existing files. Requires `>|` to truncate a file when turned on.
`noexec`	Same as `-n`.
`noglob`	Same as `-f`.
`nolog`	Do not save function definitions in history file.
`nounset`	Same as `-u`.
`privileged`	Same as `-p`.
`verbose`	Same as `-v`.
`trackall`	Same as `-h`.
`vi`	Activates the insert mode of a `vi`-style in-line editor until you press the ESC key which puts you in move mode. A return sends the line.
`viraw`	Each character is processed as it is typed in `vi` mode.
`xtrace`	Same as `-x`.

-p
Disables processing of the **$HOME/.profile** file and uses the file **/etc/suid_profile** instead of the ENV file. This mode is on whenever the effective uid (gid) is not equal to the real uid (gid). Turning this off causes the effective uid and gid to be set to the real uid and gid.

-s
Sort the positional parameters.

-t
Exit after reading and executing one command.

-u
Treat unset parameters as an error when substituting.

-v
Print shell input lines as they are read.

-x
Print commands and their arguments as they are executed.

-
Turns off **-x** and **-v** options and stops examining arguments for options.

--
Do not change any of the options; useful in setting **$1** to a value beginning with **-**. If no arguments follow this option, the positional parameters are unset.

Using **+** instead of **-** before a option causes the option to be turned off. These options can also be used when invoking the shell. The current set of options can be examined by using **$-**.

Unless **-A** is specified, the remaining *arg* arguments are positional parameters and are assigned consecutively to **$1**, **$2**, ... If neither arguments nor options are given, the values of all names are printed on the standard output.

† **shift** [*n*]
The positional parameters from **$***n***+1** ... are renamed **$1** ...; default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

test [*expr*]
Evaluate conditional expression *expr*. See *test*(1) for usage and description. The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. Four additional

primitive expressions are allowed:

`-L` *file*	True if *file* is a symbolic link.
file1 `-nt` *file2*	True if *file1* is newer than *file2*.
file1 `-ot` *file2*	True if *file1* is older than *file2*.
file1 `-ef` *file2*	True if *file1* has the same device and i-node number as *file2*.

† **`times`**
Print the accumulated user and system times for the shell and for processes run from the shell.

† **`trap`** [*arg*] [*sig* ...]
arg is a command read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or name of the signal. Trap commands are executed in signal number order. Any attempt to set a trap on a signal that was ignored upon entering the current shell has no effect. If *arg* is omitted or is `-`, all traps for *sig* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *sig* is **DEBUG**, *arg* is executed after each command. If *sig* is **ERR**, *arg* is executed whenever a command has a non-zero exit code. If *sig* is `0` or **EXIT** and the **`trap`** statement is executed inside the body of a function, the command *arg* is executed after the function completes. If *sig* is `0` or **EXIT** for a **`trap`** set outside any function, the command *arg* is executed on exit from the shell. The **`trap`** command with no arguments prints a list of commands associated with each signal number.

†† **`typeset`** [**±LRZfilrtux**[*n*]] [*name*[=*value*]] ...
Parameter assignments remain in effect after the command completes. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes can be specified:

- `-L` Left justify and remove leading blanks from *value*. If *n* is non-zero, it defines the width of the field. Otherwise, it is determined by the width of the value of first assignment. When the *name* is assigned, the value is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the `-Z` option is also set. The `-R` option is turned off.
- `-R` Right justify and fill with leading blanks. If *n* is non-zero, it defines the width of the field. Otherwise, it is determined by the width of the value of first assignment. The field is left-filled with blanks or truncated from the end if the parameter is reassigned. The `-L` option is turned off.
- `-Z` Right justify and fill with leading zeros if the first non-blank character is a digit and the `-L` option has not been set. If *n* is non-zero, it defines the width of the field. Otherwise, it is determined by the width of the value of first assignment.
- `-f` Cause *name* to refer to function names rather than parameter names. No assignments can be made to the *name* declared with the **`typeset`** statement. The only other valid options are `-t` (which turns on execution tracing for this function) and `-x` (which allows the function to remain in effect across shell procedures executed in the same process environment).
- `-i` Parameter is an integer. This makes arithmetic faster. If *n* is non-zero, it defines the output arithmetic base; otherwise the first assignment determines the output base.
- `-l` Convert all uppercase characters to lowercase. The uppercase `-u` option is turned off.
- `-r` Any given *name* is marked "read only" and cannot be changed by subsequent assignment.
- `-t` Tag the named parameters. Tags are user definable and have no special meaning to the shell.
- `-u` Convert all lowercase characters to uppercase characters. The lowercase `-l` option is turned off.
- `-x` Mark any given *name* for automatic export to the environment of subsequently executed commands.

Using `+` instead of `-` causes these options to be turned off. If no *name* arguments are given but options are specified, a list of names (and optionally the values) of the parameters that have these options set is printed. Using `+` instead of `-` retains the values to be printed. If neither names nor options are given, the names and attributes of all parameters are printed.

`ulimit` [*n*]
If *n* is given, impose a size limit of *n* 512 byte blocks on files written by child processes (files of any size can be read). If *n* is not given, the current limit is printed.

`umask` [*mask*]
The user file-creation mask is set to *mask* (see *umask*(2)). *mask* can either be an octal number or a symbolic value as described in *chmod*(1). If a symbolic value is given, the new umask value is the complement of the result of applying *mask* to the complement of the previous umask value. If *mask* is omitted, the current value of the mask is printed.

`unalias` *name* ...
The parameters given by the list of *names* are removed from the *alias* list.

`unset` [`-f`] *name* ...
The parameters given by the list of *names* are unassigned; that is, their values and attributes are erased. Read-only variables cannot be unset. If the `-f` option is set, *names* refer to function names. Unsetting **ERRNO**, **LINENO**, **MAILCHECK**, **OPTARG**, **OPTIND**, **RANDOM**, **SECONDS**, **TMOUT**, and _ removes their special meaning even if they are subsequently assigned to.

† `wait` [*job*]
Wait for the specified *job* to terminate or stop, and report its status. This status becomes the return code for the `wait` command. If *job* is not given, `wait` waits for all currently active child processes to terminate or stop. The termination status returned is that of the last process. See *Jobs* for a description of the format of a *job*.

`whence` [`-pv`] *name* ...
For each *name*, indicate how it would be interpreted if used as a command name. The `-v` option produces a more verbose report. The `-p` option does a path search for *name* even if *name* is an alias, a function, or a reserved word.

Invoking `ksh`

If the shell is invoked by `exec` (see *exec*(2)), and the first character of argument zero (`$0`) is `-`, the shell is assumed to be a login shell and commands are read first from `/etc/profile`. The expression `${HOME:-.}/.profile` is then evaluated and an attempt to open the resulting filename is made. If the file is opened successfully, the file is read. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV**, if the file exists. If the `-s` option is not present and *arg* is, a path search is performed on the first *arg* to determine the name of the script to execute. When running `ksh` with *arg*, the script *arg* must have read permission and any *setuid* and *getgid* settings are ignored. Commands are then read as described below. The following options are interpreted by the shell when it is invoked:

`-c` *string*	If the `-c` option is present, commands are read from *string*.
`-s`	If the `-s` option is present or if no arguments remain, commands are read from the standard input. Shell output, except for the output of some of the *Special commands* listed above, is written to file descriptor 2.
`-i`	If the `-i` option is present or if the shell input and output are attached to a terminal (as reported by *tty*(3C)), the shell is interactive. In this case SIGTERM is ignored (so that `kill 0` does not kill an interactive shell) and SIGINT +1 is caught and ignored (so that `wait` is interruptible). In all cases, SIGQUIT is ignored by the shell. (See *signal*(5).)
`-r`	If the `-r` option is present, the shell is a restricted shell.

The remaining options and arguments are described under the `set` command above.

`rksh` Only

`rksh` is used to set up login names and execution environments where capabilities are more controlled than those of the standard shell. The actions of `rksh` are identical to those of `ksh`, except that the following are forbidden:

- Changing directory (see *cd*(1))
- Setting the value of **SHELL**, **ENV**, or **PATH**
- Specifying path or command names containing `/`
- Redirecting output (>, >|, <>, and >>)

The restrictions above are enforced after the `.profile` and **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, `rksh` invokes `ksh` to execute it. Thus, the end-user is provided with shell procedures accessible to the full power of the standard shell, while being restricted to a limited menu of commands. This scheme assumes that the end-user does not have write and execute permissions in the same directory.

These rules effectively give the writer of the **.profile** file complete control over user actions, by performing guaranteed set-up actions and leaving the user in an appropriate directory (probably not the login directory).

The system administrator often sets up a directory of commands (usually **/usr/rbin**) that can be safely invoked by **rksh**. HP-UX systems provide a restricted editor *red* (see *ed*(1)), suitable for restricted users.

COMMAND-LINE EDITING

In-line Editing Options

Normally, each command line typed at a terminal device is followed by a new-line (carriage-return or line-feed). If either the **emacs**, **gmacs**, or **vi** option is set, the user can edit the command line. An editing option is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept Return as carriage return without line feed and that a space character must overwrite the current character on the screen. ADM terminal users should set the "space/advance" switch to "space". Hewlett-Packard terminal users should set the straps to "bcGHxZ etX".

The editing modes enable the user to look through a window at the current line. The default window width is 80, unless the value of **COLUMNS** is defined. If the line is longer than the window width minus two, a mark displayed at the end of the window notifies the user. The mark is a **>**, **<**, or ***** if the line extends respectively on the right, left, or both side(s) of the window. As the cursor moves and reaches the window boundaries, the window is centered about the cursor.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading **^** in the string restricts the match to begin at the first character in the line.

Emacs Editing Mode

This mode is invoked by either the **emacs** or **gmacs** option. Their sole difference is their handling of **^T**. To edit, the user moves the cursor to the point needing correction and inserts or deletes characters or words. All editing commands are control characters or escape sequences. The notation for control characters is circumflex (**^**) followed by the character. For example, **^F** is the notation for Ctrl-**F**. This is entered by pressing the **f** key while holding down the Ctrl (control) key. The Shift key is *not* pressed. (The notation **^?** indicates the DEL (delete) key.)

The notation for escape sequences is *M-* followed by a character. For example, *M-***f** (pronounced Meta f) is entered by depressing ESC (ASCII 033) followed by **f**. *M-***F** would be the notation for ESC followed by Shift (capital) **F**.

All edit commands operate from any place on the line (not only at the beginning). Neither the Return nor the Line Feed key is entered after edit commands, except when noted.

^F	Move cursor forward (right) one character.
*M-***f**	Move cursor forward one word. (The editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
*M-***b**	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]*char*	Move cursor forward to character *char* on current line.
*M-***^]***char*	Move cursor backward to character *char* on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the *stty*(1) command, usually **^H** or **#**.) Delete previous character.
^D	Delete current character.
eof	End-of-file character, normally **^D**, terminates the shell if the current line is null.
*M-***d**	Delete current word.
*M-***^H**	(Meta-backspace) Delete previous word.
*M-***h**	Delete previous word.
*M-***^?**	(Meta-DEL) Delete previous word (if interrupt character is **^?** (DEL, the default) this command does not work).

`^T`	Transpose current character with next character in **emacs** mode. Transpose two previous characters in **gmacs** mode.
`^C`	Capitalize current character.
M-`c`	Capitalize current word.
M-`l`	Change the current word to lowercase.
`^K`	Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less that the current cursor position, delete from the given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, from the cursor up to the given position.
`^W`	Kill from the cursor to the mark.
M-`p`	Push the region from the cursor to the mark on the stack.
kill	(User-defined kill character, as defined by the *stty*(1) command, usually `^G` or `@`.) Kill the entire current line. If two *kill* characters are entered in succession, all subsequent consecutive kill characters cause a line feed (useful when using paper terminals).
`^Y`	Restore last item removed from line (yank item back to the line).
`^L`	Line feed and print current line.
`^@`	(Null character) Set mark.
(Meta space) Set mark.	
`^J`	(New line) Execute the current line.
`^M`	(Return) Execute the current line.
`^P`	Fetch previous command. Each time `^P` is entered, the next previous command in the history list is accessed.
`^N`	Fetch next command. Each time `^N` is entered the next command in the history list is accessed.
M-`<`	Fetch the least recent (oldest) history line.
M-`>`	Fetch the most recent (youngest) history line.
`^R`*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *string* is terminated by a Return or New-Line. If *string* is preceded by a `^`, the matched line must begin with *string*. If *string* is omitted, the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.
`^O`	Operate - Execute the current line and fetch from the history file the next line relative to current line.
M-digits	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are `^F`, `^B`, *erase*, `^C`, `^D`, `^K`, `^R`, `^P`, `^N`, `^]`, *M*-`.`, *M*-`_`, *M*-`b`, *M*-`c`, *M*-`d`, *M*-`f`, *M*-`h`, *M*-`l` and *M*-`^H`.
M-letter	Softkey. User's alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value is inserted on the input queue. This *letter* must not be one of the above meta-functions.
M-`.`	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-`_`	Same as *M*-`.`.
M-`*`	Attempt file-name generation on the current word.
M-ESC	File-name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a `/` is appended if the file is a directory and a space is appended if the file is not a directory.
M-`=`	List files matching current word pattern as if an asterisk were appended.
`^U`	Multiply parameter of next command by 4.
`\`	Escape next character. Editing characters, the user's erase, kill and interrupt (normally `^?`) characters can be entered in a command line or in a search string if preceded by a `\`. The `\` removes the next character's editing features (if any).
`^V`	Display version of the shell.
M-`#`	Insert a `#` at the beginning of the line and execute it. This causes a comment to be inserted in the history file.

Vi Editing Mode

There are two typing modes. Entering a command puts you into *input* mode. To edit, the user enters *control* mode by pressing ESC and moves the cursor to the point needing correction, then inserts or deletes characters or words. Most control commands accept an optional repeat *count* prior to the command.

In vi mode on most systems, canonical processing is initially enabled and the command is echoed again if the speed is 1200 baud or greater and contains any control characters, or if less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

Setting the `viraw` option always disables canonical processing on the terminal. This mode is implicit for systems that do not support two alternate end-of-line delimiters, and can be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase	Delete previous character. (*erase* is a user-defined erase character, as defined by the *stty*(1) command, usually `^H` or `#`.)
`^W`	Delete the previous blank separated word.
`^D`	Terminate the shell.
`^V`	Escape next character. Editing characters, erase or kill characters can be entered in a command line or in a search string if preceded by a `^V`. `^V` removes the next character's editing features (if any).
`\`	Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands move the cursor. The designation [*count*] causes a repetition of the command the cited number of times.

[*count*]`l`	Cursor forward (right) one character.
[*count*]`w`	Cursor forward one alphanumeric word.
[*count*]`W`	Cursor to the beginning of the next word that follows a blank.
[*count*]`e`	Cursor to end of word.
[*count*]`E`	Cursor to end of the current blank-delimited word.
[*count*]`h`	Cursor backward (left) one character.
[*count*]`b`	Cursor backward one word.
[*count*]`B`	Cursor to preceding blank separated word.
[*count*]`|`	Cursor to column *count*. Default is 1.
[*count*]`f`*c*	Find the next character *c* in the current line.
[*count*]`F`*c*	Find the previous character *c* in the current line.
[*count*]`t`*c*	Equivalent to `f` followed by `h`.
[*count*]`T`*c*	Equivalent to `F` followed by `l`.
[*count*]`;`	Repeats the last single character find command, `f`, `F`, `t`, or `T`.
[*count*]`,`	Reverses the last single character find command.
`0`	Cursor to start of line.
`^`	Cursor to first nonblank character in line.
`$`	Cursor to end of line.

Search Edit Commands

These commands access your command history.

[*count*]`k`	Fetch previous command. Each time `k` is pressed, the next earlier command in the history list is accessed.
[*count*]`-`	Equivalent to `k`.
[*count*]`j`	Fetch next command. Each time `j` is entered, the next later command in the history list is accessed.
[*count*]`+`	Equivalent to `j`.
[*count*]`G`	The command number *count* is fetched. The default is the first command in the history list.
`/`*string*	Search backward through history for a previous command containing *string*. *string* is terminated by a "Return" or "New-line". If *string* is preceded by a `^`, the matched line must begin with *string*. If *string* is null, the previous string is used.
`?`*string*	Same as `/` but search in the forward direction.
`n`	Search for next match of the last pattern to `/` or `?` commands.
`N`	Search for next match of the last pattern to `/` or `?`, but in reverse direction. Search history for the *string* entered by the previous `/` command.

Text Modification Edit Commands

These commands modify the line.

`a`	Enter input mode and enter text after the current character.
`A`	Append text to the end of the line. Equivalent to **`$a`**.
[*count*]`c`*motion*	
`c`[*count*]*motion*	Move cursor to the character position specified by *motion*, deleting all characters between the original cursor position and new position, and enter input mode. If *motion* is `c`, the entire line is deleted and input mode entered.
`C`	Delete the current character through the end of line and enter input mode. Equivalent to `c$`.
`S`	Equivalent to `cc`.
`D`	Delete the current character through end of line. Equivalent to **`d$`**.
[*count*]`d`*motion*	
`d`[*count*]*motion*	Move cursor to the character position specified by *motion*, deleting all characters between the original cursor position and new position. If *motion* is `d`, the entire line is deleted.
`i`	Enter input mode and insert text before the current character.
`I`	Insert text before the beginning of the line. Equivalent to the two-character sequence `0i`.
[*count*]`P`	Place the previous text modification before the cursor.
[*count*]`p`	Place the previous text modification after the cursor.
`R`	Enter input mode and replace characters on the screen with characters you type in overlay fashion.
[*count*]`r`*c*	Replace the current character with *c*.
[*count*]`x`	Delete current character.
[*count*]`X`	Delete preceding character.
[*count*]`.`	Repeat the previous text modification command.
[*count*]`~`	Invert the case of the current character and advance the cursor.
[*count*]`_`	Causes the *count* word of the previous command to be appended at the current cursor location and places the editor in input mode at the end of the appended text. The last word is used if *count* is omitted.
`*`	Appends an `*` to the current word and attempts file name generation. If no match is found, the bell rings. If a match is found, the word is replaced by the matching string and the command places the editor in input mode.
ESC	
`\`	Attempt file name completion on the current word. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a `/` is appended if the file is a directory and a space is appended if the file is not a directory.

Other Edit Commands

[*count*]`y`*motion*	
`y`[*count*]*motion*	Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.
`Y`	Yanks from current position to end of line. Equivalent to **`y$`**.
`u`	Undo the last text modifying command.
`U`	Undo all the text modifying commands performed on the line.
[*count*]`v`	Returns the command **`fc -e ${VISUAL:-${EDITOR:-vi}}`** *count* in the input buffer. If *count* is omitted, the current line is used.
`^L`	Line feed and print current line. Has effect only in control mode.
`^J`	(New line) Execute the current line, regardless of mode.
`^M`	(Return) Execute the current line, regardless of mode.
`#`	Equivalent to `I#` followed by **Return**. Sends the line after inserting a `#` in front of the line and after each new-line. Useful for inserting the current command line in the history list without executing it.
`=`	List the filenames that match the current word if an asterisk were appended to it.
`@`*letter*	The user's alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value is inserted on the input queue for processing.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the classification of characters as letters, and the characters matched by character class expressions in pattern matching notation.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, **ksh** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

RETURN VALUE

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (also see the **exit** command above). If the shell is being used non-interactively, execution of the shell file is abandoned. Runtime errors detected by the shell are reported by printing the command or function name and the error condition. If the line number on which the error occurred is greater than one, the line number is also printed in brackets (**[ ]**) after the command or function name.

WARNINGS

File descriptors 10 and 14 through 20 are used internally by the Korn Shell. Applications using these and forking a subshell should not depend upon them surviving in the subshell or its descendants.

If a command which is a *tracked alias* is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell continues to load and execute the original command. Use the **-t** option of the **alias** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Some very old shell scripts contain a caret (**^**) as a synonym for the pipe character (**|**). Note however, **ksh** does not recognize the caret as a pipe character.

If a command is piped into a shell command, all variables set in the shell command are lost when the command completes.

Using the **fc** built-in command within a compound command causes the entire command to disappear from the history file.

The built-in command **.** *file* reads the entire file before any commands are executed. Therefore, **alias** and **unalias** commands in the file do not apply to any functions defined in the file.

Traps are not processed while the shell is waiting for a foreground job. Thus, a trap on **CHLD** is not executed until the foreground job terminates.

The **export** built-in command does not handle arrays properly. Only the first element of an array is exported to the *environment*.

Background processes started from a non-interactive shell cannot be accessed by using job control commands.

In an international environment, character ordering is determined by the setting of LC_COLLATE, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particularly when using range expressions in file name generation patterns. For example, the command,

```
    rm [a-z]*
```

might be expected to match all file names beginning with a lowercase alphabetic character. However, if dictionary ordering is specified by LC_COLLATE, it would also match file names beginning with an uppercase character (as well as those beginning with accented letters). Conversely, it would fail to match letters collated after **z** in languages such as Danish or Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form:

```
rm [[:lower:]]*
```

This uses LC_CTYPE to determine character classes and works predictably for all supported languages and codesets. For shell scripts produced on non-internationalized systems (or without consideration for the above dangers), it is recommended that they be executed in a non-NLS environment. This requires that LANG, LC_COLLATE, etc., be set to "C" or not set at all.

Be aware that the value of the `IFS` variable in the user's environment affects the behavior of scripts.

`ksh` implements command substitution by creating a pipe between itself and the command. If the root file system is full, the substituted command cannot write to the pipe. As a result, the shell receives no input from the command, and the result of the substitution is null. In particular, using command substitution for variable assignment under such circumstances results in the variable being silently assigned a NULL value.

AUTHOR

`ksh` was developed by AT&T.

FILES

`/etc/passwd`	to find home directories
`/etc/profile`	read to set up system environment
`/etc/suid_profile`	security profile
`$HOME/.profile`	read to set up user's custom environment
`/tmp/sh*`	for here-documents

SEE ALSO

cat(1), cd(1), echo(1), env(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), gtty(2), pipe(2), signal(5), umask(2), ulimit(2), wait(2), rand(3C), a.out(4), profile(4), environ(5), lang(5), regexp(5).

NAME

lastcomm - show last commands executed in reverse order

SYNOPSIS

`lastcomm` [*command*name] ... [*user*name] ... [*terminal*name] ...

DESCRIPTION

`lastcomm` gives information on previously executed commands. If no arguments are specified, `lastcomm` prints information about all the commands recorded during the current accounting file's lifetime. If called with arguments, only accounting entries with a matching command name, user name, or terminal name are printed. For example, to produce a listing of all executions of commands named `a.out` by user `root` on terminal `ttyd0` use:

```
lastcomm a.out root ttyd0
```

For each process entry, the following are printed.

- Name of the user who ran the process.
- Flags, as accumulated by the accounting facilities in the system.
- Command name under which the process was called.
- Amount of cpu time used by the process (in seconds).
- What time the process started.

Flags are encoded as follows:

S	Command was executed by a user who has appropriate privileges.
F	Command ran after a fork, but without a following *exec*.
D	Command terminated with the generation of a `core` file.
X	Command was terminated with the signal SIGTERM.

AUTHOR

`lastcomm` was developed by the University of California, Berkeley.

SEE ALSO

last(1), acct(4), core(4).

NAME

ld - link editor

SYNOPSIS

`ld` [`-bdmnqrstvxzENQZ`] [`+b` *path_list*] [`+e` *symbol*] ... [`+s`] [`+I` *symbol*] [`-a` *search*] [`-c` *filename*] [`-e` *epsym*] [`-h` *symbol*] ... [`-o` *outfile*] [`-u` *symbol*] ... [`-y` *symbol*] ... [`-A` *name*] [`-B` *bind*] [`-D` *offset*] [`-L` *dir*] ... [`-R` *offset*] [`-V` *num*] [`-X` *num*] [`-lx` | *file*] ...

DESCRIPTION

`ld` takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so it resolves references to external symbols, assigns final addresses to procedures and variables, revises code and data to reflect new addresses (a process called "relocation"), and updates symbolic debug information (when present in the file). By default, `ld` produces an executable file that can be run by the HP-UX loader `exec()` (see *exec*(2). Alternatively, the linker can generate a relocatable file that is suitable for further processing by `ld` (see `-r` below). It can also generate a shared library (see `-b` below). The linker marks the output file non-executable if any unresolved external references remain. `ld` may or may not generate an output file if any other errors occur during its operation; see DEPENDENCIES. `ld` recognizes three kinds of input files: object files created by the compilers, assembler, or linker (also known as `.o` files), shared libraries created by the linker, and archives of object files (called archive libraries). An archive library contains an index of all the externally-visible symbols from its component object files. (The archiver command *ar*(1) creates and maintains this index.) `ld` uses this table to resolve references to external symbols.

`ld` processes files in the same order as they appear on the command line. It includes code and data from an archive library element if and only if that object module provides a definition for a currently unresolved reference within the user's program. It is common practice to list libraries following the names of all simple object files on the command line.

Code from shared libraries is never copied into an executable program, and data is copied only if referenced directly by the program. The dynamic loader `/lib/dld.sl` is invoked at startup time by `/lib/crt0.o` if a program uses shared libraries. The dynamic loader attaches each required library to the process and resolves all symbolic references between the program and its libraries. The text segment of a shared library is shared among all processes that use the library.

Environment Variables

Arguments can be passed to the linker through the `LDOPTS` environment variable as well as on the command line. The linker picks up the value of `LDOPTS` and places its contents before any arguments on the command line.

The `LD_PXDB` environment variable defines the full execution path for the debug preprocessor **pxdb**. The default value is `/usr/bin/pxdb`. `ld` invokes **pxdb** on its output file if that file is executable and contains debug information. To defer invocation of **pxdb** until the first debug session, set `LD_PXDB` to `/dev/null`.

The `LPATH` environment variable can be used to specify default directories to search for library files. See the `-l` option.

Options

`ld` recognizes the following options:

`-a` *search*	Specify whether shared or archive libraries are searched with the `-l` option. The value of *search* should be one of `archive`, `shared`, or `default`. This option can appear more than once, interspersed among `-l` options, to control the searching for each library. The default is to use the shared version of a library if one is available, or the archive version if not. If either `archive` or `shared` is active, only the specified library type is accepted.
`-b`	Create a shared library rather than a normal executable file. Object files processed with this option should contain **position-independent code** (PIC). See the discussion of PIC in *cc*(1), *CC*(1) [part of optional C++ compiler documentation], *f77*(1), *pc*(1), and *as*(1).
`-c` *filename*	Read options from an indirect file. Each line contains zero or more arguments separated by white space. A # character implies that the rest of the line is a comment. To escape a # character, use the sequence ##.

`-d`	Forces definition of "common" storage; i.e., assign addresses and sizes, for `-r` output.
`-e` *epsym*	Set the default entry point address for the output file to be that of the symbol *epsym*. (This option only applies to executable files.)
`-h` *symbol*	Prior to writing the symbol table to the output file, mark this name as "local" so that it is no longer externally visible. This ensures that this particular entry will not clash with a definition in another file during future processing by `ld`. More than one *symbol* can be specified, but `-h` must precede each one. If used when building a shared library or program, this option prevents the named symbol from being visible to the dynamic loader.
`-l`*x*	Search a library `lib`*x*`.a` or `lib`*x*`.sl`, where *x* is one or more characters. The current state of the `-a` option determines whether the archive (`.a`) or shared (`.sl`) version of a library is searched. Because a library is searched when its name is encountered, the placement of a `-l` is significant. By default, libraries are located in `/lib` and `/usr/lib`. If the environment variable `LPATH` is present in the user's environment, it should contain a colon-separated list of directories to search. These directories are searched instead of the default directories, but `-L` options can still be used. If a program uses shared libraries, the dynamic loader `/lib/dld.sl` will attempt to load each library from the same directory in which it was found at link time (see `+s` and `+b` options).
`-m`	Produce a load map on the standard output.
`-n`	Generate an (executable) output file with file type `SHARE_MAGIC`. This option is incompatible with `-N` and `-q`.
`-o` *outfile*	Produce an output object file named *outfile* (`a.out` if *outfile* is not specified).
`-q`	Generate an (executable) output file with file type `DEMAND_MAGIC`. This option is incompatible with `-n`, `-N`, and `-Q`,
`-r`	Retain relocation information in the output file for subsequent re-linking. The `ld` command does not report undefined symbols. The `-r` option is incompatible with `-A` and `-b`.
`-s`	Strip the output file of all symbol table, relocation, and debug support information. This might impair or prevent the use of a symbolic debugger on the resulting program. This option is incompatible with `-r`. (The *strip*(1) command also removes this information.)
`-t`	Print a trace (to standard output) of each input file as `ld` processes it.
`-u` *symbol*	Enter *symbol* as an undefined symbol in the symbol table. The resulting unresolved reference is useful for linking a program solely from object files in a library. More than one *symbol* can be specified, but each must be preceded by `-u`.
`-v`	Display verbose messages during linking. For each library module that is loaded, the linker indicates which symbol caused that module to be loaded.
`-x`	Partially strip the output file; that is, leave out local symbols. The intention is to reduce the size of the output file without impairing the effectiveness of object file utilities. Note: use of `-x` might affect the use of a debugger.
`-y` *symbol*	Indicate each file in which *symbol* appears. Many such options can be given to trace many symbols, but `-y` must precede each one.
`-z`	Arrange for run-time dereferencing of null pointers to produce a `SIGSEGV` signal. (This is the complement of the `-Z` option.)
`-A` *name*	This option specifies incremental loading. Linking is arranged so that the resulting object can be read into an already executing program. The argument *name* specifies a file whose symbol table provides the basis for defining additional symbols. Only newly linked material is entered into the text and data portions of `a.out`, but the new symbol table reflects all symbols defined before and after the incremental load. Also, the `-R` option can be used in conjunction with `-A`, which allows the newly

linked segment to commence at the corresponding address. The default starting address is the old value of `_end`. The `-A` option is incompatible with `-r` and `-b`. See the *Programming on HP-UX* manual for a description of this option.

`-B` *bind*	Select run-time binding behavior of a program using shared libraries. The most common values for *bind* are `immediate` or `deferred`. The default is `deferred`, which tells the dynamic loader `/lib/dld.sl` to resolve procedure calls on first reference rather than at program start-up time. A modifier bind value of `nonfatal` can be used with the `immediate` major binding mode to allow procedure calls that cannot be resolved at program start-up to be resolved on first reference. See the example below.
`-D` offset	Set the origin (in hexadecimal) for the data segment.
`-E`	Mark all symbols defined by a program for export to shared libraries. By default, `ld` marks only those symbols that are actually referenced by a shared library seen at link time.
`-L` *dir*	Change the algorithm of searching for `lib`*x*`.a` or `lib`*x*`.sl` to look in *dir* before looking in default locations. More than one directory can be specified, but each must be preceded by `-L`. The `-L` option is effective only if it precedes the `-l` option on the command line.
`-N`	Generate an (executable) output file with file type `EXEC_MAGIC`. This option is incompatible with `-n` and `-q`. This option causes the data to be placed immediately following the text, and makes the text writable. Files of this type cannot be shared.
`-Q`	Generate an (executable) output file with file type `EXEC_MAGIC` or `SHARE_MAGIC`, depending on whether `-N` or `-n` is specified. This option is incompatible with `-q`.
`-R` *offset*	Set the origin (in hexadecimal) for the text (i.e., code) segment.
`-V` *num*	Use *num* as a decimal version stamp identifying the `a.out` file that is produced. This is not the same as the version information reported by the SCCS `what` command (see *what*(1)), nor is it the same as the version information recorded for shared library use.
`-X` *num*	Define the initial size for the linker's global symbol table. This reduces link time for very large programs, especially those with a large number of external symbols.
`-Z`	Arrange to allow run-time dereferencing of null pointers. See the discussions of `-Z` and *pointers* in *cc*(1). (This is the complement of the `-z` option.)
`+b` *path_list*	Specify a colon-separated list of directories to be searched at program run-time to locate shared libraries needed by the executable output file that were specified with either the `-l` or `-l:` (Series 700/800) option. An argument consisting of a single colon (`:`) indicates that `ld` should build the list using all the directories specified by the `-L` option and the `LPATH` environment variable (see `+s` option).
`+e` *symbol*	When building a shared library or program, mark the symbol for export to the dynamic loader. With the use of this option, only symbols explicitly marked are exported. When building a shared library, calls to symbols that are not exported are resolved internally.
`+s`	Indicates that the shared library loader can use the environment variable `SHLIB_PATH` to locate shared libraries needed by the executable output file that were specified with either the `-l` or `-l:` (Series 700/800) option. The `SHLIB_PATH` environment variable should be set to a colon-separated list of directories. If both `+s` and `+b` are used, their relative order on the command line indicates which path list will be searched first (see the `+b` option).
`+I` *symbol*	Specify the name of the initializer function when building a shared library. See the *Programming on HP-UX* manual for a description of the initializer function.

Defaults
Unless otherwise directed, `ld` names its output `a.out`. The `-o` option overrides this. Executable output files can be shared. The default state of `-a` is to search shared libraries if available, archive libraries

otherwise. The default bind behavior is `deferred`.

The default value of the `-Z`/`-z` option is `-Z`.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed (Series 700/800 only).

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, `ld` behaves as if all internationalization variables are set to "C". See *environ*(5).

DIAGNOSTICS

`ld` returns zero when the link is successful. A non-zero return code indicates that an error occurred.

EXAMPLES

Link part of a C program for later processing by `ld`. Specify a version number of 2 for the output file (note the `.o` suffix for the output object file; this is an HP-UX convention for indicating a linkable object file):

```
ld -V 2 -r file1.o file2.o -o prog.o
```

Link a simple FORTRAN program for use with the `xdb` symbolic debugger (see *xdb*(1)). Output file name is `a.out` since there is no `-o` option in the command line. Options shown are for a Series 300/400 system:

```
ld /lib/frt0.o ftn.o -lquad -lI077 -lF77 -lm -lc /usr/lib/end.o
```

Create a shared library:

```
ld -b -o libfunc.sl func1.o func2.o func3.o
```

Link a program with `libfunc.sl` but use the archive version of the C library. Specify the immediate binding mode together with the nonfatal modifier:

```
ld /lib/crt0.o -B immediate -B nonfatal program.o -L .\
     -lfunc -a archive -lc
```

Link a Pascal program on a Series 300/400 system:

```
ld /lib/crt0.o main.o -lpc -lm -lc
```

WARNINGS

`ld` recognizes several names as having special meanings. The symbol `_end` is reserved by the linker to refer to the first address beyond the end of the program's address space. Similarly, the symbol `_edata` refers to the first address beyond the initialized data, and the symbol `_etext` refers to the first address beyond the program text. The linker treats a user definition of any of these symbols as an error. The symbols `end`, `edata`, and `etext` are also defined by the linker, but only if the program contains a reference to these symbols and does not define them (see *end*(3C) for details).

Through its options, the link editor gives users great flexibility. However, those who invoke the linker directly must assume some added responsibilities. Input options should ensure the following properties for programs:

- When the link editor is called through *cc*(1), a start-up routine is linked with the user's program. This routine calls *exit*(2) after execution of the main program. If users call `ld` directly, they must ensure that the program always calls *exit*() rather than falling through the end of the entry routine.
- When linking for use with the symbolic debugger *xdb*, the user must ensure that the program contains a routine called `main`. Also, the user must link in the file `/usr/lib/end.o` as the last file named on the command line.

There is no guarantee that the linker will pick up files from archive libraries and include them in the final program in the same relative order that they occur within the library.

DEPENDENCIES

Series 300/400

The default entry point is taken to be text location 0x0 (which is also the default origin of the program text) if shared libraries are not used. Otherwise, the entry point is taken to be the first text location after the extension header placed at the beginning of the text segment by `ld` for use by `/lib/dld.sl`. This corresponds to the first procedure in the first input file that the linker reads. If the C startup routine `/lib/crt0.o` is the first object file on the command line, the label `start` denotes the entry point. Use the `-e` option to select a different entry point.

By default, the data segment begins on the next page following the text segment.

The version number specified with the `-V` option must be in the range 0 through 32767. Use of this option is not recommended because this field is used by several HP-UX commands that expect particular values here. Consult the *C Programmer's Guide* for more details on the version field.

The placement of `-L` options relative to `-l` is not significant.

The Series 300/400 linker does not support options `-m`, `-z`, and `-Z`.

On Series 300/400 systems, the compilers place an underscore at the beginning of all external names. Thus, the symbol `_end` appears to the linker as `__end`.

`ld` does not generate an output file if any other errors occur during its operation (see DESCRIPTION).

The default file type is `DEMAND_MAGIC` if the total size of text and data is greater than the constant DEMAND_THRESHOLD; otherwise the file type defaults to `SHARE_MAGIC`. DEMAND_THRESHOLD is defined in `/usr/include/a.out.h`, and this value may vary from release to release. See the manual *How HP-UX Works: Concepts for the System Administrator* for a discussion of the various file types.

Series 700/800

The linker searches for the symbol `$START$` as the program entry point. This symbol is defined in the file `/lib/crt0.o`, which should be the first file loaded for all programs regardless of source language. Use the `-e` option to select a different entry point.

By default, the data segment begins at offset 0x40001000. Since certain optimizations are now performed in the page previous to 0x40001000, some applications will not function as before if the start of the data segment is set to be within the range 0x40000000 through 0x40001000.

When invoking `ld` directly to link a C program whose `main` procedure is located in a library, the `-u main` option should be used to force the linker to load `main` from the library, since this symbol is not actually referenced until the `_start` routine is loaded from the C library. When using *cc*(1) to link the program, the compiler automatically passes this option to the linker. Because of this behavior, do not use *cc* to link a program containing a FORTRAN or Pascal main program; use *f77* or *pc* instead.

Nonsharable, executable files generated with the `-N` option cannot be executed via *exec*(2). Typically, `-N` is used when rebuilding the kernel or when preparing an image for dynamic loading. For these files, the data segment begins on the next page following the text segment.

When the `-A` option is used to do an incremental link, the linker generates extra code where a procedure call crosses a quadrant boundary (a quadrant is one gigabyte, or one fourth of the addressing space). On Series 700/800 systems, text is normally in the first quadrant and data is in the second quadrant. When an object file is intended to be read into an already-executing program, both its code and data must be placed in the second quadrant, since the first quadrant is set to read-only. Procedure calls from one quadrant to the other require the extra code, called inter-space calling stubs. The linker generates an "export" stub for the entry point designated in the incremental link, and "import" stubs for each procedure in the basis program that is called by the new object file. The import stubs require the existence of a routine in the basis program called `_sr4export`, which is supplied in `/lib/crt0.o`. If a procedure in the basis program is called indirectly by the new object file, the linker cannot detect the crossing of the quadrant boundary, and therefore will not generate the needed stub. A special version of `$$dyncall` placed in `/lib/dyncall.o` is provided for handling the inter-quadrant branch. This routine should be linked in when the `-A` option is in effect.

The Series 700/800 linker does not support the `-V` option.

`ld` treats both duplicate symbols and unresolved symbols in the same manner: an output file is generated and marked as non-executable if errors occur during its operation (see DESCRIPTION).

The default file type is `SHARE_MAGIC`. See the manual *How HP-UX Works: Concepts for the System Administrator* for a discussion of the various file types.

The Series 700/800 linker works in conjunction with the compilers to support profile-based optimization: transformations that use profile data gathered from previous runs of a program to improve run-time performance. Profile data collection and profile-based optimizations can be performed within the bodies of procedures compiled with the `+I` option to `cc` or `f77` (see *cc*(1) and *f77*(1)). For example, code can be repositioned within procedures so that fewer branches are executed and better code locality is achieved. The linker can also instrument calls between procedures, even if they were not compiled with the `+I` option. After profile data is collected, the program can be relinked and the linker will reposition the procedures to further improve code locality. See the **-I**, **-P**, **+df**, and **+pgm** options below. Also see the discussion of profile-based optimization in the *Programming on HP-UX* manual.

Object files that are compiled with the `+I` or `+P` option (see *cc*(1) and *f77*(1)) contain a compiler intermediate representation instead of PA-RISC object code. The linker then invokes a PA-RISC code generator on these files before linking them into the executable file. Because of this, linking object files compiled with `+I` or `+P` may take significantly more time than linking ordinary object files. The object files that are generated by compiling the intermediate code are written to the directory specified by the **TMPDIR** environment variable, or `/tmp` if **TMPDIR** is not specified. The object files are deleted before the linker exits.

The following environment variables are specific to the Series 700/800 linker:

FLOW_DATA Used by the profile-based optimization feature of the linker. An instrumented executable (see `-I` option) writes out the profile data to a database file named `flow.data` in the current directory. The name and location of this file can be specified by setting **FLOW_DATA** to the desired path name. The profile data is stored in the database file under a look-up name that is the same as the basename of the executable file specified at run-time. A single `flow.data` file can hold profile data for multiple program files.

FLOW_DATA_DIR Obsolete environment variable. Do not use.

The following options are specific to the Series 700/800 linker:

-a *search* The Series 700/800 linker supports two additional values for the search flag. The value `archive_shared` indicates that within each directory searched to find the library specified with **-l**, the archive form is preferred, but the shared form is allowed. The value `shared_archive` states that the shared form is preferred but the archive form is allowed.

-l: *library* Search the library specified. Similar to the **-l** option except the current state of the **-a** option is not important. The library name must contain the prefix **lib** and end with a suffix of **.a** or **.sl**.

-B *bind* Along with the major binding modes of `immediate` and `deferred`, the bind modifier `restricted` can be specified to indicate that the symbols visible to the shared libraries needed by the executable output file should be limited to the symbols available at program start-up. See the *Programming on HP-UX* manual for common uses of binding modes.

-C*n* Set the maximum parameter-checking level to *n*. The default maximum is 3. See the language manuals for the meanings of the parameter-checking level.

-G Strip all unloadable data from the output file. This option is typically used to strip debug information.

-I Instrument the code to collect profile information upon execution. The profile data gathered during program execution can be used in conjunction with the **-P** option. Programs linked with this option should use the startup file `/lib/icrt0.o` instead of `/lib/crt0.o`. This option should not be used together with the **-b**, **-s**, **-G**, **-A**, or **-N** options.

-O Turn on linker optimizations. Currently the optimizations include the elimination of unnecessary **ADDIL** instructions from the code in the executable file. The linker also rearranges data to further reduce the number of **ADDIL** instructions in the executable.

This optimization may reveal some subtle programming errors related to assumptions about data layout.

`-O` is passed on to the linker by the compilers when the `+O3` compiler option is selected.

`-P` Examine the data file produced by an instrumented program (see the `-I` option) to perform profile based optimizations on the code. This option should not be used together with the `-A`, or `-N` options.

`-S` Generate an Initial Program Loader (IPL) auxiliary header for the output file, instead of the default HP-UX auxiliary header.

`-T` Save the load data and relocation information in temporary files instead of memory during linking. This option reduces the virtual memory requirements of the linker. If the `TMPDIR` environment variable is set, the temporary files are created in the specified directory, rather than in `/tmp`.

`+df` *file* Used together with the `-P` option, this option specifies that *file* should be used as the profile database file. The default value is `flow.data`. See the discussion of the `FLOW_DATA` environment variable above.

`+pgm` *name* Used together with the `-P` option, this option specifies that *name* should be used as the look-up name in the profile database file. The default is the basename of the output file (specified by the `-o` option.)

`+FP` *flags* Specify how the environment for floating-point operations should be initialized at program start-up. By default, all behaviors are disabled. The following flags are supported (uppercase flag enables; lowercase flag disables):

`V` (`v`)	Trap on invalid floating-point operations
`Z` (`z`)	Trap on divide by zero
`O` (`o`)	Trap on floating-point overflow
`U` (`u`)	Trap on floating-point underflow
`I` (`i`)	Trap on floating-point operations that produce inexact results.
`D` (`d`)	Enable sudden underflow (flush to zero) of denormalized values.

Note: Enabling sudden underflow is an undefined operation on PA-RISC 1.0-based systems, but it is defined on all subsequent versions of PA-RISC. Selecting this flag enables sudden underflow only if it is available on the processor being used at run-time.

To dynamically change these settings at run-time, see *fpgetround*(3M).

AUTHOR

`ld` was developed by AT&T, the University of California, Berkeley, and HP.

FILES

`/lib/lib`*x*`.a`	archive libraries
`/usr/lib/lib`*x*`.a`	archive libraries
`/lib/lib`*x*`.sl`	shared libraries
`/usr/lib/lib`*x*`.sl`	shared libraries
`a.out`	output file
`/lib/dld.sl`	dynamic loader
`/usr/lib/end.o`	for use with `xdb` debugger

Series 300/400

`/lib/crt0.o`	run-time start-up for C and Pascal
`/lib/mcrt0.o`	run-time start-up for C and Pascal with profiling (see *prof*(1))
`/lib/gcrt0.o`	run-time start-up for C and Pascal with profiling (see *gprof*(1))
`/lib/frt0.o`	run-time start-up for FORTRAN
`/lib/mfrt0.o`	run-time start-up for FORTRAN with profiling (see *prof*(1))
`/lib/gfrt0.o`	run-time start-up for FORTRAN with profiling (see *gprof*(1))
`/usr/lib/end.o`	for use with `cdb`, `fdb`, or `pdb` debugger

Series 700/800

`/lib/crt0.o`	run-time start-up
`/lib/dyncall.o`	used with **-A**-option links
`/lib/mcrt0.o`	run-time start-up with profiling (see *prof*(1))
`/lib/milli.a`	millicode library automatically searched by `ld`
`/lib/gcrt0.o`	run-time start-up with profiling (see *gprof*(1))
`/lib/icrt0.o`	run-time start-up with profiling (see discussion of profile based optimization above.)
`/usr/lib/nls/$LANG/ld.cat`	message catalog
`/tmp/ld*`	temporary files
`flow.data`	file containing profile data generated by running an instrumented executable
`/bin/fdp`	program for creating the procedure link order from a profile database file created by an instrumented executable; forked by the `-P` option
`/usr/lib/uccom`	PA-RISC code generator for the C language
`/usr/lib/uf77pass1`	PA-RISC code generator for the FORTRAN language
`/usr/lib/ucomp`	Default PA-RISC code generator

SEE ALSO

ar(1), cc(1), cdb(1), chatr(1), f77(1), gprof(1), nm(1), pc(1), prof(1), strip(1), exec(2), crt0(3), end(3C), a.out(4), ar(4), dld.sl(5).

Programming on HP-UX.

How HP-UX Works: Concepts for the System Administrator.

CC(1) in optional C++ compiler documentation.

STANDARDS CONFORMANCE

`ld`: SVID2, XPG2

NAME
leave - remind you when you have to leave

SYNOPSIS
`leave` [*hhmm*]

DESCRIPTION
`leave` waits until the specified time, then reminds you to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. When you log off, `leave` exits.

The time of day is in the form *hhmm*, where *hh* is a time in hours (which can range from 0 through 11 or 0 through 24 hours), and *mm* is the number of minutes after the specified hour. If the value of *hh* is greater than 11 (24-hour clock time), the specified value is reduced by 12 to a new value in the range of 0 through 11, thus ensuring that the alarm time is always set to activate within the next 12 hours. For example, if *hhmm* is 1350 and the current time is 4:00 PM (1600), the 1350 value is changed to 150 and the alarm is set for 1:50 AM, nine hours and 50 minutes later. On the other hand, if it is 9:00 AM and *hhmm* is specifed as 2200 (10:00 PM), the value used is converted to 1000 and the alarm is set for one hour later instead of 13 hours as specified.

If no argument is provided, `leave` prompts with `When do you have to leave?` A reply of newline causes `leave` to exit; otherwise the reply is assumed to be a time. This form is suitable for inclusion in a `.login` or `.profile` file.

`leave` ignores interrupts, quits, and terminate signals. To get rid of it you should either log off or use `kill -9` giving its process ID.

EXAMPLES
The command

```
leave 1204
```

sends an alarm (a beep) to your terminal to remind you that you have to leave at 12:04 and reminds you that you are late at one minute intervals after 12:04.

WARNINGS
`leave` checks to see if a user has logged out by checking the `/etc/utmp` file every 100 seconds. If a user logs out and logs back in to the same tty before `leave` makes its periodic check, `leave` might not know that the user has logged out.

FILES
`/etc/utmp`

AUTHOR
`leave` was developed by the University of California, Berkeley.

SEE ALSO
calendar(1).

NAME

lex - generate programs for lexical analysis of text

SYNOPSIS

`lex` [`-rctvn`] [`-X`*secondary n...*] [*file*] ...

DESCRIPTION

`lex` generates programs to be used in simple lexical analysis of text.

The input *file*s contain strings and expressions to be searched for, and C text to be executed when strings are found. Multiple files are treated as a single file. If no files are specified, the standard input is used.

A file `lex.yy.c` is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in `yytext`, an external character array. Matching is done in order of the strings in the file. The strings can contain square brackets to indicate character classes, as in `[abx-z]` to indicate `a`, `b`, `x`, `y`, and `z`; and the operators `*`, `+`, and `?` mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character `.` is the class of all ASCII characters except new-line. When 16-bit support is enabled, the character `.` also matches all valid 16-bit characters under the current locale in addition to the the ASCII characters. Parentheses for grouping and vertical bar for alternation are also supported. The notation *r*`{`*d*`,`*e*`}` in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than `|`, but lower than `*`, `?`, `+`, and concatenation. The character `^` at the beginning of an expression permits a successful match only immediately after a new-line, and the character `$` at the end of an expression requires a trailing new-line. The character `/` in an expression indicates trailing context; only the part of the expression up to the slash is returned in `yytext`, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is enclosed between double quotes (") or preceded by `\`. Thus `[a-zA-Z]+` matches a string of letters.

Three subroutines defined as macros are expected: `input()` to read a character; `unput(`*c*`)` to replace a character read; and `output(`*c*`)` to place an output character. They are defined in terms of the standard streams, but can be overridden. The program generated is named `yylex()`, and the library contains a `main()` which calls `setlocale()` then calls `yylex()`. Users can define their own version of `main()`, but if `lex.yy.c` is generated using `-w` or `-m` and the locale used is other than the default, the user-defined `main()` routine must include a call similar to `setlocale (LC_ALL, yylocale)` before `yylex()` is called, or the actions of the generated scanner will be undefined. The generated `lex.yy.c` program includes the appropriate declaration and initialization of `yylocale`. The action `REJECT` on the right side of the rule causes this match to be rejected and the next suitable match executed; the function `yymore()` accumulates additional characters into the same `yytext`; and the function `yyless(`*p*`)` pushes back the portion of the string matched beginning at *p*, which should be between `yytext` and `yytext+yyleng`. The macros `input` and `output` use files `yyin` and `yyout` to read from and write to, defaulted to the standard input and the standard output, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes `%%` it is copied into the external definition area of the `lex.yy.c` file. All rules should follow a `%%`, as in `yacc` (see *yacc*(1)). Lines preceding `%%` that begin with a non-blank character define the string on the left to be the remainder of the line. This is called a definition, and can be called out later by surrounding it with `{ }`. Note that curly brackets do not imply parentheses; only string substitution is done.

Options

`lex` recognizes the following options, which must appear before any *file*s:

`-r`	Indicates `ratfor` actions (see *ratfor*(1));
`-c`	Indicates C actions – this is the default;
`-m`	Enables basic multibyte support. This option allows intermixed ASCII and 16-bit characters in a lex specification. 16-bit characters can appear in quoted and unquoted strings, regular expressions, character classes, definitions, definition names, and as endpoints of ranges. This option also enables the `.` character to recognize any valid 16-bit character as well as ASCII characters. With this option, metacharacters such as `*`, `+`, and `?` can be applied to 16-bit characters the same way they are applied to ASCII characters.

`-n`	Suppresses printing of the - summary.
`-t`	Causes the **`lex.yy.c`** program to be written instead to the standard output;
`-v`	Provides a one-line summary of statistics for the machine generated;
`-w`	Enables basic multibyte support and causes the underlying data type returned to the user, **`yytext`**, to be an array of type **`wchar_t`**. This option takes precedence over the `-m` option.
`-X`*secondary n*	Resets the sizes of certain internal **`lex`** tables. *secondary* is a single letter from the set {`dDsSac`} that specifies the table to be reset; *n* is the new size:

`d`	Table of definitions; default=200.
`D`	Table of characters in definition strings; default=5000.
`s`	Table of start conditions; default=50.
`S`	Table of characters in start condition names; default=500.
`c`	Array table for storing character classes; default=1000.
`a`	Right context/action array table; default=100.

If an array overflows, **`lex`** issues a fatal error message including a suggestion of which table to reset. For example:

```
Definitions too long, try -XD option
```

Certain table sizes for the resulting finite state machine can be set in the definitions section:

`%p` *n*	number of positions is *n* (default is 2500);
`%q` *n*	number of positions for one state is *n* (default is 300);
`%n` *n*	number of states is *n* (default is 500);
`%e` *n*	number of parse tree nodes is *n* (default is 1000);
`%a` *n*	number of transitions is *n* (default is 2000).
`%k` *n*	number of packed character classes is *n* (default is 1000);
`%o` *n*	size of output array is *n* (default is 3000);

The use of one or more of the preceding table options automatically implies `-v` unless `-n` is specified.

Other recognized directives in the definitions section:

`%l` *locale* specifies the value of the LANG environment variable when the final scanner is run. *locale* is a quoted or unquoted string such as `japanese` or **`chinese-t`**. The character string **`yylocale`** is set to the value of *locale* at runtime and the default **`main()`** subroutine provided in the lex library, **`libl`**, calls **`setlocale (LC_ALL, yylocale)`**. *locale* is also used to evaluate character attributes when reading the input specification and is used for analyzing the character set when building the tables in **`lex.yy.c`**. If the value of *locale* indicates that the basic character size is 16-bits, it will automatically enable the `-m` option.

External names generated by **`lex`** all begin with the prefix **`yy`** or **`YY`**.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the size of the characters in use unless overridden by the `%l` *locale* source directive.

`LC_MESSAGES` determines the language in which messages are displayed.

`LANG` is used as a default if `LC_CTYPE` or `LC_MESSAGES` is not set.

International Code Set Support

Single- and multi-byte character code sets are supported. Multi-byte character code set support is enabled with `-w` or `-m`.

EXAMPLES

```
D       [0-9]
%%
if      printf("IF statement\n");
[a-z]+ printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: goto loop;
                        }
                }
```

WARNINGS

The `-r` option is not yet fully operational.

The ^ operator is not supported in character classes, [], containing multi-byte characters.

The token buffer in the program built by `lex` is of fixed length,

`yytext[YYLMAX]`

where **YYLMAX** is defined to be 200 unsigned characters or 400 unsigned characters if `-m` has been specified and `LC_CTYPE` indicates a multi-byte character set. Overflow of this array is not detected in the `lex.yy.c` program.

SEE ALSO

yacc(1), malloc(3X).

LEX – Lexical Analyzer Generator in *C Programming Tools*.

STANDARDS CONFORMANCE

`lex`: SVID2, XPG2, XPG3, POSIX.2

NAME

lifcp - copy to or from LIF files

SYNOPSIS

lifcp [**-T***xxx*] [**-L***xxx*] [**-v***xxx*] [**-a**] [**-b**] [**-i** *xxx*] [**-r**] [**-t**] *file1 file2*
lifcp [**-T***xxx*] [**-L***xxx*] [**-v***xxx*] [**-a**] [**-b**] [**-i***xxx*] [**-r**] [**-t**] [*file1 file2 ...*] *directory*

DESCRIPTION

lifcp copies a LIF file to an HP-UX file, an HP-UX file to a LIF file, or a LIF file to another LIF file. It also copies a list of (HP-UX/LIF) files to a (LIF/HP-UX) directory. The last name on the argument list is the destination file or directory.

Options can be used singly or combined in any order before the file names. The space between option and argument is optional.

-T*xxx* Used only when copying files to a LIF volume. This option forces the file type of the LIF directory entry to be set to the argument given. The argument can be decimal, octal or hex, using standard "C" notation.

-L*xxx* Used only when copying files to a LIF volume. This option will set the "last volume flag" to *xxx* (0 or 1). The default "last volume flag" is **1**.

-v*xxx* Used only when copying files to a LIF volume. This option sets the "volume number" to *xxx*. The default "volume number" is one.

-a This option forces a ASCII mode of copying regardless of the file type. When copying in ASCII mode from HP-UX to LIF the default file type is BINARY (1). (For details on available modes of copying refer to *lif*(4)). This option has no effect when copying from LIF to LIF.

-b This option forces a BINARY mode of copying regardless of the file type. When copying in BINARY mode from HP-UX to LIF the default file type is BINARY (-2). (For details on available modes of copying refer to *lif*(4)). This option has no effect when copying from LIF to LIF.

-i*xxx* Used only when copying files to a LIF volume. This option sets the "implementation" field of the LIF directory entry to the argument given. The argument value can be decimal, octal or hex, using standard "C" notation. The "implementation" field can only be set for file types -2001 to -100000 (octal). The "implementation" field is set to zero for all interchange file types and for file types -2 to -200 (octal). Note that the "implementation" value controls the attributes of the LIF file with regard to protection and record sizes. **lifls -l** or **lifls -i** can be used to determine the "implementation" value of a file.

-r Forces RAW mode copying regardless of file type. When copying in RAW mode from HP-UX to LIF the default file type is BIN (-23951). **-T** option overrides the default file type. (various modes of copying are explained in *lif*(4).) -r option has no effect in LIF to LIF copy operations.

-t causes HP-UX file names to be translated to a name acceptable by a LIF utility; that is, all lowercase letters are converted to uppercase and all other characters except numerics are changed to an underscore (_). If the HP-UX file name starts with a non-letter, the file name is preceded by the capital letter X. Thus, for example, if two files named colon (:) and semicolon (;), were copied, both of them would be translated to X_. File names are truncated to a maximum of 10 characters. When copying a LIF file to an HP-UX or LIF file, **-t** has no effect. Omitting **-t** causes an error to be generated if an improper name is used.

The default copying modes when copying from LIF to HP-UX are summarized in the following table:

File Type	Default Copying Mode
ASCII	ASCII
BINARY	BINARY
BIN	RAW
other	RAW

When copying from HP-UX to LIF, the default copying mode is ASCII and an ASCII file is created.

When copying from LIF to LIF, if no options are specified, then all the LIF directory fields and file contents are duplicated from source to destination.

A LIF file name is recognized by the embedded colon (:) delimiter (see *lif*(4) for LIF file naming conventions). A LIF directory is recognized by a trailing colon. If an HP-UX file name containing a colon is used, the colon must be escaped with two backslash characters (\\) (the shell removes one of them).

The file name - (dash) is interpreted to mean standard input or standard output, depending on its position in the argument list. This is particularly useful if the data requires non-standard translation. When copying from standard input, if no other name can be found, the name "STDIN" is used.

LIF file naming conventions are known only to the LIF utilities. Since file name expansion is done by the shell, this mechanism cannot be used for expanding LIF file names.

Do not mount the special file while using *lifcp*.

DEPENDENCIES

Series 700/800

The following option is also supported:

-K*nnn*	forces each file copied in to begin on a *nnn* × 1024-byte boundary from the beginning of the volume. This is useful when files are used for Series 700/800 boot media. This option has no effect when copying from a LIF volume.

EXAMPLES

Copy HP-UX file **abc** to LIF file **CDE** on LIF volume **lifvol** which is actually an HP-UX file initialized to be a LIF volume:

```
lifcp abc lifvol:CDE
```

Copy all the HP-UX files in the current directory to the LIF volume **lifvol** which is present in the parent directory. File names are translated to appropriate LIF file names.

```
lifcp -t * ../lifvol:
```

Copy all the HP-UX object files in the current directory to the LIF volume *lifvol*. Copying mode is RAW and LIF file types are set to -5555.

```
lifcp -r -T -5555 -t *.o
```

Copy all the object files in the current directory to the LIF volume **lifvol**. Copying mode is BINARY and LIF BINARY files are created.

```
lifcp -r -T 0xffffe961 -i 0x20200080 bdat lifvol:BDAT
```

Copy a BDAT file, without a password, from a BASIC WorkStation to an HP-UX LIF volume **lifvol**. Note that **-i** controls protection and record size attributes. The file type for a BDAT file is -5791 (or 0xffffe961) and its record size is 256 bytes per record.

```
lifcp -b *.o lifvol:
```

Copy all files in the current directory to the LIF volume **lifvol** in the **root** directory. Copying mode is RAW and LIF file types are set to BIN.

```
lifcp -r -t * /lifvol:
```

Copy file **abc:** to LIF file **CDE** in **lifvol**.

```
lifcp abc\\: lifvol:CDE
```

Copy files **abc** and **def** to LIF files **ABC** and **DEF** within **lifvol**.

lifcp -t abc def lifvol:

Copy LIF file **ABC** within **lifvol** to file **ABC** within current directory.

lifcp lifvol:ABC .

Copy standard input to LIF file **A_FILE** on LIF volume **/dev/dsk/1s2**.

lifcp - /dev/dsk/1s2:A_FILE

Copy LIF file **ABC** in **lifvol** to LIF file **CDE on /dev/dsk/1s2 .**

lifcp lifvol:ABC /dev/dsk/1s2:CDE

Copy the output of *pr* to the LIF file **ABC**.

pr abc | lifcp - lifvol:ABC

Copy the output of *pr* to the LIF volume **lifvol**. LIF file **STDIN** is created since no files names are specified.

pr abc | lifcp - lifvol:

Copy LIF file **ABC** in **lifvol** to *standard output*.

lifcp lifvol:ABC -

Copy all files in current directory to LIF files of the same name on LIF volume **lifvol** (may cause errors if file names in the current directory do not obey LIF naming conventions!).

lifcp * ./lifvol:

AUTHOR

lifcp was developed by the Hewlett-Packard Company.

SEE ALSO

lifinit(1), lifls(1), lifrename(1), lifrm(1), lif(4).

DIAGNOSTICS

lifcp returns exit code 0 if the file is copied successfully. Otherwise it prints a diagnostic and returns non-zero.

NAME
lifinit - write LIF volume header on file

SYNOPSIS
`lifinit` [-v*nnn*] [-d*nnn*] [-n *string*] *file*

DESCRIPTION
`lifinit` writes a LIF volume header on a volume or file.

Options
`lifinit` recognizes the following options and command-line arguments which can appear in any order:

-v*nnn*	Sets volume size to *nnn* bytes. If *nnn* is not a multiple of 256, it is rounded down to the next such multiple.
-d*nnn*	Sets directory size to *nnn* file entries. If *nnn* is not an integer multiple of 8, it is rounded up to next such multiple.
-n *string*	Sets the volume name to be *string*. If the `-n` option is not specified, the volume name is set to the last component of the path name specified by *file*. A legal LIF volume name is 6 characters long and is limited to uppercase letters (A-Z), digits (0-9) and the underscore character (_). The first character (if any) must be a letter. The utility automatically performs translation to create legal LIF volume names. Therefore, all lowercase letters are converted to uppercase, and all other characters except numeric and underscore are replaced with a capital letter **X**. If the volume name does not start with a letter, the volume name is preceded by a capital letter **X**. The volume name is also right-padded with spaces or truncated as needed to be six characters long. If `-n` is used with no *string*, the default volume name is set to 6 spaces.

If *file* does not exist, a regular HP-UX disk file is created and initialized.

The default values for volume size are 256 Kbytes for regular files, and the actual capacity of the device for device files.

The default directory size is a function of the volume size. A percentage of the volume size is allocated to the volume directory as follows:

Volume Size	Directory Size
< 2MB	~1.3%
> 2MB	~0.5%

Each directory entry occupies 32 bytes of storage. The actual directory space is subject to the rounding rules stated above.

Do not mount the special file while using `lifinit`.

RETURN VALUE
`lifinit` returns exit code 0 if the volume is initialized successfully. Otherwise it prints a diagnostic message and returns non-zero.

WARNINGS
To prevent media corruption, do not terminate *lifinit* once it has started executing.

DEPENDENCIES
Series 300/400
Media that has never been initialized must be initialized using `mediainit` (see *mediainit*(1)) before `lifinit` can be used (refer to HP-UX System Administrator manuals for details concerning `mediainit`).

Series 700/800
The following options are also supported:

-s*nnn*	set the initial system load (ISL) start address to *nnn* in the volume label. This is useful when building boot media for Series 700/800 systems.
-l*nnn*	specifies the length in bytes of the ISL code in the LIF volume.
-e*nnn*	set the ISL entry point to *nnn* bytes from the beginning of the ISL. For example, specifying `-e3272` means that the ISL entry point is 3272 (decimal) bytes from the beginning of the ISL object module.

-K*nnn* forces the directory start location to be the nearest multiple of *nnn* × 1024 bytes from the beginning of the volume. This is necessary for booting Series 700/800 systems from LIF media.

EXAMPLES

```
lifinit -v500000 -d10 x
lifinit /dev/rdsk/1s2
```

AUTHOR

`lifinit` was developed by HP.

SEE ALSO

lifcp(1), lifls(1), lifrename(1), lifrm(1), lif(4).

NAME

lifls - list contents of a LIF directory

SYNOPSIS

`lifls [option] name`

DESCRIPTION

`lifls` lists the contents of a LIF directory on standard output. The default output format lists file names in multiple columns (similar to *ls*(1), except unsorted) if standard output is a character special file. If standard output is not a tty device, the output format is one file name per line. *name* is a path name to an HP-UX file containing a LIF volume and optional file name. If *name* is a volume name, the entire volume is listed. If *name* is of the form *volume*`:`*file*, only the file is listed. The following options are available, and only one option should be specified with a given command:

`-l`	List in long format, giving volume name, volume size, directory start, directory size, file type, file size, file start, "implementation" field (in hex), date created, last volume, and volume number.
`-C`	Force multiple column output format regardless of standard output type.
`-L`	Return the content of the "last volume flag" in decimal.
`-i`	Return the content of the "implementation" field in hex.
`-v`	Return the content of the "volume number" in decimal.

Do not mount the special file while using `lifls`.

DIAGNOSTICS

`lifls` returns zero if the directory was listed successfully. Otherwise it prints a diagnostic and returns non-zero.

EXAMPLES

```
lifls -l ../TEST/header
lifls -C /dev/rdsk/1s2
```

AUTHOR

`lifls` was developed by HP.

SEE ALSO

lifcp(1), lifinit(1), lifrename(1), lifrm(1), lif(4).

NAME
lifrename - rename LIF files

SYNOPSIS
`lifrename` *oldfile newfile*

DESCRIPTION
oldfile is a full LIF file specifier (see *lif*(4) for details) for the file to be renamed (e.g. `liffile:A_FILE`). *newfile* is new name to be given to the file (only the file name portion). This operation does not include copy or delete. Old file names must match the name of the file to be renamed, even if that file name is not a legal LIF name.

Do not mount the special file while using `lifrename`.

DIAGNOSTICS
`lifrename` returns zero if the file name is changed successfully. Otherwise it prints a diagnostic and returns non-zero.

EXAMPLES
```
lifrename liffile:A_FILE B_FILE
lifrename /dev/dsk/1s2:ABC CDE
```

AUTHOR
`lifrename` was developed by HP.

SEE ALSO
lifcp(1), lifinit(1), lifls(1), lifrm(1), lif(4).

NAME

lifrm - remove a LIF file

SYNOPSIS

`lifrm` *file1 ... filen*

DESCRIPTION

`lifrm` removes one or more entries from a LIF volume. File name specifiers are as described in *lif*(4).

Do not mount the special file while using `lifrm`.

DIAGNOSTICS

`lifrm` returns zero if the file is removed successfully. Otherwise it prints a diagnostic and returns non-zero.

EXAMPLES

```
lifrm liffile:MAN
lifrm /dev/rdsk/1s2.0:F
```

AUTHOR

`lifrm` was developed by HP.

SEE ALSO

lifcp(1), lifinit(1), lifls(1), lifrename(1), lif(4).

NAME
line - read one line from user input

SYNOPSIS
`line` [`-t` *timeout*]

DESCRIPTION
`line` copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

Options
`line` recognizes the following command-line option:

`-t` *timeout* Timeout after *timeout* seconds where *timeout* is an integer value (if a non-integer value is specified, it is converted to an integer; i.e., rounded down). A blank is required between `-t` and the *timeout* argument. This option is not documented in POSIX and other industry standards, and should not be used in portable applications.

EXTERNAL INFLUENCES
International Code Set Support
Single- and multi-byte character code sets are supported.

EXAMPLES
The following lines in a shell script prompt for a file name and display information about the file:

```
echo 'Enter file name: \c'
reply='line'
ls -l $reply
```

To limit the response time to 10 seconds, use the form:

```
reply='line -t 10'
```

then test for no response. If no response occurs before timeout expires, a default behavior should be provided.

SEE ALSO
sh(1), read(2).

STANDARDS CONFORMANCE
`line`: SVID2, XPG2, XPG3

NAME
lint - a C program checker/verifier

SYNOPSIS
`lint` [*options*] *file* ...

DESCRIPTION
`lint` attempts to detect features in C program *files* that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Program anomalies currently detected include unreachable statements, loops not entered at the top, automatic variables declared but not used, and logical expressions whose value is constant. Usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with `.c` are assumed to be C source files. Arguments whose names end with `.ln` are assumed to be the result of an earlier invocation of `lint` with either the `-c` or the `-o` option used. `.ln` files are analogous to `.o` (object) files produced by the `cc` or `c89` command (see *cc*(1) when given a `.c` file as input. Files with other suffixes are warned about and ignored.

`lint` takes all the `.c`, `.ln`, and `llib-l`*x*`.ln` files (specified by `-l`*x*) and processes them in their command-line order. By default, `lint` appends the standard C lint library (`llib-lc.ln`) to the end of the list of files. However, if the `-p` option is used, the portable C lint library (`llib-port.ln`) is appended instead. When the `-c` option is not used, the second pass of `lint` checks this list of files for mutual compatibility. When the `-c` option is used, all `.ln` and `llib-l`*x*`.ln` files are ignored.

Options
Any number of `lint` options can be used, in any order, intermixed with file name arguments. The following options are used to suppress certain kinds of complaints:

- **-a** Suppress complaints about assignments of long values to variables that are not long.
- **-b** Suppress complaints about `break` statements that cannot be reached. (Programs produced by `lex` or `yacc` often result in many such complaints).
- **-h** Do not apply heuristic tests that attempt to intuitively find bugs, improve style, and reduce waste.
- **-u** Suppress complaints about functions and external variables used and not defined or defined and not used. (This option is suitable for running `lint` on a subset of files of a larger program.)
- **-v** Suppress complaints about unused arguments in functions.
- **-x** Do not report variables referred to by external declarations but never used.

The following arguments alter `lint`'s behavior:

- **-l***x* Include additional lint library `llib-l`*x*`.ln`. For example, to include a lint version of the Math Library `llib-lm.ln`, insert `-lm` on the command line. This argument does not suppress the default use of `llib-lc.ln`. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multiple-file projects.
- **-n** Do not check compatibility against either the standard or the portable lint library.
- **-p** Attempt to check portability to other dialects of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.
- **-s** Make stricter checks about pointer and structure alignments that can prevent portability. Complain about a cast that converts a pointer from a less restrictive alignment to a more restrictive alignment. Complain about a structure member whose offset is not a multiple of its size.
- **-c** Cause `lint` to produce a `.ln` file for every `.c` file on the command line. These `.ln` files are the product of `lint`'s first pass only, and are not checked for inter-function compatibility.

-o *lib* Cause `lint` to create a lint library with the name `llib-l`*lib*`.ln`. The `-c` option nullifies any use of the `-o` option. The resulting lint library serves as input to `lint`'s second pass. The `-o` option simply causes this file to be saved in the named lint library. To produce a `llib-l`*lib*`.ln` without extraneous messages, use the `-x` option. The `-v` option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file `llib-lc` is written). These option settings are also available by using "lint comments" (see below).

-A*mode* Specify the compilation standard to be used by `lint`. The *mode* can be one of the following letters:

- c Process in a mode compatible with HP-UX releases prior to 7.0. (See *The C Programming Language*, First Edition by Kernighan and Ritchie). This option is currently the default. The default may change in future releases.
- a Process under ANSI mode (December 7, 1988 Draft proposed ANSI C standard.)

-Y
Enable support of 16-bit characters inside string literals and comments. Note that 8-bit parsing is always supported. See *hpnls*(5) for more details on international language support.

The `-D`, `-U`, and `-I` options to `cpp` and the `-g` and `-O` options to `cc` are also recognized as separate arguments. The `-g` and `-O` options are ignored, but, by recognizing these options, `lint`'s behavior is closer to that of the `cc` command. Other options are warned about and ignored. The pre-processor symbols `__lint` and `__LINT__` are defined to allow certain questionable code to be altered or removed for `lint`. In addition, the pre-processor symbol `lint` is defined in compatibility mode. By default, the lint library `llib-lc.ln` encodes the HP-UX namespace version of `libc.a`. Other standards can be checked by including the appropriate `-D` option on the `lint` command line. For example,

```
lint -D_POSIX_SOURCE file.c
```

reprocesses `llib-lc` to reflect the POSIX standard.

Certain conventional comments in the C source change the behavior of `lint`:

/*NOTREACHED*/	at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions such as `exit()`).
/*VARARGS*n**/	suppresses the usual checking for variable numbers of arguments in the following function definition. This comment must be placed just before the actual code for a function. It is not used before `extern` declarations of the same function elsewhere. The data types of the first *n* arguments are checked; a missing *n* is assumed to be 0.
/*ARGSUSED*/	enables the `-v` option for the next function.
/*LINTLIBRARY*/	at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the `-v` and `-x` options.

`lint` produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the `-c` option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name is printed, followed by a question mark.

Behavior of the `-c` and `-o` options allows for incremental use of `lint` on a set of C source files. Generally, one invokes `lint` once for each source file with the `-c` option. Each of these invocations produces a `.ln` file which corresponds to the `.c` file, and prints all messages that are about just that source file. After all the source files have been separately run through `lint`, it is invoked once more (without the `-c` option), listing all the `.ln` files with the needed `-l`*x* options. This prints all the inter-file inconsistencies. This scheme works well with **make**; allowing **make** to be used to `lint` only the source files that have been modified since the last time the set of source files were processed by `lint` (see *make*(1)).

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of comments and string literals as single- and/or multi-byte characters.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `lint` behaves as if all internationalization variables are set to "C". See *environ*(5).

When set, the `TMPDIR` environment variable specifies a directory to be used for temporary files, overriding the default directories `/tmp` and `/usr/tmp`.

Long error messages are split across lines to make them easier to read. The environment variable `COLUMNS` controls the maximum number of characters on each line.

International Code Set Support

Single- and multi-byte character code sets are supported within file names, comments, and string literals.

FILES

`/usr/lib`	the directory where the lint libraries specified by the `-l`*x* option must exist
`/usr/lib/lint[12]`	first and second passes
`/usr/bin/lint`	shell script that invokes `lint[12]`
`/usr/lib/llib-lc.ln`	declarations for C Library functions (binary format; source is in `/usr/lib/llib-lc`)
`/usr/lib/llib-port.ln`	declarations for portable functions (binary format; source is in `/usr/lib/llib-port`)
`/usr/lib/llib-lm.ln`	declarations for Math Library functions (binary format; source is in `/usr/lib/llib-lm`)
`/usr/tmp/*lint*`	temporaries

WARNINGS

`exit()`, `longjmp()`, and other functions that do not return are not understood (see *exit*(2) and *setjmp*(3C)); this causes various inaccuracies.

SEE ALSO

cc(1), cpp(1), make(1).

Lint C Program Checker tutorial in *C Programming Tools* manual.

STANDARDS CONFORMANCE

`lint`: SVID2, XPG2, XPG3

NAME

lintfor - FORTRAN inter-procedural checker

SYNOPSIS

`lintfor` [-p] [-u] [-V] [-l*x*] [-x *exclusions*] [-T] [-f *flistfile*] [-c] [-o *lib*] [*f77_options*] *file* ...

DESCRIPTION

`lintfor` detects erroneous, non-portable, or wasteful use of the FORTRAN language. It pays particular attention to mismatches between calls and definitions of **FUNCTION**s and **SUBROUTINE**s, **COMMON** block differences, and the use of uninitialized variables.

The following are problems that `lintfor` detects:

- Unused variables (declared but never referenced).
- Variables uninitialized (undefined) at the time of reference.
- Unreferenced labels.
- Mismatches in the number and types of formal and actual arguments to **FUNCTION**s and **SUBROUTINE**s.
- Mismatches in the expected and actual return value of **FUNCTION**s.
- **CALL**s of **FUNCTION**s, and the use of **SUBROUTINE**s in expressions.
- **COMMON** block layout differences

`lintfor` can also generate the following information:

- Call graph (which procedures call which procedures).
- List of procedures referenced but not defined in the source made available to `lintfor`.

Arguments whose names end with `.f` are interpreted as FORTRAN source files. Arguments whose names end with `.ln` are interpreted as the result of an earlier invocation of `lintfor -c` or `lintfor -o`. In addition, the `-f` *flistfile* option can be used to name a file containing a list of file names (one per line) to check. `lintfor` produces a warning if a file with any other suffix is specified.

`lintfor` takes all `.f`, `.ln`, and `fllib-l`*x*`.ln` files (specified by `-l`*x*) and processes them in their command-line order. If the `-c` option is not used, the second pass of `lintfor` checks this list of files for mutual compatibility. When the `-c` option is used, the `.ln` and `fllib-l`*x*`.ln` files are ignored.

Any number of `lintfor` options can be used, in any order, intermixed with file name arguments.

Options

The following options are used to change the type of output produced by `lintfor`:

`-x` *exclusions* Prevent pass 2 from reporting the listed errors and warnings. `lintfor` assigns an error number to each error and warning that it generates. To suppress the reporting of an error or warning, list its error number in a `-x` option. For example,

```
lintfor -x 1010,5001 file.f
```

does not report error number 1010 or warning number 5001 during interprocedural checking. Note that the final statistics about the number of errors and warnings found are reported as if the `-x` option were unspecified.

`-T`

Prevent pass 2 from displaying the text of errors and warnings. `lintfor` usually displays an error indication on one line followed by a textual explanation of the error message on the next line. The `-T` option suppresses the display of the textual message.

`-p` Print a call graph of the program. The call graph consists of a list of those procedures called by each procedure and a list of those procedures which call that procedure.

`-u` Suppress complaints about **FUNCTION**s and **SUBROUTINE**s that are defined but not used, or used but not defined. This option is useful when running `lintfor` on a subset of files of a larger program.

`-V` Suppress complaints about uninitialized variables. Detection of uninitialized variables takes considerable time and space. This option is useful for "quick checks" of FORTRAN programs.

The following arguments alter `lintfor`'s behavior:

`-l`*x*	Add a predefined `lintfor` library `fllib-l`*x*`.ln`. For example, to include a version of the `lintfor` math library `fllib-lm.ln`, specify `-lm` on the command line.
`-f` *flistfile*	Check the files listed in the file named *flistfile*. The file names in *flistfile* must be specified one per line, and can be either `.f` or `.ln` files.
`-c`	Cause `lintfor` to produce a `.ln` file for every `.f` file on the command line. These `.ln` files are the product of `lintfor`'s first pass only, and are not checked for inter-procedural compatibility or uninitialized variables.
`-o` *lib*	Cause `lintfor` to produce a `lintfor` library named `fllib-l`*lib*`.ln`. The `-c` option nullifies any use of the `-o` option. The `lintfor` library produced is the input that would be given to `lintfor`'s second pass. The `-o` option simply causes this file to be saved in the named `lintfor` library. In order to reduce the size of the `lintfor` libraries, the data required for uninitialized variable detection is not written if the `-o` option is given.

The `f77` `-a`, `-A`, `-C`, `-D`, `-g`, `-G`, `-I`, `-K`, `-L`, `-n`, `-N`, `-O`, `-q`, `-Q`, `-R`, `-s`, `-S`, `-v`, `-w`, `-W`, and all *+opt* options are also recognized as separate arguments; however, the `f77` `-C`, `-g`, `-G`, `-K`, `-n`, `-N`, `-O`, `+O`, `-q`, `-Q`, and `-s` options are ignored. This makes `lintfor`'s option list similar to that of the `f77` command, facilitating the use of `lintfor` instead of `f77` in makefiles (see *f77*(1)).

`lintfor` is a two-pass process. The first pass operates on a per-source-file basis. Syntax and other compiler errors are produced at this point. If all source files are processed without error and the `-c` and `-o` options are not specified, `lintfor` starts the second pass. It prints the message `Beginning inter-procedural checking` and performs the desired inter-procedural checks. The behavior of the `-c` and `-o` options allows for incremental use of `lintfor` on a set of FORTRAN source files.

Generally, `lintfor` is invoked once for each source file with the `-c` option. Each invocation produces a `.ln` file corresponding to the `.f` file, and prints diagnostics for that source file. After all source files have been run separately through `lintfor`, `lintfor` should be invoked again without the `-c` option, listing all of the `.ln` files to detect all of the inter-procedural inconsistencies. This scheme works well with `make`; it allows `make` to use `lintfor` on only those source files that have been modified since the set of source files were last run through `lintfor` (see *make*(1)).

WARNINGS

The algorithm that `lintfor` uses to detect uninitialized variables does not recognize `EQUIVALENCE` statements, nor can it follow assigned `GOTO` statements; therefore, `lintfor` might occasionally make the incorrect claim that a variable is uninitialized. `lintfor` also does not attempt to decipher variable `FORMAT` expressions; uninitialized variables used in variable `FORMAT` expressions are not detected.

DEPENDENCIES

The `TMPDIR` environment variable specifies a directory for temporary files to be used instead of the default directories `/tmp` and `/usr/tmp`.

FILES

`/usr/lib/fllib-l`*x*`.ln`	`lintfor` libraries that may be specified by the `-l`*x* option
`/usr/lib/lintfor1`	First pass of `lintfor`
`/usr/lib/lintfor2`	Actual inter-procedural checking program
`/usr/tmp/*lint*`	Temporary files

SEE ALSO

f77(1), lint(1), make(1).

(Optional Lisp Software Required)

NAME

lisp - HP Common Lisp environment

SYNOPSIS

`lisp` [*options*]

DESCRIPTION

`lisp` is the HP Common Lisp environment. It is provided in the following versions:

Series 300/400:	Lucid version 2.15
	Lucid version 4.0
Series 800:	Lucid version 3.0
	Lucid version 4.0

This manual entry focuses on HP Common Lisp version 4.0.

You can specify options when invoking the Lisp environment (see the section Options below). Once in the environment, you can compile functions, files, execute and debug programs, use the Lisp editor, have multiple processes running at the same time, use the windowing system, take advantage of the delivery tool kit to optimize Lisp programs, or use the the Common Lisp Object System (CLOS). Refer to the *HP Common Lisp User's Guide* for more information.

Invoking Lisp

To invoke Lisp, type the Lisp environment or image name on the keyboard and press Return. For example, to invoke the Lisp environment from a Lisp image called `mylisp`, use the command:

`mylisp` [Return]

Shell environment variables can also be used to invoke a Lisp environment. First, set and export the variable in your `.profile` file (Bourne, or Korn, or POSIX Shell) or set the variable in your `.login` file (C Shell). For example, if using Korn Shell, add the following to `.profile`:

```
LISP=/usr/lib/lisp/lisp-dev/lisp-de
export LISP
```

To invoke Lisp, type:

`$LISP` [Return]

Using this method, there is no need to update the environment variable until you move Lisp to a new directory or change the image name.

To initialize the Lisp environment when Lisp is invoked, create a file named `lisp-init.lisp` in your home directory. This initialization file can then be compiled for faster performance. `lisp-init` is loaded whenever Lisp is invoked, and all Lisp expressions in the file are evaluated.

Another variable called `*enter-top-level-hook*` can be set to control the behavior of Lisp when Lisp is started. If the variable is bound to a compiled function or a symbol that names a compiled function, that function is called before Lisp enters the top level.

Options

The following command-line options are recognized when invoking the `lisp` environment.

`-load`|`-l` *filename*
: Load the specified file when starting Lisp.

`-no-init-file`|`-n`
: Do not load the initialization file. The initialization file is `init-lisp.lisp` or `init-lisp.hbin`.

`-eval`|`-e` *form*
: Evaluate *form*. The return value is lost. Produces side-effects only.

`-quit`|`-q`
: Terminate the Lisp environment. To terminate Lisp from the top level of the Lisp environment, type `(quit)`.

Lisp Compilation and Execution

The options listed under the commands below all start with `:` The options listed are not discussed in detail in this document. For more information on the commands and their options, refer to the *HP Common Lisp*

User's Guide.

(`compile-file` *"path/filename"* [`:output-file`*"outputname"*]
[*compiler-options*
:messages :file-messages
:warnings :undef-warnings
:show-optimizations
:fast-entry :write-safety
:read-safety :tail-merge
:notinline :egc])

`Compile-file` compiles the Lisp source file specified by *path/filename*. If no path is specified, Lisp uses the value of `*default-pathname-defaults*` to look for a file. The compiled code name is placed in a file having the same name as the source, except it ends in `.hbin` or `.6bin`. If the `:output-file`*"outputname"* option is specified, the compiled code is named *outputname*.

Use compiler options to control the code optimization, increase the speed of compiled code, control the size of compiled code, suppress warnings, convert tail-recursive calls to iterative constructions, and enable execution of code during ephemeral garbage collection. Enter options when calling `compile-file` or by calling `compiler-options` first followed by `compile-file`.

For example, the following commands (denoted in

`LITERAL` typeface) entered from the Lisp environment:

```
> (compiler-options :warnings nil
:show-optimizations t)
> (:WARNINGS NIL :SHOW-OPTIMIZATIONS T)
> (compile-file "dr.l" :output-file " mydr.hbin")
```

cause the `dr.l` Lisp source code to be compiled with compiler warning messages suppressed and code optimizations reported to the terminal. The compiled code will be written to a file called `mydr.hbin`.

(`load` *filename* [*:verbose :print :if-does-not-exist*
:if-source-only :if-source-newer
:ignore-binary-dependencies])

Read and evaluate all forms in *filename* (i.e. load binary code after it is compiled so that it is available for use).

filename can be a pathname, stream, string, or a symbol. Specify `load` options to:

- Compile a source file first and then load it if the binary file did not exist when calling `load`.
- Print the values of all expressions that are loaded.
- Specify that you want to load whichever is newer, the source or the binary code.
- Load a binary file that was compiled for a certain run-time feature onto a machine that does not have the correct architecture.
- Load a source file without getting prompted to compile it.

Communicating With HP-UX

(`run-program` *name* [*:input :output :error-output*
:wait :arguments
:if-input-does-not-exist
:if-output-exists
:if-error-output-exists])

`Run-program` runs HP-UX programs from the Lisp environment. *name* contains the full pathname to the program to be run. This form is used to run shell scripts, specify options to get the HP-UX process id of the program being run, get the error output from a program, get the program exit status, and pass arguments to a program.

EXAMPLES

The following invokes the Lisp environment, loads a file called `myfun`, evaluates it (a message will print), and terminates Lisp — all by specifying command-line options when invoking the Lisp environment.

(Optional Lisp Software Required)

```
$ lisp -load "myfun" -eval "(myfun)" -quit
> hi there
$
```

This example shows how to run the Korn shell from Lisp, execute the *ls* command, and get back into Lisp.

```
> (run-program "/bin/ksh")
$ ls *.l
dr.l
mylisp.l
nl.l
$ exit
NIL
NIL
0
NIL
>
```

WARNINGS

When running a large application, be sure enough swap space is available to execute the program. In some cases it may be necessary to ensure that Lisp is the only process running in order to prevent running out of space, causing Lisp to halt.

DEPENDENCIES

If using the Window Tool Kit, X11 is required.

Full support is now provided for international character sets, including double-byte character sets such as Kanji. A double-byte character is considered as one character object so that it is treated in the same manner as a single-byte character.

To use Kanji, the Japanese Host Input Conversion program (JHIC) which is in the Native Language Input/Output (NLIO) subsystem is needed to convert Roman characters to Katakana, Hiragana, or Kanji characters.

Kanji characters are displayed in the Window Tool Kit if your terminal supports Japanese characters. Kanji is not supported in the Editor.

HP-UX 7.0 is required. It is NOT supported on prior releases of HP-UX.

Series 300/400:
HP Common Lisp II 2.15 cannot be installed and run unless you have an ID-module, codeword certificate, and the necessary codeword.

Series 800:
A color monitor is required when using the color Window Tool Kit.

FILES

Files that end in `.l` or `.lisp` are understood to be Lisp source files, although it is not a requirement to end Lisp source files in `.l` or `.lisp`. When compiled, the resulting binary file ends with `.6bin` on Series 300/400 and `.hbin` on Series 800 systems.

Series 300/400:

file.lisp	Lisp source file
file.6bin	Lisp Binary file

Series 800:

file.lisp	Lisp source file
file.hbin	Lisp Binary file

AUTHOR

HP Common Lisp was developed by HP and Lucid, Incorporated.

SEE ALSO

HP Common Lisp User's Guide, HP Part No. 92640-90002
HP Common Lisp Advanced User's Guide, HP Part No. 92640-90007
HP Common Lisp Editor, HP Part No. 92640-90009

Common Lisp: The Language, by Guy Steele, Jr., HP Part No. 9320-6047

(Requires Optional LISP SoftBench Software)

NAME

lispbench - LISP SoftBench interface tool

SYNOPSIS

`lispbench`

DESCRIPTION

`lispbench` is a window-based set of tools for programming in Common LISP that is based on HP SoftBench and HP Encapsulator. For more information refer to the *HP LISP-SoftBench Interface User's Guide*.

USAGE

The main screen presents pull-down menu bars and a LISP buffer (listener). Additionally, `lispbench` contains tools that have their own pull-down menu bars and buffers.

The pull-down menu bars can be used to access all of `lispbench`'s features. Commands that are available from the pull-down menu bar are detailed in the Commands section below. All of these commands are also accessible through user-defined key bindings.

The LISP buffer area is an I/O buffer which can be used interactively to read, eval, or print forms through the LISP listener (a traditional LISP listener).

The `Build` tool and the Common LISP `Help` tool contain two buffers each. The lower buffers in both of these tools are windows to the LISP listener. The top buffer in the `Build` tool, `Build Edit Buffer`, is used for editing different build constructs, and the one in the Common LISP `Help` tool, Common LISP `List Buffer` furnishes a list of available commands in Common LISP.

`lispbench` should be invoked by using the SoftBench Tool Manager.

Starting From The Tool Manager

Select the `Tool:Start` function from the pull-down menu in the SoftBench Tool Manager. A window pops up showing a list of tools available from the SoftBench Tool Manager. A single click on `LISP` from the list selects `lispbench`. Press the `Start` button, or double-click on `LISP` to start `lispbench`.

COMMANDS

Main lispbench Interface Screen:

The main screen has several pull down menus. These are labeled `File`, `Actions`, `Utilities`, and `Help`. Commands can be issued from buttons in the pull-down Menu, or by using a key binding. These key bindings can be used from any of the tool windows as well.

File Pull-Down Menu:

`Edit...` Opens a new Edit window for editing Common LISP source files. The default editor and the currently supported one is *Gnu Emacs* . Default keyboard accelerator: **Meta-E.**

`Set Context...` Changes the context in which `lispbench` runs. The context consists of the name of the host system and the base directory where the user's files reside. Default keyboard accelerator: **Meta-C.**

`Load File...` Loads a Common LISP file into `lispbench`. Default keyboard accelerator: **Ctrl-Meta-L.**

`Compile File...`
Compiles a Common LISP file and creates a binary file containing native machine code. Default keyboard accelerator: **Ctrl-Meta-C.**

`Write Buffer...`
Writes the entire contents of one or more of `lispbench`'s I/O buffers to a file. Default keyboard accelerator: **Meta-W.**

`Quit LISPBench` Quits `lispbench`. Default keyboard accelerator: **Meta-Q.**

Actions Pull-Down Menu:

`Evaluate...` Evaluates a Common LISP form. Default keyboard accelerator: **Ctrl-Meta-X** or use the `Enter` buttons.

`MacroExpand...` Expands a Common LISP macro before evaluating it. Default keyboard accelerator: **Ctrl-Meta-M.**

`PrettyPrint...` Displays a formatted version of a Common LISP form. Default keyboard accelerator: **Ctrl-Meta-P**.

`Describe...` Displays information about a Common LISP object. Default keyboard accelerator: **Ctrl-Meta-D**.

`SourceCode...` Displays the source code of an interpreted function in HP Common LISP. Default keyboard accelerator: **Ctrl-Meta-S**.

`ArgList...` Displays the arguments to a function in HP Common LISP. Default keyboard accelerator: **Ctrl-Meta-A**.

`Disassemble...` Disassembles a compiled function and displays the assembly code for that function in HP Common LISP. Default keyboard accelerator: **Ctrl-Meta-I**.

`Show Debugger Commands`
Displays the window based Debugger Commands List. Default keyboard accelerator: **Ctrl-Meta-E**.

`Hide Debugger Commands`
Hides the window based Debugger Commands List. Default keyboard accelerator: **Ctrl-Meta-H**.

Utilities Pull-Down Menu:

`Trace...` Displays the tracer options and invoke the trace tool. Default keyboard accelerator: **Ctrl-Shft-R**.

`Step...` Displays the stepper options and invoke the step tool. Default keyboard accelerator: **Ctrl-Shft-S**.

`Inspect...` Display the inspector options and invoke the inspect tool. Default keyboard accelerator: **Ctrl-Shft-E**.

`Static Analyzer...`
Displays the Static Analyzer Tool. Default keyboard accelerator: **Ctrl-Shft-X**.

`Storage Management...`
Displays HP Common LISP's storage management options. Default keyboard accelerator: **Ctrl-Shft-M**.

`Build...` Displays the `Build` tool. Default keyboard accelerator: **Ctrl-Shft-B**.

`Common LISP Help`
Provides information about Common LISP functionality. Default keyboard accelerator: **Ctrl-Shft-L**.

Help Pull-Down Menu:

`Item Help` Provides help information for any item on the screen. The cursor changes to a question mark. Clicking on an item will retrieve help for that item. Default keyboard accelerator: **Ctrl-Shft-H**.

`Application Help`
Provides general help information for the `lispbench` tool. Once Help is activated, information can be obtained for any SoftBench tool. Default keyboard accelerator: **Ctrl-Shft-A**.

Build Tool screen:

The `Build` tool has two pull-down menus. These are labeled `Tool` and `Help`. There are also 6 command buttons for editing, defining, and building a system. After a command button is pressed and the necessary input is provided, the `OK` button should be pressed to execute the command. To disregard the command, use the `Cancel` button.

Tool Pull-Down Menu:

`Quit` Hide the `Build` Window. Default keyboard accelerator: **Meta-S**.

Help Pull-Down Menu:

`Item Help` Provides help information for any item on the screen. The cursor changes to a question mark. Clicking on an item retrieves help for that item. Default keyboard accelerator: **Ctrl-Shft-H**.

`Application Help`
: Provides general help information for the `lispbench` tool. Once Help is activated, information can be obtained for any SoftBench tool. Default keyboard accelerator: **Ctrl-Shft-A**.

Pushbutton Area

`Edit System`	Creates or modifies a system.
`Delete System`	Deletes an existing system.
`Edit Module`	Creates or modifies a module for an existing system.
`Delete Module`	Deletes an existing module.
`Define System`	Defines a new system.
`Build`	Builds new or existing systems.

Common LISP Help Tool Screen:

The Common LISP `Help` tool has 3 pull-down menus. These are labeled `Tool`, `Actions`, and `Help`.

Tool Pull-Down Menu:

`Quit` Hide the Common LISP `Help` Window. Default keyboard accelerator: **Meta-S**.

Actions Pull-Down Menu:

`View...` Display information about a selected command in the Common LISP Help buffer. Default keyboard accelerator: **Ctrl-Shft-V**.

`Go To Command...`
: Prompts the user for a command name and displays information about that command. Default keyboard accelerator: **Ctrl-Shft-G**.

`Write Help Text...`
: Copy the information about a particular command to a file. Default keyboard accelerator: **Ctrl-Shft-W**.

Help Pull-Down Menu:

`Item Help` Provides help information for any item on the screen. The cursor changes to a question mark. Clicking on an item retrieves help for that item. Default keyboard accelerator: **Ctrl-Shft-H**.

`Application Help`
: Provides general help information for the `lispbench` tool. Once Help is activated, information can be obtained for any SoftBench tool. Default keyboard accelerator: **Ctrl-Shft-A**.

Debug Tools Tool Screen:

The Debug Tools tool has two pull-down menus. These are labeled `Tool` and `Help`. There are also three command buttons which provide access to the tracer, stepper, or the inspector.

Tool Pull-Down Menu:

`Quit` Hide the `Debug Tools` Window. Default keyboard accelerator: **Meta-S**.

Help Pull-Down Menu:

`Item Help` Provides help information for any item on the screen. The cursor changes to a question mark. Clicking on an item retrieves help for that item. Default keyboard accelerator: **Ctrl-Shft-H**.

`Application Help`
: Provides general help information for the `lispbench` tool. Once Help is activated, information can be obtained for any SoftBench tool. Default keyboard accelerator: **Ctrl-Shft-A**.

Pushbuttons Area

`Trace...` Displays the tracer options and invoke the trace tool.

`Step...` Displays the stepper options and invoke the step tool.

`Inspect...` Displays the inspector options and invoke the inspect tool.

Static Analyzer Tool screen:

The Static Analyzer tool has two pull-down menus. These are labeled `Tool` and `Help`. There are also six toggle buttons which provide different classes of functionality in the static analyzer.

Tool Pull-Down Menu:

`Quit` Hide the `Static Analyzer` Window. Default keyboard accelerator: **Meta-S**.

Help Pull-Down Menu:

`Item Help` Provides help information for any item on the screen. The cursor changes to a question mark. Clicking on an item retrieves help for that item. Default keyboard accelerator: **Ctrl-Shft-H**.

`Application Help`
Provides general help information for the `lispbench` tool. Once Help is activated, information can be obtained for any SoftBench tool. Default keyboard accelerator: **Ctrl-Shft-A**.

Toggle Buttons Area

`Analysis` Provides functions to control what is analyzed.

`Macro Settings` Provides functions to change how analysis is done with respect to macros.

`Database Queries`
Displays a variety of information about a Common LISP program.

`Display` Displays information about Common LISP objects and files.

`Structure Queries`
Provides information about the structures defined in a Common LISP program.

`Variable Usage Queries`
Provides information about the binding, referencing, and setting of variables.

Storage Management Tool Screen:

The Storage Management tool has two pull-down menus. These are labeled `Tool` and `Help`. There are also five toggle buttons which provide different classes of functionality in the storage manager.

Tool Pull-Down Menu:

`Quit` Hide the `Storage Manager` Window. Default keyboard accelerator: **Meta-S**.

Help Pull-Down Menu:

`Item Help` Provides help information for any item on the screen. The cursor changes to a question mark. Clicking on an item retrieves help for that item. Default keyboard accelerator: **Ctrl-Shft-H**.

`Application Help`
Provides general help information for the `lispbench` tool. Once in Help, information may be obtained for any SoftBench tool. Default keyboard accelerator: **Ctrl-Shft-A**.

The Toggle Buttons Area

`Report` Provides information on the current storage management settings.

`Memory Mgmt Settings`
Provide functions to configure the current memory allocations.

`Dynamic Gc` Provides Dynamic garbage collection functions.

`Egc` Provides Ephemeral garbage collection functions.

`Consing Space` Provides functions to control where new objects are created.

MESSAGES

`lispbench` supports the following message (tool class `LISP`) for use with Emacs. This message sends an evaluation request to Emacs to read in the contents of a file (the first data field of the message) and evaluate the entire file. Using this message, `lispbench` can easily be expanded to support more requests from

Emacs.

```
    Request EDIT EVAL-EXP
```

FILES

`/usr/softbench/bin/lispbench`
shell script to start *LISP SoftBench Interface*

`/usr/lib/X11/app-defaults/Lisp`
X11 application resource file

NOTES

Set the environment variable **CL** to the desired version of *Common LISP* you want to use and use **CLEXT** to indicate the type of LISP file extensions your *Common LISP* uses.

AUTHOR

lispbench was developed by HP.

SEE ALSO

encapsulate(1), softbench(1), softbench(5).

NAME

ln - link files and directories

SYNOPSIS

`ln` [`-f`] [`-i`] [`-s`] *file1 new_file*
`ln` [`-f`] [`-i`] [`-s`] *file1* [*file2* ...] *dest_directory*
`ln` [`-f`] [`-i`] [`-s`] *directory1* [*directory2* ...] *dest_directory*

DESCRIPTION

`ln` links:

- *file1* to a new or existing *new_file*,
- *file1* to a new or existing file named *file1* in existing *dest_directory*,
- *file1*, *file2*, ... to new or existing files of the same name in existing *dest_directory*,
- *directory1*, *directory2*, ... to new directories of the same name in existing *dest_directory*,
- or creates symbolic links between files or between directories.

If links are to *dest_directory*, corresponding file or directory names in that directory are linked to *file1*, *file2*, ..., or *directory1*, *directory2*, ..., etc., as appropriate. If two or more existing files or directories (excluding destination file name *new_file*) are specified, the destination must be a directory. If *new_file* already exists as a regular file (or link to another file), its contents (or the existing link) and its ACL are destroyed only if the `-f` option is specified. The ACL on the new_file after the link is the same as that of the source_file file.

If the `-f` and `-i` options are specified and the link being created is the name of an existing link or ordinary file and the access permissions of the file forbid writing, `ln` asks permission to overwrite the file. If the access permissions of the directory forbid writing, `ln` aborts and returns with the error message `cannot unlink` *new_file* (even if the file is an ordinary file and not a link to another file). When asking for permission to overwrite an existing file or link, `ln` prints the mode (see *chmod*(2) and Access Control Lists below), followed by the first letters of the words `yes` and `no` in the current native language, prompting for a response, and reading one line from the standard input. If the response is affirmative and is permissible, the operation occurs; if not, the command proceeds to the next source file, if any.

Hard links and symbolic links are created with the same ownerships and permissions as the file or directory to which they are linked. If ownership or permissions are changed on a link, file, or directory, the same changes appear on corresponding hard and symbolic links.

`ln` does not permit hard links to a directory.

If *file1* is a file and *new_file* is a link to an existing file or an existing file with other links, *new_file* is disassociated from the existing file and links and linked to *file1*. When `ln` creates a link to a new or existing filename, ownerships and permissions are always identical to those for the file to which it is linked. If `chown`, `chgrp`, or `chmod` is used to change ownership or permissions of a file or link, the change applies to the file and all associated links. The last modification time and last access time of the file and all associated links are identical (see *chown*(1), *chgrp*(1), and *chmod*(1)).

For a discussion of symbolic links, see *symlink*(4).

Options

`ln` recognizes the following options:

- `-f` Force existing *destination* pathnames to be removed to allow the link.
- `-i` Write a prompt to the standard error output requesting confirmation for each link that would overwrite an existing file. This option takes effect only if used in conjunction with the `-f` option.
- `-s` Cause `ln` to create symbolic links instead of the usual hard links. A symbolic link contains the name of the file to which it is linked (see WARNINGS below). The referenced file is used when an `open()` operation is performed on the link (see *open*(2)). A `stat()` on a symbolic link returns the linked-to file; an `lstat()` must be performed to obtain information about the link (see *stat*(2)). A `readlink()` call can be used to read the contents of the symbolic link (see *readlink*(2)). Symbolic links can span file systems and can refer to directories.

Access Control Lists (ACLs)

If optional ACL entries are associated with *new_file*, `ln` displays a plus sign (+) after the access mode when asking permission to overwrite the file.

If *new_file* is a new file, it inherits the access control list of *file1*, altered to reflect any difference in ownership between the two files (see *acl*(5)).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG and **LC_CTYPE** determine the local language equivalent of y (for yes/no queries).

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `ln` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

The following command creates filenames `new_file1` and `new_file2` in `dest_dir` which are linked back to the original files `file1` and `file2`:

```
ln file1 file2 dest_dir
```

If `new_file1` and/or `new_file2` exists, it is removed and replaced by a link to `file1` or `file2`, respectively. If existing `new_file1` or `new_file2` is a link to another file or a file with links, the existing file remains. Only the link to `new_file` is broken and replaced by a new link to the `file1` or `file2`.

WARNINGS

`ln` does not create hard links across file systems.

Use care in defining symbolic links. Unless *dest_directory* is the current working directory or the existing file is given in absolute path name form, it is very easy to define an endless loop, as in the case of:

```
cd /users/my_dir
ln -s file1 /dest_directory
```

which links the new file to itself instead of the intended `/users/my_dir/file1`.

DEPENDENCIES

NFS Access control lists of networked files are summarized (as returned in `st_mode` by `stat()`), but not copied to the new file. When using `ln` on such files, a + is not printed after the mode value when asking for permission to overwrite a file.

AUTHOR

`ln` was developed by AT&T, the University of California, Berkeley and HP.

SEE ALSO

cp(1), cpio(1), mv(1), rm(1), link(1M), readlink(2), stat(2), symlink(2), symlink(4), acl(5).

STANDARDS CONFORMANCE

`ln`: SVID2, XPG2, XPG3, POSIX.2

NAME

locale - get locale-specific (NLS) information

SYNOPSIS

`locale` [`-a`|`-m`]
`locale` [`-ck`] *name* ...

DESCRIPTION

`locale` displays information about the current locale or about available locales.

When invoked with no arguments, `locale` displays the name and actual or implied value of each of the locale-related environment variables: **LANG**, **LC_CTYPE**, **LC_COLLATE**, **LC_MONETARY**, **LC_NUMERIC**, **LC_TIME**, **LC_MESSAGES**, and **LC_ALL** in the order shown, one per line. An actual value is the value the variable actually has in the user's environment; an implied value is derived from the value of some other variable. Implied values are displayed enclosed in double quotes; actual values are unquoted.

The determination of implied values is that if the variable **LC_ALL** is present and has a non-null value, that is the actual value for **LC_ALL**, and all the other variables take its value as an implied value. If **LC_ALL** is not set, all the **LC_*** variables that are set are shown with their value as an actual value. Any that have no value are shown with the value of the **LANG** environment variable as their implied value. **LC_ALL** is displayed as **LC_ALL=\n** if it has no value.

`locale` can take multiple arguments which can be locale category names, locale keywords, or the special word `charmap` (see *localedef*(1M) for a description of locale keywords and charmaps). If an argument is a keyword, the value associated with that keyword in the current environment is displayed and possibly other information, depending on selected options. If an argument is a category name (i.e., **LC_***), the values of all keywords defined in that category are displayed. If an argument is the special word `charmap`, the charmap file (if any) that was used in the definition of the current locale is displayed.

Non-printable characters are printed as hexadecimal values in the form,

"\xhh"

except that, if a different escape character has been defined for the locale, it is displayed instead of the '\'.

Options

The following options are available:

`-a` List all available locales. These are the possible meaningful values that can be assigned to **LANG** or any of the **LC_*** variables on the system, and are dependent on what locales have been installed on the system. This option excludes all others.

`-c` Display names of locale categories that have been selected either explicitly or by giving a keyword contained therein. This option can be used with the `-k` option.

`-k` Display names of keywords that have been selected either explicitly or by their containing category having been given as an argument. Keyword names and values are displayed as:

"<keyword>=<value>"

Without the `-k` option, only the values are displayed. This option can be used with the `-c` option.

`-m` Display a list of available charmap files on the system. See *localedef*(1M) for a definition of charmap files and how they are used.

EXTERNAL INFLUENCES

Environment Variables

LANG Determines the locale to use for the locale categories when both **LC_ALL** and the corresponding category do not have values specified.

LC_ALL Overrides the values of **LANG** and all the **LC_*** variables in determining the locale.

LC_CTYPE
Determines the locale for interpretation of sequences of bytes as characters (e.g., single- versus multi-byte characters in arguments).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`locale` exits with one of the following values:

- **0** All requested information was found and displayed successfully.
- **>0** An error occurred either in finding the information or in displaying it.

EXAMPLES

If the locale environment variables are set as:

```
LANG=abc
LC_COLLATE=xyz
```

The command:

```
locale
```

gives the following output:

```
LANG=abc
LC_CTYPE="abc"
LC_COLLATE=xyz
LC_MONETARY="abc"
LC_NUMERIC="abc"
LC_TIME="abc"
LC_MESSAGES="abc"
LC_ALL=locale -ck decimal_point
```

The command:

```
LC_ALL=POSIX locale -ck decimal_point
```

produces:

```
LC_NUMERIC decimal_point="."
```

If **LANG** is set to `POSIX` and no other locale variables are set, the command:

```
locale LC_NUMERIC
```

would produce:

```
"."
","
0;3
""
```

which correspond to the keywords *decimal_point, thousands_sep, grouping* and *alt_digit.*

SEE ALSO

nlsinfo(1), localedef(1m), nl_langinfo(3c), localeconv(3c)

STANDARDS CONFORMANCE

`locale`: POSIX.2

NAME

lock - reserve a terminal

SYNOPSIS

`lock`

DESCRIPTION

`lock` requests a password from the user, then prints `LOCKED` on the terminal and refuses to relinquish the terminal until the password is repeated. If the user forgets the password, the only recourse is to log in elsewhere and kill the lock process.

NAME

logger - make entries in the system log

SYNOPSIS

`logger` [`-t` *tag*] [`-p` *pri*] [`-i`] [`-f` *file*] [*message* ...]

DESCRIPTION

`logger` provides a program interface to the `syslog()` system log module (see *syslog*(3C)).

A message can be given on the command line, which is logged immediately, or a file is read and each line is logged. If no *file* or *message* is specified, the contents of the standard input are logged.

Options

`logger` recognizes the following command-line options and arguments:

`-t` *tag*	Mark every line in the log with the specified *tag*. The default is the value returned by `getlogin()` (see *getlogin*(3C)). If `getlogin()` returns NULL, `syslog` is the default.
`-p` *pri*	Enter the message with the specified priority. The priority can be specified numerically or as a *facility*`.`*level* pair. For example, `-p local3.info` logs the message or messages as `info`rmational level in the `local3` facility. The default is `user.notice`.
`-i`	Log the process ID of the logger process with each line.
`-f` *file*	Log the contents of the specified file.
message	The message to log; if not specified, the file specified by the `-f` option or standard input is logged.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which diagnostic messages from `logger` to the user are displayed.

EXAMPLES

Send the message `System rebooted` to the `syslogd` daemon:

```
logger System rebooted
```

Send output from the `users` command (see *users*(1) to the `syslogd` daemon, marked as level `info` and facility `local0`. The message is tagged with the string `USERS`:

```
users | logger -p local0.info -t USERS
```

WARNINGS

`logger` has no effect if the `syslogd` daemon (see *syslogd*(1M)) is not running on the system.

AUTHOR

`logger` was developed by the University of California, Berkeley.

SEE ALSO

syslogd(1M), getlogin(3C), syslog(3C).

STANDARDS CONFORMANCE

`logger`: POSIX.2

NAME

login - sign on

SYNOPSIS

`login` [*name* [*env-var* ...]]

DESCRIPTION

`login` is used at the beginning of each terminal session to properly identify the prospective user. `login` can be invoked as a user command, or by the system when an incoming connection is first established. `login` is also invoked by the system when a previous user shell terminates but the terminal does not disconnect.

If `login` is invoked as a command it must replace the initial command interpreter (user's login shell). This is accomplished by typing:

```
exec login
```

from the user shell.

If *name* is not specified on the command line, `login` asks for `login name` and, if required, the corresponding password. Terminal echo is turned off (where possible) during typing of the password so that it will not appear on any written record of the session. If the login name provided is not valid, `login` requests a password. This is done to make it more difficult for an unauthorized user to log in on the system by trial and error. After three unsuccessful login attempts, a **HANGUP** signal is issued.

As a security precaution, some installations use an option that requires a second "dialup" password. This occurs only for dial-up connections, and is prompted by the message `dialup password:`. Both passwords must be correct for a successful login. See *dialups*(4) for details on dialup security.

If password aging has been invoked by the user with appropriate privileges on your behalf, your password may have expired. In this case, you will be diverted into `passwd` to change it, after which you can attempt to login again (see *passwd*(1)).

If login is not successfully completed within a certain period of time (e.g., one minute), the terminal is silently disconnected.

After a successful login, the accounting files are updated, user and group id's, group access list, and working directory are initialized, and the user's command interpreter, usually one of the shells listed in the *sh*(1) manual entry, is determined from corresponding user entries in files `/etc/passwd` and `/etc/logingroup` (if `/etc/passwd` does not specify a shell for *user*, `/bin/sh` is used by default). `login` then forks the appropriate shell by using the last component of the shell pathname preceded by a `-` (for example, `-sh` or `-ksh`). When the command interpreter is invoked with its name preceded by a minus in this manner, the shell performs its own initialization, including execution of profile, login, or other initialization scripts.

For example, if the user login shell is *sh*(1) or *ksh*(1), the shell executes the profile files `/etc/profile` and `$HOME/.profile` if they exist (and possibly others as well, depending on what they contain). Depending on what these profile files contain, messages regarding mail in your mail file or any messages you may have received since your last login may be displayed.

If the command name field is `*`, a `chroot()` to the directory named in the directory field of the entry is performed. At that point `login` is re-executed at the new level which must have its own root structure, including `/bin/login` and `/etc/passwd`.

The basic *environment* (see *environ*(5)) is initialized to:

> **HOME**=*your_login_directory*
> **PATH**=`:/bin:/usr/bin`
> **SHELL**=*last_field_of_passwd_entry*
> **MAIL**=`/usr/mail/`*your_login_name*
> **TZ**=*timezone_specification*

For users with appropriate privileges, **PATH** is augmented to include `/etc`. In the case of a remote login, the enviroment variable **TERM** is also set to the remote user's terminal type.

The environment can be expanded or modified by supplying additional arguments to `login`, either at execution time or when `login` requests your login name. The arguments can take either the form *xxx* or *xxx*=*yyy*. Arguments without an equal sign are placed in the environment as

L*n*=*xxx*

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If the name already appears in the environment, the new value replaces the older one. There are two exceptions. The variables `PATH` and `SHELL` cannot be changed. This prevents users logged in with restricted shell environments from spawning secondary shells that are not restricted. Both `login` and `getty` understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

If `/etc/btmp` is present, all unsuccessful login attempts are logged to this file. This feature is disabled if the file is not present. A summary of bad login attempts can be viewed by users with appropriate privileges by using `lastb`, see *last*(1M).

If `/etc/securetty` is present, login security is in effect, meaning that only users with appropriate privileges are allowed to log in successfully on the ttys listed in this file. Restricted ttys are listed by device name, one per line. Valid tty names are dependent on installation. Some examples could be `console`, `tty01`, `ttya1`, etc. Note that this feature does not inhibit a normal user from using `su`.

DIAGNOSTICS

The following diagnostics appear if the associated condition occurs:

`Login incorrect`
: User name or password cannot be matched.

`No shell`
`Cannot open password file`
`No directory`
: Consult system administrator.

`Your password has expired.  Choose a new one`
: Password aging is enabled and the user's password has expired.

`No Root Directory`
: Attempted to log into a subdirectory that does not exist (i.e., passwd file entry had shell name *, but the system cannot *chroot* to the given directory).

`No /bin/login or /etc/login on root`
: Same as above except sub-root login command not found.

`Bad user id` or `Bad group id.`
: *setuid* or *setgid* failed.

`Unable to change to directory` *name*
: Cannot `chdir` to your home directory.

`No shell` User shell (or `/bin/sh` if shell name is null in `/etc/passwd`) could not be *exec*'d.

`Sorry, single-user`
: Occurs if the version field from *uname*(2) starts with `A` (or if the *uname* system call fails) and if your terminal name is not `/dev/console` and if your home shell is not named `/usr/lib/uucp/uucico`. You are not logged in.

`No utmp entry. You must exec "login" from the lowest level "sh"`
: Attempted to execute `login` as a command without using the shell's `exec` internal command or from other than the initial shell.

`.rhosts is a soft link`
: Personal equivalence file is a symbolic link.

`Bad .rhosts ownership`
: Personal equivalence file is not owned by the local user or by the the user with appropriate privileges.

`Remuser too long`

`Locuser too long`
`Terminal type too long`
Indicated string was too long for `login's` internal buffer.

WARNINGS

If `/etc/group` is linked to `/etc/logingroup`, and group membership for the user trying to log in is managed by the Network Information Service (NIS), and no NIS server is able to respond, `login` waits until a server does respond.

AUTHOR

`login` was developed by AT&T and HP.

FILES

`$HOME/.profile`	personal profile (individual user initialization)
`$HOME/.rhosts`	personal equivalence file for the remote login server
`/etc/btmp`	history of bad login attempts
`/etc/d_passwd`	dialup security encrypted passwords
`/etc/dialups`	lines which require dialup security
`/etc/hosts.equiv`	system list of equivalent hosts allowing logins without passwords
`/etc/logingroup`	group file - defines group access lists
`/etc/motd`	message-of-the-day
`/etc/passwd`	password file - defines users, passwords, and primary groups
`/etc/profile`	system profile (initialization for all users)
`/etc/securetty`	list of valid ttys for root login
`/etc/utmp`	users currently logged in
`/etc/wtmp`	history of logins, logouts, and date changes
`/usr/mail/`*your_name*	mailbox for user *your_name*

VARIABLES

HOME	user's home directory.
PATH	path to be searched for commands.
SHELL	which command interpreter is being used.
MAIL	where to look for mail.
TERM	user's terminal type.
TZ	current timezone.
xxx	User-specified named variables.
L*xxx*	User-specified unnamed variables.

SEE ALSO

mail(1), newgrp(1), passwd(1), sh(1), su(1), getty(1M), last(1M), initgroups(3C), dialups(4), group(4), passwd(4), profile(4), utmp(4), environ(5), privilege(5).

NAME
logname - get login name

SYNOPSIS
`logname`

DESCRIPTION
`logname` writes the user's login name to standard output. The login name is equivalent to that returned by `getlogin()` (see *getlogin*(2)).

EXTERNAL INFLUENCES
Environment Variables
LANG determines the language in which diagnostic messages are displayed.

FILES
`/etc/profile`

SEE ALSO
env(1), login(1), getlogin(3C), logname(3C), environ(5).

STANDARDS CONFORMANCE
`logname`: SVID2, XPG2, XPG3, POSIX.2

NAME

lorder - find ordering relation for an object library

SYNOPSIS

`lorder` [*files*]

DESCRIPTION

The input consists of one or more object or library archive *files* (see *ar*(1)) placed on the command line or read from standard input. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. Output can be processed by `tsort` to find an ordering of a library suitable for one-pass access by `ld` (see *tsort*(1) and *ld*(1)) Note that the link editor `ld` is capable of multiple passes over an archive in the archive format and does not require that `lorder` be used when building an archive. Using the `lorder` command may, however, allow for a slightly more efficient access of the archive during the link edit process.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Build a new library from existing `.o` files:

```
ar cr library 'lorder (**.o | tsort'
```

When creating libraries with so many objects that the shell cannot properly handle the `*.o` expansion, the following technique may prove useful:

```
ls | grep '.o$' | lorder | tsort | xargs ar cq library
```

WARNINGS

Object files whose names do not end with `.o` are overlooked, even when contained in library archives. Their global symbols and references are attributed to some other file.

DEPENDENCIES

Series 700/800:

The symbol table maintained by the Series 700/800 version of `ar` allows `ld` to randomly access symbols and files in the archive, making the use of `lorder` unnecessary when building archive libraries (see *ar(1)).*

FILES

`/tmp/*symref,` `/tmp/*symdef`	temporary files

SEE ALSO

ar(1), ld(1), tsort(1).

STANDARDS CONFORMANCE

`lorder`: SVID2, XPG2

NAME

lp, cancel, lpalt - send/cancel/alter requests to an LP line printer or plotter

SYNOPSIS

`lp` [`-c`] [`-d`*dest*] [`-m`] [`-n`*number*] [`-o`*option*] [`-p`*priority*] [`-s`] [`-t`*title*] [`-w`] [*files* ...]

`cancel` [*ids*] [*printers*] [`-a`] [`-e`] [`-i`] [`-u`*user*]

`lpalt` *id* [`-d`*dest*] [`-i`] [`-m`] [`-n`*number*] [`-o`*option*] [`-p`*priority*] [`-s`] [`-t`*title*] [`-w`]

DESCRIPTION

`lp` arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer or supported plotter. If no file names are specified, standard input is assumed. The file name - specifies standard input and can be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed or plotted.

`lp` associates a unique ID with each request and prints it on the standard output. This ID can be used later to cancel (see `cancel`), alter (see `lpalt`), or find the status (see *lpstat*(1)) of the request.

`lp` recognizes the following options which can be specified in any order prior to the file names. Blanks are not currently allowed between an option and its corresponding argument.

`-c` Make copies of the *files* to be printed immediately when `lp` is invoked. Normally, *files* are linked into a spool directory. Ownership and mode of the linked *files* remains unchanged. If the `-c` option is given or linking is not possible the *files* are copied, in which case the ownership and mode are set to allow read access to owner `lp` and group *bin* only. It should be noted that if the *files* are linked rather than copied, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output. The standard input is always copied instead of linked.

`-d`*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, the request will be printed only on that specific printer. If *dest* is a class of printers, the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations might not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable `LPDEST` (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

`-m` Send mail (see *mail*(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

`-n`*number* Print *number* copies (default of 1) of the output.

`-o`*option* Specify printer-dependent or class-dependent *options*. Several such *options* can be collected by specifying the `-o` keyletter more than once. For more information about what *options* are valid for printers supported on your system, see the particular model script in `/usr/spool/lp/model` associated with the specified printer.

`-p`*priority* Give *priority* to the print request. This parameter is used to select next spooled file for the targeted printer or class of printers. *priority* must be in between 0 (lowest priority) and 7 (highest priority). If the value is less than *fence* (see *lpfence*(1M)), the print request is deferred. Default is the default *priority* set by `lpadmin` (see *lpadmin*(1M)) of the printer when printer is specified for *dest*, and is the highest default *priority* among printers of the class when class is specified for *dest*.

`-s` Suppress messages from `lp` such as "request ID is ...".

`-t`*title* Print *title* on the banner page of the output.

`-w` Write a message on the user's terminal after the *files* have been printed. If the user is not logged in or `rlpdaemon` is not running on local system for remote printing *rlpdaemon*(1M)), mail will be sent instead.

`cancel` cancels line printer requests that were made by the `lp` command. At least one *id* or *printer* must be specified.

ids	Specify a request *id* (as returned by `lp`) for a local printer. This cancels the associated request even if it is currently printing. Specifying a request *id* (as returned by `lp`) for a remote printer cancels the associated request if it is owned by the user, even if it is currently printing.
printers	Specify the names of *printers* (for a complete list, use **`lpstat`**). Specifying a local printer cancels the request currently printing on that printer. Specifying a remote printer cancels the request currently printing on that printer if it is owned by the user. If the **`-a`**, **`-e`**, or **`-u`** option is specified, this option specifies the printer on which to perform the cancel operation.
`-a`	Remove all requests a user owns on the specified printer (see *printers*). The owner is determined by the user's login name and host name on the machine where the `lp` command was invoked.
`-e`	Empty the spool queue of all requests for the specified printer (see *printers*). Only users with appropriate privileges can use the **`-e`** option.
`-i`	Cancel only local requests.
`-u`*user*	Remove any requests queued belonging to user. Multiple **`-u`** options are allowed. Only users with appropriate privileges can use the **`-u`** option.

In any case, the cancellation of a request that is currently printing frees the printer to print its next available request.

`lpalt` alters a line printer request made by a previous `lp` command. A new unique ID is returned to the standard output.

id	Specify a request *id* (as returned by `lp`) for a local printer. This alters the associated request only if it is not currently printing. Specifying a request *id* (as returned by `lp`) for a remote printer alters the associated request if it is owned by the user, only if it is not currently printing.
`-i`	Alter only local requests.

HP Clustered Environment

In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, **`lp`** and **`cancel`** behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occured.

EXAMPLES

Assuming there is an existing HP2934A line printer named **`lp2`**, configured with the **`hp2934a`** model interface program. This model has the **`-c`** option which causes the printer to print in a compressed mode. To obtain compressed print on **`lp2`**, use the command:

```
lp -dlp2 -oc files
```

WARNINGS

A remote print request can be cancelled only by the user who requested it, and only on the system from which the request was spooled.

If the restrict cancel feature (selected by `lpadmin -orc` (see *lpadmin*(1M)) is enabled for the specified printer, a user can only cancel/alter requests owned by the user.

For remote system, `lpalt` cannot change *dest* and *priority*.

FILES

`/usr/spool/lp/*`

SEE ALSO

enable(1), lpstat(1), mail(1), slp(1), accept(1M), lpadmin(1M), lpana(1M), lpsched(1M), mklp(1M), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M).

STANDARDS CONFORMANCE

`lp`: SVID2, XPG2, XPG3, POSIX.2

`cancel`: SVID2

NAME

lpfilter, divpage, fontdl, lprpp, plotdvr, printstat, reverse - filters invoked by lp interface scripts

SYNOPSIS

/usr/lib/divpage [**-p** | **-l**] [**-h** | **-q**] [**-n***FontID*] *filename*

/usr/lib/fontdl [**-n***FontID*] [**-l**] [**-p**] *filename*

/usr/lib/lprpp [**-i**] [**-o**] [**-e**] [**-l***nn*] [**-n**] [**-p**]

/usr/lib/plotdvr -l *request_id* **-u** *username* [**-e**] [**-f**] [**-i**] *filename*

/usr/lib/printstat -l *request_id* **-u** *username filename*

/usr/lib/reverse [**-l** *page_length*]

Remarks

The structure of these filters is currently under review. They may become obsolete or be restructured in a future release.

DESCRIPTION

Various filters are used by the **lp** subsystem to obtain specialized behavior for specific types of devices or data. This entry describes currently supported filters.

A number of these filters communicate with the user who originated the print request. They use the specified *username* and *filename* to determine the location of the user. *filename* is used to determine the *hostname* of the system where the request originated, and must have the form [*dirname*]**/d?A***nnnhostname* or [*dirname*]**/dA***nnnnhostname*, where *dirname* is not a pathname, but only the name of the basename's parent directory. *filename* meets this requirement when it is set to **$6** in the interface script for the printer.

divpage

Provides capabilities for printing multiple pages per sheet and selection of built-in fonts.

Options:

-p	Set mode to portrait (default).
-l	Set mode to landscape.
-h	Print half pages (default).
-q	Print quarter pages.
-n*FontID*	Use font number *FontID*. Default is 0. Causes the string $^{\text{E}}$**c(***FontID***X** to be sent to the printer.

fontdl

fontdl downloads the font contained in *filename* to a printer connected to standard output.

Options:

-n*FontID*	Specifies the ID number by which the font will be referenced. Default is 0.
-l	Specify landscape mode. Default is portrait.
-p	Specifies proportional spacing. Default is fixed.

lprpp

This is a filter that converts backspace overstrike to line overprint with horizontal print positioning to enhance bold print. This functionality is required on printers such as the LaserJet, which cannot produce bold print by overstriking.

Options:

-i	Converts <ANYCHAR> to <ANYCHAR><BACKSPACE>_ to italicize ANYCHAR. Also properly italicizes overstruck (bold) characters. Does *not* work correctly for "hashed-overstrike" such as <ANYCHAR><BACKSPACE><DIFFERENTCHAR><BACKSPACE>_.
-o	Print only the odd numbered pages. Used with **-e** for double-sided printing.

`-e`	Print only the even numbered pages. Used with `-o` for double sided printing.
`-l`*nn*	Specifies the page length in lines. Default is 60 unless *-n* or *-p* is selected, in which case it is 66.
`-n`	Specifies nroff mode for printing output of the *nroff* command. Prints 66 lines per page with the first line appearing on logical line 4 of the printer.
`-p`	Specifies pr mode for printing output from the *pr* command. Prints 66 lines per page with the first line appearing on logical line 3 of the printer.

plotdvr

HP-GL plotter filter. This filter scans the data for PG commands (paper feed). When this data is encountered, the filter strips it from the data stream and informs the requesting user of the need to change paper in the plotter.

Options:

`-l`*request_id*	Specifies the printer request ID and is used in various messages regarding the plot request.
`-u`*username*	The requesting user's login name, used to communicate with the user regarding the request.
`-e`	Specifies the use of escape sequences, rather than HP-GL commands, to determine plotter status.
`-f`	Plot without stopping for paper changes. The PG commands are not stripped from the data stream and the user is not notified of them. This option is used on plotters capable of automatic page feed.
`-i`	Prevents initialization of the plotter.

printstat

Interrogates an RS232 printer as to its status, and does not return until the printer is ready. If the printer is *off-line*, *out of paper*, or *disconnected*, the submitter of the print request is notified of this condition periodically until it is corrected. When the printer is ready to print, the command exits.

Standard input and standard output must both be connected to the serial printer device.

This program uses the *send-status* command ^{E}c?^{D}1? to determine status. Not all serial printers respond to this command. Only the following configurations support this command:

Printer	Comments
LaserJet	Not supported
LaserJetII	Supported
LaserJetIID	Requires HP26013A module
LaserJetIIP	Not supported
LaserJetIII	Requires HP26013A module
LaserJet2000	Not supported

Options:

`-l`*request-id*	Print request ID used in various communications with the user.
`-u`*username*	The requesting user's login name, used to communicate with the user regarding the request.

reverse

Prints the data appearing on the standard input in reverse page order to the standard output. This command can handle up to 2000 pages.

Options:

`-l`*page_length*	Specifies the page length, in lines. Default is 66.

DIAGNOSTICS

Error and diagnostic messages appear on the printed output, on the user's terminal, or are mailed to the user, depending on circumstances.

WARNINGS

There is little consistency in the interface to these filters.

SEE ALSO

lp(1), lpadmin(1).

HP-UX System Administration Manual.

NAME
lpstat - print LP status information

SYNOPSIS
`lpstat` [*options*] [*id* ...]

DESCRIPTION
`lpstat` prints information about the current status of the LP line printer system.

If no *options* are given, `lpstat` prints the status of all requests made to `lp` by the user (see *lp*(1)). Any arguments that are not *options* are assumed to be request *ids* (as returned by `lp`). `lpstat` prints the status of such requests. *options* can appear in any order and can be repeated and intermixed with other arguments. Some of the keyletters below can be followed by an optional *list* that can be in one of two forms:

- A list of items separated from one another by a comma, or
- A list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces as in:

  ```
  -u"user1, user2, user3"
  ```

Omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed. For example:

```
lpstat -o
```

prints the status of all output requests.

`-a`[*list*]	Print acceptance status (with respect to `lp`) of destinations for requests. *list* is a list of intermixed printer names and class names.
`-c`[*list*]	Print class names and their members. *list* is a list of class names.
`-d`	Print the system default destination for `lp`.
`-i`	Inhibit the reporting of remote status.
`-o`[*list*]	Print the status of output requests. *list* is a list of intermixed printer names, class names, and request *ids*. See the `-i` option.
`-p`[*list*]	Print the status of printers. *list* is a list of printer names.
`-r`	Print the status of the LP request scheduler
`-s`	Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.
`-t`	Print all status information. Same as specifying `-r`, `-s`, `-a`, `-p`, `-o`. See the `-i` option.
`-u`[*list*]	Print status of output requests for users. *list* is a list of login names.
`-v`[*list*]	Print the names of printers and the path names of the devices associated with them. *list* is a list of printer names.

HP Clustered Environment
In the HP Clustered Environment, all spooling is handled as if the cluster nodes were a single system and all printers attached to either the cluster server or clients can be available. Remote spooling applies to spooling from or to machines outside of the cluster nodes.

EXTERNAL INFLUENCES
Environment Variables
`LC_TIME` determines the format and contents of date and time strings.

`LANG` determines the language in which messages are displayed.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, *lpstat* behaves as if all internationalization variables are set to "C".

See *environ*(5).

EXAMPLES

Check whether your job is queued:

```
lpstat
```

Check the relative position of a queued job:

```
lpstat -t
```

Verify that the job scheduler is running:

```
lpstat -r
```

FILES

`/usr/spool/lp/*`

SEE ALSO

enable(1), lp(1), rlpstat(1M).

STANDARDS CONFORMANCE

`lpstat`: SVID2, XPG2, XPG3

NAME

ls, l, ll, lsf, lsr, lsx - list contents of directories

SYNOPSIS

ls [**-abcdfgilmnopqrstuxACFHLR1**] [*names*]
l [*ls_options*] [*names*]
ll [*ls_options*] [*names*]
lsf [*ls_options*] [*names*]
lsr [*ls_options*] [*names*]
lsx [*ls_options*] [*names*]

DESCRIPTION

For each directory argument, **ls** lists the contents of the directory. For each file argument, **ls** repeats its name and any other information requested. The output is sorted in ascending collation order by default (see Environment Variables below). When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

If you are the user with appropriate privileges, all files except **.** and **..** are listed by default.

There are three major listing formats. The format chosen depends on whether the output is going to a login device (determined by whether output device file is a *tty* device), and can also be controlled by option flags. The default format for a login device is to list the contents of directories in multi-column format, with entries sorted vertically by column. (When individual file names (as opposed to directory names) appear in the argument list, those file names are always sorted across the page rather than down the page in columns because individual file names can be arbitrarily long.) If the standard output is not a login device, the default format is to list one entry per line. The **-C** and **-x** options enable multi-column formats, and the **-m** option enables stream output format in which files are listed across the page, separated by commas. In order to determine output formats for the **-C**, **-x**, and **-m** options, **ls** uses an environment variable, **COLUMNS**, to determine the number of character positions available on each output line. If this variable is not set, the *terminfo* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns is assumed.

Options

ls recognizes numerous options:

-a List all entries; usually entries whose names begin with a period (**.**) are not listed.

-b Force printing of non-graphic characters to be in the octal *\ddd* notation.

-c Use time of last modification of the inode (file created, mode changed, etc.) for sorting (**-t**) or printing (**-l** (ell)).

-d If an argument is a directory, list only its name (not its contents); often used with **-l** (ell) to get the status of a directory.

-f Force each argument to be interpreted as a directory and list the name found in each slot. This option disables **-l** (ell), **-t**, **-s**, and **-r**, and enables **-a**; the order is the order in which entries appear in the directory.

-g Same as **-l**, (ell) except that only the group is printed (owner is omitted). If both **-l** (ell) and **-g** are specified, the owner is not printed.

-i For each file, print the inode number in the first column of the report. When used in multi-column output, the number precedes the file name in each column.

-l (ell) List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see further DESCRIPTION and Access Control Lists below). If the time of last modification is greater than six months ago, or any time in the future, the year is substituted for the hour and minute of the modification time. If the file is a special file, the size field contains the major and minor device numbers rather than a size.

-m Stream output format.

-n The same as **-l**, (ell) except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

`-o`	The same as `-l`, (ell) except that only the owner is printed (group is omitted). (If both `-l` (ell) and `-o` are specified, the group is not printed).
`-p`	Put a slash (`/`) after each file name if that file is a directory.
`-q`	Force printing of non-graphic characters in file names as the character (`?`).
`-r`	Reverse the order of sort to get reverse (descending) collation or oldest first, as appropriate.
`-s`	Give size in blocks, including indirect blocks, for each entry. The first entry printed is the total number of blocks in the directory. When used in multi-column output, the number of blocks precedes the file name in each column.
`-t`	Sort by time modified (latest first) before sorting alphabetically.
`-u`	Use time of last access instead of last modification for sorting (`-t` option) or printing (`-l` (ell) option).
`-x`	Multi-column output with entries sorted across rather than down the page.
`-A`	The same as `-a`, except that the current directory "." and parent directory ".." are not listed. For the user with appropriate privileges, this flag defaults to ON, and is turned off by `-A`.
`-C`	Multi-column output with entries sorted down the columns.
`-F`	Put a slash (`/`) after each file name if that file is a directory or a symbolic link to a directory; put an asterisk (`*`) after each file name if that file is executable; put an at sign (`@`) after each file name if that file is a symbolic link to a file; put a vertical bar (`|`) after each file name if that file is a FIFO.
`-H`	Put a plus sign (`+`) after each file name if that file is a hidden directory (context-dependent file). This option also implies `-F`. See *cdf*(4). When used with the `-l` (ell) option on device files, the cnode name (if found in `/etc/clusterconf`) or cnode ID of the device file is printed following the minor device number. A cnode ID of zero is used to indicate a device file that can be accessed from any cnode.
`-L`	If the argument is a symbolic link, list the file or directory to which the link refers rather than the link itself.
`-R`	Recursively list subdirectories encountered.
`-1`	(one) The file names will be listed in single column format regardless of the output device. This forces single column format to the user's terminal.

Specifying more than one of the options in the following mutually exclusive pairs is not considered an error: `-C` and `-l` (ell), `-m` and `-l` (ell), `-x` and `-l` (ell), `-C` and `-1` (one), `-c` and `-u`.

`ls` is normally known by several shorthand-version names for the various formats:

`l`	equivalent to `ls -m`.
`ll`	equivalent to `ls -l` (ell).
`lsf`	equivalent to `ls -F`.
`lsr`	equivalent to `ls -R`.
`lsx`	equivalent to `ls -x`.

The shorthand notations are implemented as links to `ls`. Option arguments to the shorthand versions behave exactly as if the long form above had been used with the additional arguments.

Mode Bits Interpretation (–l option)

The mode printed in listings produced by the `-l` (ell) option consists of 10 characters. The first character indicates the entry type:

`d`	Directory
`b`	Block special file
`c`	Character special file
`l`	Symbolic link
`H`	Hidden directory (context-dependent file). Hidden directories are only visible when the `-H` option has been specified;

`p`	Fifo (also called a "named pipe") special file
`n`	Network special file
`s`	socket
`-`	Ordinary file.

The next 9 characters are interpreted as three sets of three bits each which identify access permissions for owner, group, and others as follows:

```
┌──────────────── 0400  read by owner (r or -)
| ┌────────────── 0200  write by owner (w or -)
| | ┌──────────── 0100  execute (search directory) by owner (x, s, S, or -)
| | | ┌────────── 0040  read by group (w or -)
| | | | ┌──────── 0020  write by group (r or -)
| | | | | ┌────── 0010  execute/search by group (x, s, S, or -)
| | | | | | ┌──── 0004  read by others (r or -)
| | | | | | | ┌── 0002  write by others (w or -)
| | | | | | | | ┌ 0001  execute/search by others (x, t, T, or -)
| | | | | | | | |
r w x r w x r w x
```

Mode letters are interpreted as follows:

`-`	Permission *not* granted in corresponding position.
`r`	Read permission granted to corresponding user class.
`w`	Write permission granted to corresponding user class.
`x`	Execute (or search in directory) permission granted to corresponding user class.
`s`	Execute (search) permission granted to corresponding user class. In addition, SUID (Set User ID) permission granted for owner, or SGID (Set Group ID) permission granted for group, as indicated by position.
`S`	Same as `s` except that execute (search) permission is denied to corresponding user class.
`t`	(last position only) Execute (search) permission granted to others and "sticky bit" is set (see *chmod*(2) `S_ISVTX` description).
`T`	Same as `t` except execute (search directory) permission denied to others.

When an option is specified that results in a listing of directory and/or file sizes in bytes or blocks (such as the `-s` or `-l` (ell) option), a total count of blocks, including indirect blocks, is also printed at the beginning of the listing.

Access Control Lists (ACLs)

If a file has optional ACL entries, the `-l` (ell) option displays a plus sign (+) after the file's permissions. The permissions shown are a summary representation of the file's access control list, as returned by `stat()` in the *st_mode* field (see *stat*(2)). To list the contents of an access control list, use the `lsacl` command (see *lsacl*(1) and *acl*(5)).

EXTERNAL INFLUENCES

Environment Variables

If the `COLUMNS` variable is set, `ls` uses the width provided in determining positioning of columnar output.

`LC_COLLATE` determines the order in which the output is sorted.

`LC_CTYPE` determines which characters are classified as non-graphic for the `-b` and `-q` options, and the interpretation of single and/or multi-byte characters within file names.

`LC_TIME` determines the date and time strings output by the `-g`, `-l` (ell), `-n`, and `-o` options.

`LANG` determines the language in which messages (other than the date and time strings) are displayed.

If `LC_COLLATE`, `LC_CTYPE`, or `LC_TIME` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `ls` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`ls` exits with one of the following values:

0	All input files were listed successfully.
>0	`ls` was aborted because errors occurred when accessing files. The following conditions cause an error:

- Specified file not found.
- User has no permission to read the directory.
- Process could not get enough memory.
- Invalid option specified.

EXAMPLES

Print a long listing of all the files in the current working directory (including the file sizes). List the most recently modified (youngest) file first, followed by the next older file, and so forth, to the oldest. Files whose names begin with a `.` are also printed.

```
ls -alst
```

WARNINGS

Setting options based on whether the output is a login (*tty*) device is undesirable because `ls -s` is much different than `ls -s | lpr`. On the other hand, not using this setting makes old shell scripts that used `ls` almost inevitably fail.

Unprintable characters in file names may confuse the columnar output options.

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

DEPENDENCIES

NFS The `-l` (ell) option does not display a plus sign (+) after the access permission bits of networked files to represent existence of optional access control list entries.

AUTHOR

`ls` was developed by AT&T, the University of California, Berkeley and HP.

FILES

`/etc/passwd`	to get user IDs for `ls -l` (ell) and `ls -o`.
`/etc/group`	to get group IDs for `ls -l` (ell) and `ls -g`.
`/usr/lib/terminfo/?/*`	to get terminal information.

SEE ALSO

chmod(1), find(1), lsacl(1), stat(2), cdf(4), acl(5).

STANDARDS CONFORMANCE

`ls`: SVID2, XPG2, XPG3, POSIX.2

NAME

lsacl - list access control lists (ACLs) of files

SYNOPSIS

lsacl [-l] *file* ...

DESCRIPTION

lsacl lists access control lists (ACLs) of one or more files in symbolic, "short" form, one file's ACL per line of output, followed by the file name; see *acl*(5) for ACL syntax.

Options

lsacl recognizes the following option:

-l Print ACLs in long form. Each file's ACL can be more than one line long, and is always preceded by file name, colon, and newline. Consecutive file names are separated by blank lines.

If a hyphen (-) is given as a file name argument, **lsacl** prints the ACL for the standard input. By default, it prints the file name as **-**. For the **-l** option it prints a file name of **<stdin>**.

Unlike **ls**, **lsacl** cannot list ACLs of files in subdirectories automatically or list the ACL entries of the files in the current directory by default. A good way to do the latter is:

```
lsacl *
```

or

```
lsacl .* *
```

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **lsacl** behaves as if all internationalization variables are set to "C". See *environ*(5).

RETURN VALUE

If **lsacl** succeeds, it returns zero. If invoked incorrectly, it returns a value of 1. If **lsacl** is unable to read the ACL for a specified file, it prints an error message to standard error, continues, and later returns a value of 2.

EXAMPLES

List the ACL for the file **dir/file1**:

```
lsacl dir/file1
```

List ACLs for all files in the current directory, in long form.

```
lsacl -l .* *
```

List ACLs for all files under **mydir**:

```
find mydir -print | sort | xargs lsacl
```

DEPENDENCIES

NFS:

lsacl is not supported on remote files.

AUTHOR

lsacl was developed by HP.

SEE ALSO

chacl(1), getaccess(1), ls(1), getacl(2), acl(5).

NAME

m4 - macro processor

SYNOPSIS

m4 [*options*] [*file* ...]

DESCRIPTION

m4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

Options

m4 recognizes the following command-line options:

-e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode may be very difficult.

-s Enable line sync output for the C preprocessor (#line ...)

-B*int*
: Change the size of the push-back and argument collection buffers from the default of 4,096.

-H*int*
: Change the size of the symbol table hash array from the default of 199. The size should be prime.

-S*int*
: Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

-T*int*
: Change the size of the token buffer from the default of 512 bytes.

To be effective, the flags listed above must appear before any file names and before any -D or -U flags.

-D*name*[=*val*]
: Defines *name* to *val* or to null in *val*'s absence.

-U*name*
: undefines *name*.

Macro calls have the form:

name (*arg1* **,** *arg2* **,** ... **,** *argn*)

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

Built-In Macros

m4 makes available the following built-in macros. They can be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

changecom	Change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

`changequote`	Change quote symbols to the first and second arguments. The symbols may be up to five characters long. `changequote` without arguments restores the original values (i.e., ' ').
`decr`	Returns the value of its argument decremented by 1.
`define`	The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is equivalent to **$***, but each argument is quoted (with the current quotes).
`defn`	Returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
`divert`	**m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The `divert` macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
`divnum`	Returns the value of the current output stream.
`dnl`	Reads and discards characters up to and including the next new-line.
`dumpdef`	Prints current names and definitions, for the named items, or for all if no arguments are given.
`errprint`	Prints its argument on the diagnostic output file.
`eval`	Evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ** (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
`ifdef`	If the first argument is defined, the value is the second argument; otherwise the third. If there is no third argument, the value is null. The word `unix` is predefined on HP-UX system versions of **m4**.
`ifelse`	Has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
`include`	Returns the contents of the file named in the argument.
`incr`	Returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
`index`	Returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.
`len`	Returns the number of characters in its argument.
`m4exit`	Causes immediate exit from **m4**. Argument 1, if given, is the exit code; the default is 0.
`m4wrap`	Argument 1 is pushed back at final EOF; example: `m4wrap('cleanup()')`
`maketemp`	Fills in a string of **XXXXX** in its argument with the current process ID.
`popdef`	Removes current definition of its argument(s), exposing the previous one, if any.
`pushdef`	Similar to `define`, but saves any previous definition.
`shift`	Returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

`sinclude`	Identical to `include`, except that it says nothing if the file is inaccessible.
`substr`	Returns a substring of its first argument. The second argument is a zero-origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
`syscmd`	Executes the HP-UX system command given in the first argument. No value is returned.
`sysval`	Is the return code from the last call to *syscmd*.
`traceoff`	Turns off trace globally and for any macros specified. Macros specifically traced by `traceon` can be untraced only by specific calls to `traceoff`.
`traceon`	With no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
`translit`	Transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
`undefine`	Removes the definition of the macro named in its argument.
`undivert`	Causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

SEE ALSO

cc(1), cpp(1).

m4 Macro Processor tutorial in *Programming on HP-UX*.

STANDARDS CONFORMANCE

m4: SVID2, XPG2, XPG3

NAME

hp9000s200, hp9000s300, hp9000s500, hp9000s800, pdp11, u3b, u3b5, vax - provide truth value about your processor type

SYNOPSIS

```
hp9000s200
hp9000s300
hp9000s400
hp9000s500
hp9000s700
hp9000s800
hp-mc680x0
hp-pa
pdp11
u3b
u3b5
vax
```

DESCRIPTION

The following commands return a true value (exit code 0) if the a processor type matches the command name. Otherwise a false value (exit code non-zero) is returned. These commands are commonly used within `make` makefiles and shell procedures to improve portability of applications (see *make*(1)).

Command	True if Processor is:
`hp9000s200`	HP 9000 Series 200
`hp9000s300`	HP 9000 Series 300
`hp9000s400`	HP 9000 Series 400
`hp9000s500`	HP 9000 Series 500
`hp9000s700`	HP 9000 Series 700
`hp9000s800`	HP 9000 Series 800
`hp-mc680x0`	HP 9000 Series 200, 300, or 400
`hp-pa`	HP 9000 Series 700 or 800
`pdp11`	PDP-11/45 or PDP-11/70
`u3b`	3B 20S computer
`u3b5`	3B 5 computer
`vax`	VAX-11/750 or VAX-11/780

SEE ALSO

make(1), sh(1), test(1), true(1).

NAME

mail, rmail - send mail to users or read mail

SYNOPSIS

`mail [+][-epqr][-f` *file* `]`

`mail [-dt]`*person* ...

`rmail [-dt]`*person* ...

Remarks:

See *mailx*(1) and *elm*(1) for an enhanced user mail interface.

DESCRIPTION

The `mail` command, when used without arguments, prints the user's mail, message-by-message, in last-in, first-out order.

For each message, `mail` prints a `?` prompt and reads a line from the standard input to determine the disposition of the message. Commands that automatically proceed to the next message exit from `mail` if `mail` already on the last message.

Commands

`mail` supports the following commands:

<new-line>	Go on to next message. Exit if already on last message.
`+`	Same as <new-line>.
`n`	Same as <new-line>.
`d`	Delete message and go on to next message.
`p`	Print message again.
`-`	Go back to previous message.
`s` [*files*]	Save message in the named *files* (default is **mbox**), mark the message for deletion from the user's *mailfile*, and proceed to next message.
`y` [*files*]	Same as `s` [*files*].
`w` [*files*]	Save message without its header (the "From ..." line), in the named *files* (default is **mbox**), mark the message for deletion, and go on to next message.
`m` *person* ...	Mail the message to each named *person*, mark the message for deletion, and go on to next message.
`q`	Put undeleted mail back in the `mailfile` and stop.
EOT (Ctrl-D)	Same as `q`.
`x`	Abort. Leave original `mailfile` unchanged and stop.
`!`*command*	Escape to the command interpreter and execute *command*.
`?`	Print a command summary.
`*`	Same as `?`.

Command-Line Options

The following command-line options alter printing of the mail:

`+`	Cause messages to be printed in first-in, first-out order.
`-e`	Suppresses printing of mail and returns the exit value: 0 = Mail present, 1 = No mail, 2 = Other error
`-p`	Prints all mail without prompting for disposition.
`-q`	Causes `mail` to terminate if an interrupt is received. Normally an interrupt only causes the termination of the printing of the current message.

`-r` Same as `+`.

`-f` *file* Causes `mail` to use *file* (for example, **mbox**) instead of the default *mailfile*.

`-t` Causes the outbound message to be preceded by each *person* the mail is sent to. A *person* is usually a user name recognized by `login` (see *login*(1)). If a *person* being sent mail is not recognized, or if `mail` is interrupted during input, the file `dead.letter` will be saved to allow editing and resending. Note that `dead.letter` is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of `dead.letter`.

`-d` Causes `mail` to deliver mail directly. This isolates `mail` from making routing decisions, and allows it to be used as a local delivery agent. Typically this option is used by auto-routing facilities when they deliver mail locally.

When *persons* are named, `mail` takes the standard input up to an end-of-file (or up to a line consisting of just a `.`) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1)). Everything after the first exclamation mark in *person* is interpreted by the remote system. In particular, if *person* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying `a!b!cde` as a recipient's name causes the message to be sent to user `b!cde` on system `a`. System `a` then interprets that destination as a request to send the message to user `cde` on system `b`. This might be useful, for instance, if the sending system can access system `a` but not system `b`. `mail` does not use `uucp` if the remote system is the local system name (i.e., localsystem!user).

The `mailfile` can be manipulated in two ways to alter the function of `mail`. The *other* permissions of the file can be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file is preserved, even when empty, to perpetuate the desired permissions. The file can also contain the first line:

`Forward to` *person*

which causes all mail sent to the owner of the `mailfile` to be forwarded to *person*. This is especially useful for forwarding all of a person's mail to a given machine in a multiple-machine environment. In order for forwarding to work properly the `mailfile` should have "mail" as group ID, and the group permission should be read-write.

`rmail` only permits the sending of mail. `uucp` uses `rmail` as a security precaution.

When a user logs in, the command `mail -e` can be used to detect the presence of mail, if any, and so indicate. When terminating, `mail` produces a notification message if new mail arrived while `mail` was running.

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of the displayed date and time strings.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `mail` behaves as if all internationalization variables are set to "C". See *environ*(5).

When set, the `TMPDIR` environment variable specifies a directory to be used for temporary files, overriding the default directory `/tmp`.

International Code Set Support

Between HP-UX systems, single- and multi-byte character code sets are supported within mail text. Headers are restricted to characters from the 7-bit USASCII code set (see *ascii*(5)).

WARNINGS

Conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed. To force printing, type a `p`.

Lines that look like postmarks in the message (that is, "From ...") are preceded by >.

`mail` treats a line consisting solely of a dot (.) as the end of the message, except when the `rmail -d` command is used.

The maximum allowable line length in mail messages is `BUFSIZ` bytes as defined in `/usr/include/stdio.h`. If line length exceeds this limit, `mail` truncates the line starting at beginning-of-line, and uses only the trailing `BUFSIZ` characters.

Using two separate mail programs to access the same mail file simultaneously (usually inadvertently from two separate windows) can cause unpredictable results.

FILES

`/usr/mail/*.lock`	Lock for mail directory
`dead.letter`	Unmailable text
`/tmp/ma*`	Temporary file
`$MAIL`	Variable containing path name of `mailfile`
`$HOME/mbox`	Saved mail
`/etc/passwd`	To identify sender and locate persons
`/usr/mail`	Directory for incoming mail (mode `775`, group ID `mail`)
`/usr/mail/`*user*	Incoming mail for *user*; that is, the **mailfile** (mode `660`, group ID `mail`)

SEE ALSO

login(1), mailx(1), uucp(1), write(1).

STANDARDS CONFORMANCE

`mail`: SVID2, XPG2, XPG3

`rmail`: SVID2

NAME

mailfrom - summarize mail folders by subject and sender

SYNOPSIS

`mailfrom` [`-n`] *[* filename *]*

DESCRIPTION

`mailfrom` outputs one line for each message in the folder specified or in the users incoming mailbox. Each line has the form;

from subject

If *filename* is specified, the program reads that file rather than the incoming mailbox. If the specified file cannot be accessed due to permissions, `mailfrom` returns the message `No mail`.

Options

`mailfrom` recognizes one option:

`-n` Number headers using the same numbering convention as used with programs such as *readmail*(1).

AUTHOR

`mailfrom` was developed by HP.

SEE ALSO

elm(1), readmail(1).

NAME

mailstats - print mail traffic statistics

SYNOPSIS

`mailstats`

DESCRIPTION

`mailstats` reads and interprets the `sendmail` statistics file, `/usr/lib/sendmail.st`, then prints out the mail traffic statistics. If `/usr/lib/sendmail.st` exists, `sendmail` collects statistics about your mail traffic and stores them in the statistics file. This file does not grow.

Statistics are gathered on a per-mailer basis for each mailer defined in the `sendmail` configuration file. Statistics are kept on the number of messages and the number of bytes for all inbound and outbound traffic.

To show the definitions of the mailers (delivery agents) reported on by mailstats, use the command:

```
grep '^M' /usr/lib/sendmail.cf
```

The first one listed is mailer 0, the second is mailer 1, etc.

To clear the statistics file, execute, as root:

```
cp /dev/null /usr/lib/sendmail.st
```

The statistics file is then invalid until some mail is actually transferred.

DIAGNOSTICS

`mailstats` generates error messages if the statistics file is not accessible or if the size of the statistics file has changed. Error messages are:

`mailstats: file size changed`
Either `/usr/lib/sendmail.st` is zero length, meaning that no mail has been transferred since it was cleared out, or its size has changed. Since the size of this file is supposed to remain constant, any change in size means that the file is invalid.

`mailstats: /usr/lib/sendmail.st: No such file or directory`
The statistics file does not exist.

`mailstats: /usr/lib/sendmail.st: Permission denied`
The statistics file's permissions are set so that you cannot read it.

EXAMPLES

Here is a typical example of `mailstats` output:

```
Statistics from Fri Dec  7 12:34:47 1990
M msgsfr  bytes_from  msgsto  bytes_to
0    216        318K      68       76K
3     80        108K      75      124K
4      0          0K      60       64K
```

This example shows that mailers 0, 3 and 4 have handled the given amounts of mail traffic since Friday, December 6th. Specifically, `sendmail` has sent 60 messages containing 64 Kbytes via mailer number 4, but has not received any inbound messages via mailer number 4.

AUTHOR

`mailstats` was developed by the University of California, Berkeley.

FILES

`/usr/lib/sendmail.st`	mail traffic statistics file
`/usr/lib/sendmail.cf`	sendmail configuration file

SEE ALSO

sendmail(1m).

NAME

mailx - interactive message processing system

SYNOPSIS

Send mode:

`mailx` [`-FU`] [`-s` *subject*] [`-r` *address*] [`-h` *number*] *address* ...

Receive mode:

`mailx -e`
`mailx` [`-UHiNn`] [`-u` *user*]
`mailx -f` [`-UHiNn`] [*filename*]

Obsolescent:

`mailx` [`-f` *filename*] [`-UHiNn`]

DESCRIPTION

`mailx` provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, `mailx` provides commands to facilitate saving, deleting, and responding to messages. When sending mail, `mailx` allows editing, reviewing and other modification of the message as it is created.

Incoming mail for each user is stored in a standard file called the **system mailbox** for that user. When using `mailx` to read messages, the system mailbox is used unless an alternate mailbox file is specified by using the `-f` option with or without a specific filename. As incoming messages are read from the system mailbox, they are marked to be moved to a secondary file for storage (unless specific action is taken) so that the messages need not be seen again. This secondary file is called the *mbox* and is usually located in the user's `HOME` directory (see `MBOX` description under ENVIRONMENT VARIABLES below for a description of this file and other environment variables used by `mailx`). Messages remain in this file until specifically removed.

Command-line options start with a hyphen (`-`), and any other arguments are assumed to be destinations (recipients). If no recipients are specified, `mailx` attempts to read messages from the system mailbox.

Options

`mailx` recognizes the following command-line options:

`-e`	Test for presence of mail. `mailx` prints nothing and exits with a successful return code if there is mail to read. Sometimes used in login scripts such as `$HOME/.profile` to check for mail during login.
`-f`	Read messages from *filename* instead of the user's system mailbox. If *filename* is not specified, the secondary *mbox* is used.
`-F`	Record the message in a file named after the first recipient. Overrides the `record` environment variable, if set.
`-h` *number*	The number of network "hops" made so far. This is provided for network software to prevent infinite delivery loops.
`-H`	Print header summary only.
`-i`	Ignore interrupts. Also see the description of the `ignore` environment below.
`-n`	Do not initialize from the system default `mailx.rc` file.
`-N`	Do not print initial header summary.
`-r` *address*	Pass *address* to network delivery software. All tilde commands are disabled.
`-s` *subject*	Set the Subject header field to *subject*.
`-u` *user*	Read *user*'s *mailbox*. Can be used only if read access to *user*'s mailbox is not read protected.
`-U`	Convert UUCP-style addresses to Internet standards. Overrides the `conv` environment variable.
-d	Turn on debugging output. Neither particularly interesting nor recommended.

When reading mail, `mailx` operates in **command mode**. A header summary of the first several messages is displayed, followed by a prompt indicating that `mailx` can accept regular commands (see COMMANDS below). When sending mail, `mailx` operates in **input mode**. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, `mailx` reads the message and stores it in a temporary file. Commands can be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

The behavior of `mailx` at any given time is governed by a set of **environment variables**; flags and valued parameters that are set and cleared by using the `set` and `unset` commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line can be of three types: login names, shell commands, or alias groups. Login names can be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is assumed to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as `lp` (see *lp*(1)) for recording outgoing mail on paper. Alias groups are set by the alias command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[*command*] [*msglist*] [*arguments*]

If no command is specified in command mode, print is assumed. In input mode, commands are recognized by the escape character (tilde unless redefined by the `escape` environment variable), and lines not treated as commands are treated as input for the message.

Each message is assigned a sequential number, and there is always the notion of a **current message**, marked by a > in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces. The message list can include:

n	Message number *n*.
.	The current message.
^	The first undeleted message.
$	The last message.
*	All messages.
n-*m*	An inclusive range of message numbers, *n* through *m*.
user	All messages from *user*.
/*string*	All messages with *string* in the subject line (uppercase-lowercase differences are ignored).
:*c*	All messages of type *c*, where *c* is one of:

`d`	deleted messages
`n`	new messages
`o`	old messages
`r`	read messages
`u`	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded using normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands, and are documented with the commands below.

At start-up time, `mailx` reads commands from a system-wide file (`/usr/lib/mailx/mailx.rc`) to initialize certain parameters, then from a private start-up file (`$HOME/.mailrc`) for personalized variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, Copy, edit, followup, Followup, hold, mail, preserve, reply, Reply, shell, and visual. Any errors in the start-up file cause the remaining lines in the file to be ignored.

COMMANDS

The following is a complete list of **mailx** commands:

! *command*	Escape to the shell. See the description of the **SHELL** environment variable below.
# *comment*	Null command (comment). Useful in **.mailrc** files.
=	Print the current message number.
?	Print a summary of commands.
<new-line>	Advance to next message and print. If this is the first command entered, the first unread message is printed. (To read the current message, use print.)
alias *alias name* ... group *alias name* ...	Declare an alias for the given names. The names are substituted when *alias* is used as a recipient. Useful in the **.mailrc** file.
alternates *name* ...	Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, alternates prints the current list of alternate names. See also **allnet** under ENVIRONMENT VARIABLES.
cd [*directory*] chdir [*directory*]	Change directory. If *directory* is not specified, **$HOME** is used.
copy [*filename*] copy [*msglist*] *filename*	Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.
Copy [*msglist*]	Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.
delete [*msglist*]	Delete messages from the *mailbox*. If **autoprint** is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES). See also **dp**.
discard [*header-field* ...] ignore [*header-field* ...]	Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The Print and Type commands override this command.
dp[*msglist*] dt[*msglist*]	Delete the specified messages from the mailbox and print the next message after the last one deleted. Roughly equivalent to a delete command followed by a print command.
echo *string* ...	Echo the given string or strings (similar to **echo** – see *echo*(1)).
edit [*msglist*]	Edit the given messages. The messages are placed in a temporary file and the **EDITOR** variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed* (see *ed*(1)).
exit xit	Exit from **mailx**, without changing the mailbox. No messages are saved in the *mbox* (see also quit).
file [*filename*] folder [*filename*]	Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, and substitutions are made as follows:

`%`	the current mailbox.
`%`*user*	the mailbox for *user*.
`#`	the previous file.
`&`	the current *mbox*.

Default file is the current mailbox.

`folders`
Print the names of the files in the directory set by the `folder` variable (see ENVIRONMENT VARIABLES).

`followup` [*message*]
Respond to a message and record the response in a file whose name is derived from the author of the message. Overrides the `record` variable, if set. See also the `Followup`, `Save`, and `Copy` commands and `outfolder` (see ENVIRONMENT VARIABLES).

`Followup` [*msglist*]
Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is extracted from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the `followup`, `Save`, and `Copy` commands and `outfolder` (see ENVIRONMENT VARIABLES).

`from` [*msglist*]
Print the header summary for the specified messages.

`group` *alias name ...*
`alias` *alias name ...*
Declare an alias for the given names. The names are substituted when *alias* is used as a recipient. Useful in the `.mailrc` file.

`headers` [*message*]
Prints the page of headers which includes the message specified. The `screen` variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the `z` command.

`help`
Prints a summary of commands.

`hold` [*msglist*]
`preserve` [*msglist*]
Holds the specified messages in the *mailbox*.

`if s|r`
  mail-commands
`else`
  mail-commands
`endif`
Conditional execution, where `s` executes the accompanying *mail-commands*, up to an `else` or `endif` if the program is in send mode, and `r` causes the accompanying *mail-commands* to be executed only in receive mode. Intended for use in `.mailrc` files.
`ignore` *header-field ...*
`discard` *header-field ...*
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are `status` and `cc`. All fields are included when the message is saved. The `Print` and `Type` commands override this command.

`list`
Prints all commands available. No explanation is given.

`mail` *name ...*
Mail a message to the specified users.

`mbox` [*msglist*]
Arrange for the given messages to end up in the standard *mbox* save file when `mailx` terminates normally. See `MBOX` description under ENVIRONMENT VARIABLES for a description of this file. See also the `exit` and `quit` commands.

`next` [*message*]
Go to next message matching *message*. A *msglist* can be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user since the name would be interpreted as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

`pipe` [*msglist*] [*command*]
`|` [*msglist*] [*command*]
Pipe messages in *msglist* through the specified *command*. Each message is treated as if it were read. If *msglist* is not specified, the current message is used. If *command* is not specified, the command specified by the current value of the `cmd` variable is used. If *msglist* is specified, *command* must also be specified. If the **`page`** variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

`preserve` [*msglist*]
`hold` [*msglist*]
Preserve the specified messages in the *mailbox*.

`Print` [*msglist*]
`Type` [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command.
`print` [*msglist*]
`type` [*msglist*]
Print the specified messages. If `crt` is set, messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` variable. The default command is `pg` (see *pg*(1)), but many users prefer **`more`** (see *more*(1) – see ENVIRONMENT VARIABLES).

`quit`
Exit from **`mailx`**, storing messages that were read in *mbox* and unread messages in the user's system mailbox. Messages that have been explicitly saved in a file are deleted.

`Reply` [*msglist*]
`Respond` [*msglist*]
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If `record` is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

`reply` [*message*]
`respond` [*message*]
Reply to the specified message, including all other recipients of the message. If `record` is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

`Save` [*msglist*]
Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is based on the author's name with all network addressing stripped off. See also the `Copy`, `followup`, and `Followup` commands and `outfolder` (see ENVIRONMENT VARIABLES).

`save` [*filename*]
`save` [*msglist*] *filename*
Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when **`mailx`** terminates unless **`keepsave`** is set (see ENVIRONMENT VARIABLES and the `exit` and `quit` commands).

`set`
`set` *name*
`set` *name=string*
`set` *name=number*
Define a variable called *name*. The variable can be given a null, string, or numeric value. `Set` by itself prints all defined variables and their values (see ENVIRONMENT VARIABLES for detailed descriptions of the **`mailx`** variables).

`shell`
Invoke an interactive shell (see **SHELL** under ENVIRONMENT VARIABLES).

`size` [*msglist*]
Print the size in characters of the specified messages.

`source` *filename*
Read commands from the given file and return to command mode.

`top` [*msglist*]
Print the top few lines of the specified messages. If the `toplines` variable is set, it is interpreted as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

`touch` [*msglist*]
Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it is placed in the *mbox* upon normal termination. See `exit` and `quit`.

`Type` [*msglist*]
`Print` [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command.

`type` [*msglist*]
`print` [*msglist*]
Print the specified messages. If `crt` is set, messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` variable. The default command is *pg*(1) but many users prefer *more*(1) (see ENVIRONMENT VARIABLES).

`unalias` *alias*
Discard the specified *alias* names.

`undelete` [*msglist*]
Restore the specified deleted messages. Restores only messages that were deleted in the current mail session. If `autoprint` is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

`unset` *name*`...`
Cause the specified variables to be erased. If the variable was a shell variable imported from the execution environment, it cannot be erased.

`version`
Prints the current version and release date.

`visual` [*msglist*]
Edit the given messages with a screen editor. The messages are placed in a temporary file and the `VISUAL` variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

`write` [*msglist*] *filename*
Write the given messages on the specified file, except for the header (the "From ..." line) and trailing blank line. Otherwise equivalent to the `save` command.

`xit`
`exit`
Exit from `mailx`, without changing the *mailbox*. No messages are saved in the *mbox* (see also `quit`).

`z`[+ | -]
Scroll the header display forward or backward one screen-full. The number of headers displayed is set by the `screen` variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands can be used only when in input mode, by beginning a line with the tilde escape character (~). See `escape` (ENVIRONMENT VARIABLES) for changing this special character.

~! *command*	Escape to the shell.
~.	Simulate end of file (terminate message input).
~: *mail-command* ~_ *mail-command*	Perform the command-level request. Valid only when sending a message while reading mail.

~?	Print a summary of tilde escapes.
~A	Insert the autograph string **Sign** into the message (see ENVIRONMENT VARIABLES).
~a	Insert the autograph string **sign** into the message (see ENVIRONMENT VARIABLES).
~b *name* ...	Add *name* to the blind carbon copy (Bcc) list.
~c *name* ...	Add *name* to the carbon copy (Cc) list.
~d	Read in the **dead.letter** file. See **DEAD** (under ENVIRONMENT VARIABLES) for a description of this file.
~e	Invoke the editor on the partial message. Also see the **EDITOR** environment variable descripton below.
~f [*msglist*]	Forward the specified messages. The messages are inserted into the message without alteration.
~h	Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it can be edited as if you had just typed it.
~i *string*	Insert the value of the named variable into the text of the message. For example, **~A** is equivalent to **~i Sign**.
~m [*msglist*]	Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.
~p	Print the message being entered.
~q	Quit (terminate) input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in **dead.letter**. See the description of the **DEAD** environment variable below for a description of this file.
~R *name* ...	Add *name* to the Reply-To list.
~r *filename*	
~< *filename*	
~<! *command*	Read in the specified file. If the argument begins with an exclamation point (**!**), the rest of the string is assumed to be an arbitrary shell command and is executed, with the standard output inserted into the message.
~s *string* ...	Set the subject line to *string*.
~t *name* ...	Add the given *name*s to the To list.
~v	Invoke a preferred screen editor on the partial message. Also see the **VISUAL** environment variable description below.
~w *filename*	Write the partial message onto the given file, without the header.
~x	Exit as with **~q** except the message is not saved in *dead.letter*.
~| *command*	Pipe the body of the message through the given *command*. If *command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following variables are internal **mailx** program variables. They can be imported from the execution environment or set by the **set** command at any time. The **unset** command can be used to erase variables.

allnet	All network names whose login names match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the **metoo** variable.
append	Upon termination, append messages to the end of the *mbox* file instead of inserting them at the beginning of the file. Default is **noappend**.
askbcc	Prompt for the Bcc list after the message is entered. Default is **noaskbcc**.
askcc	Prompt for the Cc list after the message is entered. Default is **noaskcc**.

`asksub`	Prompt for a subject if it is not specified on the command line with the `-s` option. Enabled by default.
`autoprint`	Enable automatic printing of messages after delete and undelete commands. Default is `noautoprint`.
`bang`	Enable special-case treatment of exclamation points (`!`) in shell escape command lines as in *vi*(1). Default is `nobang`.
`cmd=`*command*	Set the default command for the `pipe` command. No default value.
`conv=`*conversion*	Convert UUCP addresses to the specified address style. The only valid conversion currently supported is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also `sendmail` and the `-U` command-line option.
`crt=`*number*	Pipe messages having more than *number* lines through the command specified by the value of the `PAGER` variable `pg` by default (see *pg*(1)). Disabled by default.
`DEAD=`*filename*	The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is `$HOME/dead.letter.`
`debug`	Enable verbose diagnostics for debugging. Messages are not delivered. Default is `nodebug`.
`dot`	When processing input from a terminal, interpret an ASCII period character on a line by itself as end-of-file. Default is `nodot`.
`EDITOR=`*command*	The command to run when the edit or `~e` command is used. Default is `ed` (see *ed*(1)).
`escape=`*c*	Substitute *c* for the `~` escape character.
`folder=`*directory*	The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (`/`), `$HOME` is used as a prefix. There is no default for the `folder` variable. See also `outfolder` below.
`header`	Enable printing of the header summary when entering `mailx`. Enabled by default.
`hold`	Preserve all messages that are read in the system mailbox instead of putting them in the standard *mbox* save file. Default is `nohold`.
`ignore`	Ignore interrupts while entering messages. Useful when communicating over noisy dial-up lines. Default is `noignore`.
`ignoreeof`	Ignore end-of-file during message input. Input must be terminated by a period (`.`) on a line by itself or by the `~.` command. Default is `noignoreeof`. See also `dot` above.
`keep`	When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.
`keepsave`	Keep messages that have been saved in other files in the system mailbox instead of deleting them. Default is `nokeepsave`.
`MBOX=`*filename*	The name of the file to save messages which have been read. The `xit` command overrides this function, as does saving the message explicitly in another file. Default is `$HOME/mbox`.
`metoo`	Usually, when a group (alias) containing the sender is expanded, the sender is removed from the expansion. Setting this option causes the sender to be included in the group. Default is `nometoo`.
`LISTER=`*command*	The command (and options) to use when listing contents of the `folder` directory. The default is *ls*(1).
`onehop`	When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses,

improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

`outfolder` Cause the files used to record outgoing messages to be located in the directory specified by the `folder` variable. Default is `nooutfolder`. See `folder` above and the `Save`, `Copy`, `followup`, and `Followup` commands.

`page` Used with the `pipe` command to insert a form feed after each message sent through the pipe. Default is `nopage`.

`PAGER=` *command* The command to use as a filter for paginating output. This can also be used to specify the pager command-line options (for example, `set PAGER=`Default is `pg` (see *pg*(1)), but many users prefer `more` (see *more*(1)).

`prompt= string` Set the *command mode* prompt to *string*. Default is `?`.

`quiet` Refrain from printing the opening message and version when entering **mailx**. Default is `noquiet`.

`record=`*filename* Record all outgoing mail in *filename*. Disabled by default. See also `outfolder` above.

`save` Enable saving of messages in `dead.letter` on interrupt or delivery error. See DEAD for a description of this file. Enabled by default.

`screen=`*number* Set the number of lines in a screen-full of headers for the headers command.

`sendmail=`*command*
Alternate command for delivering messages. Default is `mail` (see *mail*(1)).

`sendwait` Wait for background mailer to finish before returning. Default is `nosendwait`.

`SHELL=`*command* The name of a preferred command interpreter. Default is `/bin/sh` (see *sh-Bourne*(1).

`showto` When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

`sign=`*string* The variable that is inserted into the text of a message when the `~a` (autograph) command is given. No default (see also `~i` under ILDE ESCAPES).

`Sign=`*string* The variable inserted into the text of a message when the `~A` command is given. No default (see also `~i` (TILDE ESCAPES)).

`SMARTMAILER` When `SMARTMAILER` is set, various commands use the `From:` line instead of the default `From` line.

`toplines=`*number*
The number of lines of header to print with the `top` command. Default is 5.

`VISUAL=`*command* The name of a preferred screen editor. Default is `vi` (see *vi*(1)).

EXTERNAL INFLUENCES

Environment Variables

The following are environment variables taken from the execution environment and are not alterable within `mailx`.

`HOME` The user's home directory. This is usually the current directory immediately after login.

`MAILRC` The name of the mailer start-up file. Default is `$HOME/.mailrc`.

`LC_COLLATE`
`LC_CTYPE` `LC_COLLATE` and `LC_CTYPE` influence `mailx` when the command interpreter (see SHELL below) is invoked. To determine the behavior of `LC_COLLATE` and `LC_CTYPE`, see the corresponding shell manual entry for the applicable comand interpreter

`LC_TIME` `LC_TIME` determines the format and contents of the date and time strings displayed. If `LC_TIME` is not specified in the environment, or is set to the empty string, the value of **LANG** is used as a default. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **mailx** behaves as if all internationalization variables are set to "C". See *environ*(5).

`TMPDIR`	When set, the **TMPDIR** environment variable specifies a directory to be used for temporary files, overriding the default directory `/tmp`.

International Code Set Support

Single- and multi-byte character code sets are supported within mail text. Headers are restricted to characters from the 7-bit USASCII character code set (see *ascii*(5)).

WARNINGS

Where *command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by **mailx**. The new internationalization standards need some time to settle down.

mail(1), the standard mail delivery program, treats a line consisting solely of a dot (**.**) as the end of the message.

Using two separate mail programs to access the same mail file simultaneously (usually inadvertently from two separate windows) can cause unpredictable results.

FILES

`/usr/mail/`	Post office directory (mode 775, group ID **mail**)
`/usr/mail/`*user*	System mailbox for *user* (mode 660, owned by *user*,*group* ID **mail**)
`$HOME/.mailrc`	Personal start-up file
`/usr/lib/mailx/mailx.help*`	Help message files
`/usr/lib/mailx/mailx.rc`	Global start-up file
`$HOME/mbox`	Secondary storage file
`/tmp/R[emqsx]*`	Temporary files

SEE ALSO

mail(1), pg(1), ls(1).

mailx tutorial in *Mail Systems Users Guide*.

STANDARDS CONFORMANCE

mailx: SVID2, XPG2, XPG3, POSIX.2

NAME

make - maintain, update, and regenerate groups of programs

SYNOPSIS

`make` [`-f` *makefile*] [`-bBdeiknpqrsSt`] [*macro_name=value*] [*names*]

DESCRIPTION

Makefile Structure

A makefile can contain four different kinds of lines: target lines, shell command lines, macro definitions, and include lines.

TARGET LINES:

Target lines consist of a blank-separated, non-null list of targets, followed by a colon (`:`) or double colon (`::`), followed by a (possibly null) list of prerequisite files called **dependents**. Pattern Matching Notation (see *regexp*(5)) is supported for the generation of file names as dependents.

SHELL COMMAND LINES:

Text following a semicolon (`;`) on a target line, and all following lines that begin with a tab are shell commands to be executed to update the target (see the Environment section below about SHELL). The first line that does not begin with a tab or # begins a new target definition, macro definition, or include line. Shell commands can be continued across lines by using a <backslash><new-line> sequence.

Target lines with their associated command lines are called *rules*.

MACROS:

Lines of the form *string1* = *string2* are macro definitions. Macros can be defined anywhere in the makefile, but are usually grouped together at the beginning. *string1* is the macro name; *string2* is the macro value. *string2* is defined as all characters up to a comment character or an unescaped new-line. Spaces and tabs immediately to the left and right of the = are ignored. Subsequent appearances of `$(`*string1*`)` anywhere in the makefile (except in comments) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. An optional substitute sequence, `$(`*string1* [`:`*subst1*`=`[*subst2*]]`)` can be specified, which causes all nonoverlapping occurrences of *subst1* at the end of substrings in the value of *string1* to be replaced by *subst2*. Substrings in a macro value are delimited by blanks, tabs, new-line characters, and beginnings of lines. For example, if

```
    OBJS = file1.o file2.o file3.o
```

then

```
    $(OBJS:.o=.c)
```

evaluates to

```
    file1.c file2.c file3.c
```

Macro values can contain references to other macros (see WARNINGS):

```
    ONE = 1
    TWELVE = $(ONE)2
```

The value of `$(TWELVE)` is set to `$(ONE)2` but when it is used in a target, command, or include line, it is expanded to `12`. If the value of `ONE` is subsequently changed by another definition further down in the makefile or on the command line, any references to `$(TWELVE)` reflect this change.

Macro definitions can also be specified on the command line and override any definitions in the makefile. Certain macros are automatically defined for **make** (see Built-in Macros). See the Environment section for a discussion of the order in which macro definitions are treated.

INCLUDE LINES:

If the string `include` appears as the first seven letters of a line in a makefile, and is followed by one or more space or tab characters, the rest of the line is assumed to be a file name and is read and processed by the current invocation of **make** as another makefile after any macros in the filename have been expanded.

General Description

`make` executes commands previously placed in a makefile to update one or more target names. Target names are typically names of programs. If no `-f` option is specified, the filenames **`makefile`**, **`Makefile`**, **`s.makefile`**, and **`s.Makefile`** are tried in that order. If `-f -` is specified, the standard input is used. More than one `-f` option can be specified. The makefile arguments are processed in the order specified. A space between the `-f` and the filename `must` be present, and multiple makefile names must each have their own `-f` option preceding them. The contents of a makefile override the built-in rules and macros if they are present.

If no target names are specified on the command line, **make** updates the first target in the (first) makefile that is not an inference rule. A target is updated only if it depends on files that are newer than the target. Missing files are deemed to be out-of-date. All dependents of a target are recursively updated, if necessary, before the target is updated. This effects a depth-first update of the dependency tree for the target.

If a target does not have any dependents specified after the separator on the target line (*explicit dependents*), any shell commands associated with that target are executed if the target is out-of-date.

A target line can have either a single or double colon between the target name or names and any explicit dependent names. A target name can appear on more than one target line, but all of those lines must be of the same (single- or double-colon) type. For the usual single-colon case, at most one of these target lines can have explicit commands associated with it. If the target is out-of-date with any of its dependents on any of the lines, the explicit commands are executed, if they are specified, or else a default rule can be executed. For the double-colon case, explicit commands can be associated with more than one of the target lines containing the target name; if the target is out-of-date with any of the dependents on a particular line, the commands for that line are executed. A built-in rule may also be executed.

Target lines and their associated shell command lines are also referred to as **rules**. Hash marks (#) and new-line characters surround comments anywhere in the makefile except in rules. Comments in the rules depend on the setting of the `SHELL` macro.

The following makefile says that `pgm` depends on two files: `a.o` and `b.o`, and that they in turn depend on their corresponding source files (`a.c` and `b.c`) and a common file `incl.h`:

```
OBJS = a.o b.o

pgm: $(OBJS)
     cc $(OBJS) -o pgm

a.o: incl.h a.c
     cc -c a.c

b.o: incl.h b.c
     cc -c b.c
```

Command lines are executed one at a time, each by its own shell. Each command line can have one or more of the following prefixes: -, @, or +. These prefixes are explained below.

Commands returning non-zero status normally terminate **make**. The `-i` option or the presence of the special target `.IGNORE` in the makefile cause **make** to continue executing the makefile regardless of how many command lines cause errors, although the error messages are still printed on standard output. If - is present at the beginning of a command line, any error returned by that line is printed to standard output but **make** does not terminate. The prefix - can be used to selectively ignore errors in a makefile. If the `-k` option is specified and a command line returns an error status, work is abandoned on the current target, but continues on other branches that do not depend on that target. If the `-k` option is present in the **MAKEFLAGS** environment variable, processing can be returned to the default by specifying the `-S` option.

The `-n` option specifies printing of a command line without execution. However, if the command line has the string `$(MAKE)` or `${MAKE}` in it or + as a prefix, the line is always executed (see discussion of the **MAKEFLAGS** macro under Environment). The `-t` (touch) option updates the modified date of a file without executing any commands.

A command line is normally printed before it is executed, but if the line has a @ at the beginning, printing is suppressed. The `-s` option or the presence of the special target `.SILENT:` in the makefile suppresses printing of all command lines. The @ can be used to selectively turn off printing. Everything printed by **make** (except the initial tab) is passed directly to the shell without alteration. Thus,

```
echo a\
b
```

produces

```
ab
```

just as the shell would.

The `-b` option allows old makefiles (those written for the old version of `make`) to run without errors. The old version of `make` assumed that if a target did not have any explicit commands associated with it, the user intended the command to be null, and would not execute any `.DEFAULT` rule that might have been defined. The current version of `make` operates in this mode by default. However, the current version of `make` provides a `-B` option which turns this mode off so that if a target does not have explicit commands associated with it and a `.DEFAULT` rule is defined, the `.DEFAULT` rule is executed. Note that the `-b` and `-B` options have no effect on the search and possible location and execution of an appropriate inference rule for the target. The search for a built-in inference rule other than `.DEFAULT` is always performed.

The signals `SIGINT`, `SIGQUIT`, `SIGHUP`, and `SIGTERM` (see *signal*(5)) cause the target to be deleted unless the target depends on the special name `.PRECIOUS`.

Options

The following is a brief description of all options and some special names. Options can occur in any order. They can be specified separately, or together with one `-`, except for the `-f` option.

`-b` Compatibility mode for old (Version 7) makefiles. This option is turned on by default.

`-B` Turn off compatibility mode for old (Version 7) makefiles.

`-d` Debug mode. Print out detailed information on files and times examined. (This is very verbose and is intended for debugging the `make` command itself.)

`-e` Environment variables override assignments within makefiles.

`-f` *makefile* Description file name, referred to as the makefile. A file name of `-` denotes the standard input. The contents of the makefile override the built-in rules and macros if they are present. Note that the space between `-f` and *makefile* `must` be present. Multiple instances of this option are allowable (except for `-f -`), and are processed in the order specified.

`-p` Write to standard output the complete set of macro definitions and target descriptions.

`-i` Ignore error codes returned by invoked commands. This mode is also entered if the special target name `.IGNORE` appears in the makefile.

`-k` When a command returns nonzero status, abandon work on the current entry, but continue on other branches that do not depend on that target. This is the opposite of `-S`. If both `-k` and `-S` are specified, the last one specified is used.

`-n` No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed. However, lines that contain the string `$(MAKE)` or `${MAKE}` or that have + as a prefix to the command are executed.

`-q` Question. The `make` command returns a zero or non-zero status code, depending on whether the target file is or is not up-to-date. Targets are not updated with this option.

`-r` Clear suffix list and do not use the built-in rules.

`-s` Silent mode. Command lines are not printed to standard output before their execution. This mode is also entered if the special target name `.SILENT` appears in the makefile.

`-S` Terminate if an error occurs while executing the commands to bring a target up-to-date. This is the default and the opposite of `-k`. If both `-k` and `-S` are specified, the last one given is used. This enables overriding the presence of the `k` flag in the **MAKEFLAGS** environment variable.

`-t` Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

[*macro_name=value*]
Zero or more command line macro definitions can be specified. See the Macros section.

[*names*] Zero or more target names that appear in the makefile. Each target so specified is updated by `make`. If no names are specified, `make` updates the first target in the makefile that is

not an inference rule.

Environment

All variables defined in the environment (see *environ*(5)) are read by `make` and are treated and processed as macro definitions, with the exception of the `SHELL` environment variable which is always ignored. `make` automatically sets `SHELL` to `/bin/sh`. Variables with no definition or empty string definitions are included by `make`.

There are four possible sources of macro definitions which are read in the following order: internal (default), current environment, the makefile(s), and command line. Because of this order of processing, macro assignments in a makefile override environment variables. The `-e` option allows the environment to override the macro assignments in a makefile. Command-line macro definitions always override any other definitions.

The `MAKEFLAGS` environment variable is processed by `make` on the assumption that it contains any legal input option (except `-f`, `-p`, and `-d`) defined for the command line. The `MAKEFLAGS` variable can also be specified in the makefile. If `MAKEFLAGS` is not defined in either of these places, `make` constructs the variable for itself, puts the options specified on the command line and any default options into it, and passes it on to invocations of commands. Thus, `MAKEFLAGS` always contains the current input options. This proves very useful for recursive `make`s. Even when the `-n` option is specified, command lines containing the string `$(MAKE)` or `${MAKE}` are executed; hence, one can perform a `make -n` recursively on an entire software system to see what would have been executed. This is possible because the `-n` is put into `MAKEFLAGS` and passed to the recursive invocations of `$(MAKE)` or `${MAKE}`. This is one way of debugging all of the makefiles for a software project without actually executing any of the commands.

Each of the commands in the rules is given to a shell to be executed. The shell used is the shell command interpreter (see *sh*(1)), or the one specified in the makefile by the `SHELL` macro. To ensure the same shell is used each time a makefile is executed, the line:

```
SHELL=/bin/sh
```

or a suitable equivalent should be put in the macro definition section of the makefile.

Suffixes

Target and/or dependent names often have suffixes. Knowledge about certain suffixes is built into `make` and used to identify appropriate inference rules to be applied to update a target (see the section on Inference Rules). The current default list of suffixes is:

```
.o .c .c~ .C .C~ .cxx .cxx~ .cpp .cpp~ .cc .cc~
.y .y~ .l .l~ .s .s~ .sh .sh~
.h .h~ .H .H~  .p .p~ .f .f~ .r .r~
```

These suffixes are defined as the dependents of the special built-in target `.SUFFIXES`. This is done automatically by `make`.

Additional suffixes can be specified in a makefile as the dependents list for `.SUFFIXES`. These additional values are added to the default values. Multiple suffix lists accumulate. The order of the suffix list is significant (see the Inference Rules section). If the user wishes to change the order of the suffixes, he must first define `.SUFFIXES` with a null dependent list, which clears the current value for `.SUFFIXES`, and then define `.SUFFIXES` with the suffixes in the desired order. The list of suffixes built into `make` on any machine can be displayed by:

```
make -fp - 2>/dev/null </dev/null
```

The list of built-in suffixes incorporated with the definitions in a given makefile called `mymakefile` can be displayed by:

```
make -fp mymakefile 2>/dev/null </dev/null
```

Inference Rules

Certain target or dependent names (such as those ending with `.o`) have inferable dependents such as `.c` and `.s`, etc. If no update commands for such a name appear in the makefile, and if an inferable dependent file exists, that dependent file is compiled to update the target. In this case, `make` has inference rules that allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. There are currently default inference rules defined for:

Single Suffix Rules

```
        .c .c~
        .C .C~ .cxx .cxx~ .cpp .cpp~ .cc .cc~
        .sh .sh~
        .p .p~
        .f .f~
        .r .r~
```

Double Suffix Rules

```
        .c.o .c~.o .c~.c .c.a .c~.a
        .C.o .C~.o .C~.C .C.a .C~.a
        .cxx.o .cxx~.o .cxx~.cxx .cxx.a .cxx~.a
        .cpp.o .cpp~.o .cpp~.cpp .cpp.a .cpp~.a
        .cc.o .cc~.o .cc~.cc .cc.a .cc~.a
        .s.o .s~.o .s~.a
        .p.o .p~.o .p~.p .p.a .p~.a
        .f.o .f~.o .f~.f .f.a .f~.a
        .r.o .r~.o .r~.r .r.a .r~.a
        .y.o .y~.o .y.c .y~.c
        .l.o .l~.o .l.c
        .h~.h  .H~.H .hxx~.hxx .hpp~.hpp
        .C.o .C~.o .C.a .C~.a
```

Double suffix inference rules (`.c.o`) define how to build `x.o` from `x.c`. Single suffix inference rules (`.c`) define how to build `x` from `x.c`. In effect, the first suffix is null. Single suffix rules are useful for building targets from only one source file; e.g., shell procedures and simple C programs.

A tilde in the above rules refers to an SCCS file (see *sccsfile*(4)). Thus, the rule `.c~.o` would transform an SCCS C source file into an object file (`.o`). Since the `s.` of the SCCS files is a prefix, it is incompatible with `make`'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule to create a file with suffix `.o` from a file with suffix `.c` is specified as an entry with `.c.o` as the target and no dependents. Shell commands associated with the target define the rule for making a `.o` file from a `.c` file. Any target name that has no slashes in it and starts with a dot is identified as an inference (*implicit*) rule instead of a target (*explicit*) rule. Targets with one dot are single suffix inference rules; targets with two dots are double suffix inference rules. Users can, in a makefile, define additional inference rules and either redefine or cancel default inference rules.

The default inference rule for changing a `.c` file into a `.o` file might look like this:

```
    .c.o:
        $(CC) $(CFLAGS) -c $<
```

and the default inference rule for changing a `yacc` file to a C object file might look like this:

```
    .y.o:
        $(YACC) $(YFLAGS) $<
        $(CC) $(CFLAGS) -c y.tab.c
        rm y.tab.c
    mv y.tab.o $@
```

Certain macros are used in the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, `CFLAGS`, `LDFLAGS`, and `YFLAGS` are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. `LDFLAGS` is commonly used to designate linker/loader options. These macros are automatically defined by `make` but can be redefined by the user in the makefile.

The macro `LIBS` is, by convention, used to specify the order of inclusion of any special libraries during the linking phase of compilation. To specify a particular order of inclusion for a particular set of libraries, the existing single suffix rule for a `.c` file,

```
    $(CC) $(CFLAGS) $< $(LDFLAGS) -o $ @
```

can be redefined as

```
    $(CC) $(CFLAGS) $< $(LDFLAGS) -o $ @ $(LIBS)
```

as well as defining **LIBS** in the makefile.

There are also some special built-in macros used in the inference rules (@, <). See the Built-in Macros section.

If a target does not have explicit dependents, or if a dependent does not also have a target that matches it with associated explicit rules, **make** looks for the first inference rule that matches both the target's (dependent's) suffix (which may be null) and a file which matches the other suffix of the rule. Since it conducts this search by going through the list of **.SUFFIXES** values front to back, the order in which **.SUFFIXES** is defined is significant.

To print out the rules compiled into the **make** on any machine, type:

```
    make -fp - 2>/dev/null </dev/null
```

Since **make** defines an inference rule **.c.o**, the example in the General Description section can be rewritten more simply:

```
    OBJS = a.o b.o
        pgm: $(OBJS)
            cc $(OBJS) -o pgm
        $(OBJS): incl.h
```

Libraries

If a target or dependent name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library that contains **file.o** (this assumes the **LIB** macro has been previously defined). The expression **$(LIB)(file1.o file2.o)** is not valid. Rules pertaining to archive libraries have the form **.*xx*.a** where *xx* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *xx* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
    lib: lib(file1.o) lib(file2.o) lib(file3.o)
          @echo lib is now up-to-date
    .c.a:
        $(CC) -c $(CFLAGS) $<
       ar rv $@ $*.o
        rm -f $*.o
```

(See the section on Built-in Macros for an explanation of the <, @, and * symbols.) In fact, the **.c.a** rule listed above is built into **make** and is unnecessary in this example. This rule is applied to each dependent of **lib** in turn. The following example accomplishes this more efficiently:

```
    lib:  lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $?
        @echo lib is now up-to-date
    .c.a:;
```

Here substitution in the macros is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution sequence translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a** rule, which would have created and archived each object file, one by one. This particular construct speeds up archive library maintenance considerably, but becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

Kernel entry points are designated by double parentheses around the entry point name, **lib((***entry_name***))**, but are otherwise handled as described above.

Built-In Targets

make has knowledge about some special targets. These must be specified in the makefile to take effect (with the exception of `.SUFFIXES`, which is automatically set by **make** but can be changed by the user).

`.DEFAULT`	If a file must be made but there are no explicit commands or relevant built-in rules for it, the commands associated with the target name `.DEFAULT` are used if `.DEFAULT` has been defined in the makefile. `.DEFAULT` does not have any explicit dependents.
`.PRECIOUS`	Dependents of this target are not removed when `QUIT`, `INTERRUPT`, `TERMINATE`, or `HANGUP` are received.
`.SILENT`	Same effect as the `-s` option. No dependents or explicit commands need to be specified.
`.IGNORE`	Same effect as the `-i` option. No dependents or explicit commands need to be specified.
`.SUFFIXES`	The explicit dependents of `.SUFFIXES` are added to the built-in list of known suffixes and are used in conjunction with the inference rules. If `.SUFFIXES` does not have any dependents, the list of known suffixes is cleared. There are no commands associated with `.SUFFIXES`.

Built-in Macros

There are five internally maintained macros that are useful for writing rules for building targets. In order to clearly define the meaning of these macros, some clarification of the terms **target** and **dependent** is necessary. When **make** updates a target, it may actually generate a series of targets to update. Before any rule (either explicit or implicit) is applied to the target to update it, recursion takes place on each dependent of the target. The dependent, upon recursion, becomes a target itself, and may have or generate its own dependents, which in turn are recursed upon until a target is found that has no dependents, at which point the recursion stops. Not all targets processed by **make** appear as explicit targets in the makefile; some of them are explicit dependents from the makefile while others are implicit dependents generated as **make** recursively updates the target. For instance, when the following makefile is executed:

```
pgm: a.o b.o
     cc a.o b.o -o pgm
```

the following series of targets to be made is generated:

`--- pgm`
with two dependents and an explicit rule to follow

`--- a.o`
(recursively) with an implicit dependent of `a.c` which matches the implicit rule `.c.o`

`--- a.c`
(recursively) with no implicit dependents and no implicit rules. This stops the recursion and simply returns the last modification time of the file `a.c`.

`--- b.o`
(recursively) with an implicit dependent of `b.c` which matches the implicit rule `.c.o`

`--- b.c`
(recursively) with no implicit dependents and no implicit rules. This stops the recursion and merely returns the last modification time of the file `b.c`.

In the definitions below, the word *target* refers to a target specified in the makefile, an explicit dependent specified in the makefile which becomes the target when **make** recurses on it, or an implicit dependent (generated as a result of locating an inference rule and file that match the suffix of the target) which becomes the target when **make** recurses on it. The word dependent refers to an explicit dependent specified in the makefile for a particular target, or an implicit dependent generated as a result of locating an appropriate inference rule and corresponding file that matches the suffix of the target.

It may be helpful to think of target rules as user specified rules for a particular target name, and inference rules as user or **make** specified rules for a particular class of target names. It may also be helpful to remember that the value of the target name and its corresponding dependent names change as **make** recurses on both explicit and implicit dependents, and that inference rules are only applied to implicit dependents or to explicit dependents which do not have target rules defined for them in the makefile.

$@ The $@ macro is the full target name of the current target, or the archive filename part of a library archive target. It is evaluated for both target and inference rules.

$% The `$%` macro is only evaluated when the current target is an archive library member of the form **libname**(*member*.o) or **libname**((*entry*.o)). In these cases, **$@** evaluates to **libname** and `$%` evaluates to **member.o** or **entry.o**. `$%` is evaluated for both target and inference rules.

$? The `$?` macro is the list of dependents that are out-of-date with respect to the current target; essentially, those modules that have been rebuilt. It is evaluated for both target and inference rules, but is usually only used in target rules. `$?` evaluates to one name only in an inference rule, but may evaluate to more than one name in a target rule.

$< In an inference rule, `$<` evaluates to the source file name that corresponds to the implicit rule which matches the suffix of the target being made. In other words, it is the file that is out-of-date with respect to the target. In the **.DEFAULT** rule, the `$<` macro evaluates to the current target name. `$<` is evaluated only for inference rules. Thus, in the `.c.o` rule, the `$<` macro would evaluate to the `.c` file. An example for making optimized `.o` files from `.c` files is:

```
.c.o:
    cc -c -O $*.c
```

or:

```
.c.o:
    cc -c -O $<
```

$*
The macro `$*` is the current target name with the suffix deleted. It is evaluated only for inference rules.

These five macros can have alternative forms. When an uppercase D or F is appended to any of the five macros, the meaning is changed to "directory part" for D and "file part" for F. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, `./` is generated. When the `$?` macro contains more than one dependent name, the `$(?D)` expands to a list of directory name parts and the `$(?F)` expands to a list of the filename parts.

In addition to the built-in macros listed above, other commonly used macros are defined by **make**. These macros are used in the default inference rules, and can be displayed with the **-p** option. These macros can be used in target rules in the makefile. They can also be redefined in the makefile.

$$@** The **$$@ macro has meaning *only* on dependency lines. Macros of this form are called **dynamic dependencies** because they are evaluated at the time the dependency is actually processed. **$$@** evaluates to exactly the same thing as **$@** does on a command line; i.e., the current target name. This macro is useful for building large numbers of executable files, each of which has only one source file. For instance, the following HP-UX commands could all be built using the same rule:

```
CMDS = cat echo cmp chown
$(CMDS) : $$@.c
    $(CC) -O $? -o $@
```

If this makefile is invoked with `make cat echo cmp chown`, **make** builds each target in turn using the generic rule, with **$$@** evaluating to `cat` while `cat` is the target, to `echo` when the target is `echo`, and so forth.

The dynamic dependency macro can also take the F form, **$$(@F)** which represents the filename part of **$$@**. This is useful if the targets contain pathnames. For example:

```
INCDIR = /usr/include
INCLUDES = $(INCDIR)/stdio.h \
       $(INCDIR)/pwd.h    \
                $(INCDIR)/dir.h    \
                $(INCDIR)/a.out.h
        $(INCLUDES) : $$(@F)
            cp $? $@
            chmod 0444 $@
```

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in pattern matching notation.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, **make** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single and multi-byte character code sets are supported.

RETURN VALUES

make returns a 0 upon successful completion or a value greater than 0 if an error occurred. If the -q option is specified, **make** returns 0 if the target was up-to-date and a value greater than 0 if the target was not up-to-date.

EXAMPLES

The following example creates an executable file from a C source code file without a makefile, if program.c exists in the current directory:

```
make program
```

The following example shows more than one makefile specified and some command line macros defined, and updates the first target in module1:

```
make -f module1 -f module2 RELEASE=1.0 CFLAGS=-g
```

The following example updates two targets in a default makefile currently residing in the current directory:

```
make clobber prog
```

The following example updates the prog target in a specified makefile, allows environment variables to override any common variables in the makefile, clears the built-in suffix list and ignore the built-in rules, and outputs exhaustive debugging information:

```
make -erd -f module1 prog
```

WARNINGS

Be wary of any file (such as an include file) whose access, modification, and last change times cannot be altered by the **make**-ing process. For example, if a program depends on an include file that in turn depends on another include file, and if one or both of these files are out-of-date, **make** tries to update these files each time it is run, thus unnecessarily re-**make**ing up-to-date files that are dependent on the include file. The solution is to manually update these files with the touch command before running **make** (see *touch*(1)). (Note that it is generally a bad idea to include the *touch*(1) command in your *makefile*, because it can cause **make** to update a program that otherwise did not need to be updated.)

Some commands return non-zero status inappropriately; use -i to overcome the difficulty.

File names with the characters **= : @ $** do not work.

Built-in commands that are directly executed by the shell such as cd (see *cd*(1)), are ineffectual across new-lines in **make**.

The syntax (lib(file1.o file2.o file3.o) is illegal.

You cannot build lib(file.o) from file.o.

The macro `$(a:.o=.c~)` does not work.

Expanded target lines cannot contain more than 16384 characters, including the terminating new-line.

Macros within other macro definitions do not expand properly when string substitution is involved.

If no makefile exists in the current directory, typing

```
make filename
```

results in `make` attempting to build `filename` from `filename.c`

If `make` is invoked in a shell script with a quoted argument that evaluates to NULL (such as $@), `make` fails.

DEPENDENCIES

NFS Warning:

When comparing modification times of files located on different NFS servers, `make` behaves unpredictably if the clocks on the servers are not synchronized.

FILES

[Mm]`akefile`
`s.`[Mm]`akefile`

SEE ALSO

cc(1), cd(1), lex(1), mkmf(1), sh(1), yacc(1), environ(5), lang(5), regexp(5).

`make` tutorial in *Programming on HP-UX.*

A Nutshell Handbook, Managing Projects With Make by Steve Talbot, Second Edition, O'Reilly & Associates, Inc., 1986.

STANDARDS CONFORMANCE

`make`: SVID2, XPG2, XPG3, POSIX.2

NAME

makekey - generate encryption key

SYNOPSIS

`/usr/lib/makekey`

DESCRIPTION

`makekey` improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, `.`, `/`, and uppercase and lowercase letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

`makekey` is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output will be pipes.

SEE ALSO

crypt(1), ed(1), passwd(4).

NAME

man - find manual information by keywords; print out a manual entry

SYNOPSIS

`man -k` *keyword* ...
`man -f` *file* ...
`man` [-] [*section* [*subsection*]] *entry_name* ...

DESCRIPTION

`man` accesses information from the on-line version of the *HP-UX Reference*. It can be used to:

- List all manual entries whose one-line description contains any of a specified set of keywords.
- Display or print one-line descriptions of entries specified by name.
- Search on-line manual directories by entry name and display or print the specified entry or entries.
- Search a specified on-line manual section (directory) and display or print the specified entry or entries in that section.

Searching for Entry Names by Keyword (first form)

The first form above searches the one-line descriptions of individual entries for specified keywords. Arguments are as follows:

`-k` *keyword* — `-k` followed by one or more keywords causes `man` to print the one-line description of each manual entry whose one-line description contains text matching one or more of the specified keywords (similar to the behavior of *grep*(1)). Keywords are separated by blanks (space or tab).

Before this option can be used, file `/usr/lib/whatis` must exist. `/usr/lib/whatis` can be created by running *catman*(1M).

Obtaining One-Line Description of an Entry (second form)

The second form above finds and displays or prints the one-line descriptions of specified individual entries. Arguments are as follows:

`-f` *file* — `-f` followed by one or more file names causes `man` to print the one-line description of each manual entry found whose name matches *file*. When specifying two or more files, *file* arguments are separated by blanks (space or tab). If entry names matching *file* exist in two or more sections, the one-line description of each matched file name is output.

Before this option can be used, file `/usr/lib/whatis` must exist. `/usr/lib/whatis` can be created by running *catman*(1M).

Viewing Individual Manual Entries (third form)

The third form shown above is used for viewing one or more individual manual entries. `man` in this form recognizes the following arguments:

`-` — (optional) When the `-` argument is present, `man` sends the formatted manual entry directly to standard output without processing it through the output filter specified by the `PAGER` environment variable.

section[*subsection*]

(optional) Search in the specified section for the given *entry_name*. *section* specifies a single section number to search for one or more the entries indicated. *section* corresponds to the section number where the entry appears in the *HP-UX Reference*. It can be followed by an optional uppercase/lowercase subsection identifier such as `3C` which would indicate a library routine in Section 3. `3`, `3c`, and `3C` are interpreted as equivalent, since all Section 3 manual entries are stored in the same or in related directories (such as `/usr/man/man3.Z` and `/usr/man/man3`. However, if an entry is in Section 1M, *section* must be specified as `1m` or `1M`.

entry_name — Search for a specific entry name where *entry_name* is the name of the manual entry without its section-number suffix. Except for names exceeding 11 characters, *entry_name* is identical to the name of the manual entry as listed at the top of each page, or is the same as one of the keywords in the left-hand part of the one-line description in the corresponding manual entry.

If *entry_name* is longer than 11 characters, **man** first searches for the full-length *entry_name*. If not found, *entry_name* is truncated to 11 characters to ensure that there is room for the *section* suffix in 14-character source file names. Files in the **/usr/man/*** directories are normally installed with the filename truncated to 11 characters where necessary so that the name plus a three-character section suffix does not exceed the maximum filename length on short filename systems.

If *section* is not specified (see next argument description), **man** searches all sections of the manual in order: **man1**, **man2**, **man1M**, **man3**, **man4**, **man5**, **man6**, **man7**, **man8**, then **man9**; and printing the first matching entry it encounters.

If the standard output is a teletype, and if the **-** flag is not given, **man** pipes its output through **more** (see *more*(1)), with the **-s** option, to eliminate multiple blank lines and stop after each screenfull. This default behavior can be changed by setting the **PAGER** variable in the user's environment. The value of **PAGER** must be a string that names an output filter (such as *pg*(1)), along with the desired options.

File Search Conventions

man searches in several directories, as appropriate, for the specified manual entry. The search continues until either the entry is found or all candidate directories are searched. The first three directories searched, in order, are: **/usr/man**, **/usr/contrib/man**, and **/usr/local/man**.

The **MANPATH** environment variable can be used to specify directories to be searched, and, if set, overrides the default paths given above. The **MANPATH** variable follows the same form as the **PATH** variable (see *environ*(5)).

Within each of these directories, **man** searches in the **cat*.Z** subdirectories, the **man*.Z** subdirectories, the **cat*** subdirectories, and the **man*** subdirectories. **man*.Z** and **man*** directories contain *nroff*(1)-compatible source text for the entries. **cat*.Z** and **cat*** directories contain the formatted versions of the entries. **man*.Z** and **cat.Z** directories contain entries in compressed form. Files in these directories are uncompressed by **uncompress** (see *compress*(1)) before being processed for printing or display.

If the **LANG** environment variable is set to any valid language name defined by *lang*(5), and the **MANPATH** variable is not set, or is set to the default directories, **man** searches in three additional directories for the manual entry before searching in **/usr/man**. First, **man** searches in **/usr/man/$LANG**, then in **/usr/contrib/man/$LANG**, then in **/usr/local/man/$LANG**. Thus, native-language manual entries are displayed if they are present and installed properly in the system.

If the **MANPATH** environment variable is set to anything other than the default, the above directories with **$LANG** as part of the path are not automatically searched. All directories must be explicitly given in **MANPATH**. The **%L**, **%l**, **%t**, and **%c** specifiers can be used as path components to cause locale-specific directories to be searched. See *environ*(5) for a complete description of **MANPATH**.

man uses the most recent version that it finds in the subdirectories searched. If the most recent version is in:

man*.Z	The entry is uncompressed, formatted, and displayed. If the **cat*.Z** directory exists, the formatted entry is compressed and installed in **cat*.Z**. If the **cat*** directory exists, the formatted entry is installed in **cat***.
cat*.Z	The entry is uncompressed and displayed.
man*	The entry is formatted, and displayed. If the **cat*.Z** directory exists, it is compressed, and installed in **cat*.Z**. If the **cat*** directory exists, the formatted entry is installed in **cat***.
cat*	The entry is displayed.

If only the **cat*** or **cat*.Z** subdirectory is present and/or *nroff*(1) is not installed, only entries that are already formatted can be displayed.

If you choose to have the formatted entries on your system, run *catman*(1M) with the default, which creates the **cat*.Z** directories (after removing any **cat*** directories that exist on your system) and also creates the file **/usr/lib/whatis** used by the **man -k** option. If you choose to have the **cat*** directories, it would be space-saving to remove any **cat*.Z** directories that may exist on your system. Beware that **man** updates both directories (**cat*** and **cat*.Z**) if they both exist.

Special Manual Entries

Some situations may require creation of manual entries for local use or distribution by third-party software suppliers. The manual formatting macros have been structured to redefine page footers so that manual entries not originating from Hewlett-Packard Company do not show the **HP** name in the footer. For more information about this change and a description of the manual formatting macros used with `nroff` or `troff`, see *man*(5).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed. **LANG** is also used to determine the search path (as described above).

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG** for messages, but not for the search path.

If any internationalization variable contains an invalid setting, **man** behaves as if all internationalization variables are set to "C". See *environ*(5).

MANPATH, if set, gives a list of directories to be searched for the given entry, replacing the default paths.

`PAGER` , if set, defines an output filter to be used instead of *more*(1) to paginate output.

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

List the manual entries that contain the word `copy` and `password` in their respective one-line description (NAME) lines:

```
man -k grep
```

Text similar to the following results:

```
grep, egrep, fgrep (1) - search a file for a pattern
```

Print the one-line description of the *grep*(1) manual entry:

```
man -f grep
```

Print the entire *grep*(1) manual entry:

```
man grep
```

Set a search path that includes a path directly below the current directory. The manual entry, `mypage` is assumed to exist in the directory `./man1` (or `./man1.Z`, `cat1`, or `cat1.Z`).

```
MANPATH=.:/usr/man:/usr/contrib/man:/usr/local/man
export MANPATH
man mypage
```

Display the manual entry for *id*(1), with the output piped through `pg -c`:

```
PAGER="pg -c"
export PAGER
man id
```

List all printed manuals available for the current system (see *manuals*(5):

```
man manuals
```

WARNINGS

Manual entries are structured such that they can be printed on a phototypesetter, conventional line printer, and screen display devices. However, due to line printer and display device limitations, some information may be lost in certain situations.

FILES

`/usr/lib/whatis`	keyword database
`/usr/man/cat*[.Z]/*`	formatted manual entries [compressed]
`/usr/man/man*[.Z]/*`	raw (*nroff*(1) source) manual entries [compressed]

`/usr/contrib/man/cat*[.Z]/*`	
`/usr/contrib/man/man*[.Z]/*`	
`/usr/local/man/cat*[.Z]/*`	
`/usr/local/man/man*[.Z]/*`	
`/usr/man/$LANG/cat*[.Z]/*`	formatted native-language manual entries [compressed]
`/usr/man/$LANG/man*[.Z]/*`	raw (*nroff*(1) source) native-language manual entries [compressed]
`/usr/contrib/man/$LANG/cat*[.Z]/*`	
`/usr/contrib/man/$LANG/man*[.Z]/*`	
`/usr/local/man/$LANG/cat*[.Z]/*`	
`/usr/local/man/$LANG/man*[.Z]/*`	

SEE ALSO

col(1), compress(1), fixman(1), grep(1), more(1), catman(1M), environ(5), man(5) manuals(5).

NAME

mediainit - initialize disk or cartridge tape media, partition DDS tape

SYNOPSIS

`mediainit` [`-vr`] [`-f` *fmt_optn*] [`-i` *interleave*] [`-p` *size*] *pathname*

DESCRIPTION

`mediainit` initializes mass storage media by formatting the media, writing and reading test patterns to verify media integrity, then sparing any defective blocks found. This process prepares the disk or tape for error-free operation. Initialization destroys all existing user data in the area being initialized.

`mediainit` can also used for partitioning DDS tape media. See the `-p` option below for further details.

Options

The following command options are recognized. They can be specified in any order, but all must precede the *pathname*. Options without parameters can be listed individually or grouped together. Options with parameters must be listed individually, but white space between the option and its parameter is discretionary.

`-v` Normally, `mediainit` provides only fatal error messages which are directed to standard error. The `-v` (verbose) option sends device-specific information related to low-level operation of `mediainit` to standard output (stdout). This option is most useful to trained service personnel because it usually requires detailed knowledge of device operation before the information can be interpreted correctly.

`-r` (re-certify) This option forces a complete tape certification whether or not the tape has been certified previously. All record of any previously spared blocks is discarded, so any bad blocks will have to be rediscovered. This option should be used only if:

- It is suspected that numerous blocks on the tape have been spared which should not have been, or
- It is necessary to destroy (overwrite) all previous data on the tape.

`-f` *fmt_optn*
The format option is a device-specific number in the range `0` through `239`. It is intended solely for use with certain SS/80 devices that support multiple media formats (independent from interleave factor). For example, certain microfloppy drives support 256-, 512-, and 1024-byte sectors. `mediainit` passes any supplied format option directly through to the device. The device then either accepts the format option if it is supported, or rejects it if it is not supported. Refer to device operating manuals for additional information. The default format option is `0`.

`-i` *interleave*
The interleave factor, *interleave*, refers to the relationship between sequential logical records and sequential physical records. It defines the number of physical records on the media that lie between the beginning points of two consecutively numbered logical records. The choice of interleave factor can have a substantial impact on disk performance. For CS/80 and SS/80 drives, consult the appropriate device operating manual for details. For Amigo drives, see DEPENDENCIES.

`-p` *size*
Partition DDS cartridge media into two logical separate volumes: partition 0 and partition 1:

- *size* specifies the minimum size of partition 1 (in Mbytes).
- Partition 0 is the remainder of the tape (partition 0 physically follows partition 1 on the tape).

The actual size of partition 1 is somewhat larger than the requested size to allow for tape media errors during writing. Thus, a *size* of 400 formats the DDS tape into two partitions where partition 1 holds at least 400 Megabytes of data, and the remainder of the tape is used for partition 0 (for a 1300 Mbyte DDS cartridge, this means that partition 0 has a size somewhat less than 900 Mbytes).

Note that it is unnecessary to format a DDS tape before use unless the tape is being partitioned. Unformatted DDS media does not require initialization when used as a single partition tape. Accessing partition 1 on a single-partition tape produces an error. To change a two-partition tape to single-partition, use `mediainit` with `0` specified as the *size*.

pathname
pathname is the path name to the character (raw) device special file associated with the device unit or volume that is to be initialized. `mediainit` aborts if you lack either read or write permission to the device special file, or if the device is currently open for any other process. This prevents accidental initialization of the root device or any mounted volume. See DEPENDENCIES for additional Series 800 requirements. `mediainit` locks the unit or volume being initialized so that no other processes can access it.

When a given CS/80 or SS/80 device contains multiple units, or a given unit contains multiple volumes as defined by the drive controller, any available unit or volume associated with that controller can be initialized, independent of other units and volumes that share the same controller. Thus, you can initialize one unit or volume to any format or interleave factor without affecting formats or data on companion units or volumes. However, be aware that the entire unit or volume (as defined by the drive controller) is initialized without considering the possibility that it may be subdivided into smaller structures by the the operating software. When such structures exist, unexpected loss of data is possible.

`mediainit` dominates controller resources and limits access by competing processes to other units or volumes sharing the same controller. If other simultaneous processes need access to the same controller, some access degradation can be expected until initialization is complete; especially if you are initializing a tape cartridge in a drive that shares the root disk controller. See Series 800 DEPENDENCIES for additional Series 800 information.

In general, `mediainit` attempts to carefully check any `-f` (format option) or `-i` (interleave options) supplied, and aborts if an option is out of range or inappropriate for the media being initialized. Specifying an interleave factor or format option value of `0` has the same effect as not specifying the option at all.

For disks that support interleave factors, the acceptable range is usually `1` (no interleave) through $n-1$, where n is the number of sectors per track. With SS/80 hard disks, the optimum interleave factor is usually determined by the speed (normal or high) of the HP-IB interface card used and whether DMA is present in the system. The optimum interleave factor for SS/80 flexible disk drives is usually a constant (often 2), and is independent of the type of HP-IB interface used. The optimum interleave factor for CS/80 disks is usually `1` and is also usually not related to the type of HP-IB interface being used. In any case, refer to the appropriate device operating manual for recommended values.

If a disk being initialized requires an interleave factor but none is specified, `mediainit` provides an appropriate, though not necessarily optimum default. For CS/80 and SS/80 disks, `mediainit` uses whatever the device reports as its current interleave factor. SS/80 floppy drives report their minimum (usually best) interleave factor, if the currently installed media is unformatted.

When a given device supports format options, the allowable range of interleave factors may be related to the specified format option. In such instances, `mediainit` cannot check the interleave factor if one is specified.

Notes

Most types of mass storage media must be initialized before they can be used. HP hard disks, flexible disks, and cartridge tapes require some form of initialization, but 9-track tapes do not. Initialization usually involves formatting the media, writing and reading test patterns, then sparing any defective blocks. Depending upon the media and device type, none, some, or all of the initialization process may have been performed at the factory. `mediainit` completes whatever steps are appropriate to prepare the media for error-free operation.

Most HP hard disks are formatted and exhaustively tested at the factory by use of a process more thorough but also more time-consuming than appropriate for `mediainit`. However, `mediainit` is still valuable for ensuring the integrity of the media after factory shipment, formatting with the correct interleave factor, and sparing any blocks which may have become defective since original factory testing was performed.

HP flexible disks are not usually formatted prior to shipment, so they must undergo the entire initialization process before they can be used.

All HP CS/80 cartridge tapes are certified and formatted prior to shipment from the factory. When a tape is certified, it is thoroughly tested and defective blocks are spared. `mediainit` usually certifies a tape only if it has not been certified previously. If the tape has been previously certified and spared, `mediainit` usually reorganizes the tape's spare block table, retaining any previous spares, and optimizing their assignment for maximum performance under sequential access. Reorganizing the spare block table takes only a few seconds, whereas complete certification takes about a half-hour for 150-foot tapes, and over an hour for

600-foot tapes.

HP CS/80 cartridge tape drives have a feature called "auto-sparing". If under normal usage the drive has trouble reading a block, the drive logs the fact and automatically spares out that block the next time data is written to it. Thus, as a tape is used, any marginal blocks that were not spared during certification are spared automatically if they cause problems. This sparing is automatic within the device, and is totally independent of `mediainit`.

Reorganization of a tape's spare block table technically renders any existing data undefined, but the data is not usually destroyed by overwriting. To ensure that old tape data is destroyed, which is useful for security, complete tape re-certification can be forced with the `-r` option.

Some applications may require that a file system be placed on the media before use. `mediainit` does not create a file system; it only prepares media for writing and reading. If such a file system is required, other utilities such as `newfs`, `lifinit`, or `mkfs` must be invoked after running `mediainit` (see *newfs*(1M), *lifinit*(1), and *mkfs*(1M)).

RETURN VALUE

`mediainit` returns one of the following values:

0	Successful completion.
1	A device-related error occurred.
2	A syntax-related error was encountered.

ERRORS

Appropriate error messages are printed on standard error during execution of `mediainit`.

EXAMPLES

Format an HP 9122 SS/80 3-1/2-inch flexible disk with an interleave factor of 2, 1024-byte sectors, and double-sided HP format:

```
mediainit -i 2 -f 3 /dev/rdsk/9122
```

Format a floppy drive and create a file system on it:

```
mediainit -v -i2 -f16 /dev/rdsk/floppy
newfs -n /dev/rdsk/floppy ibm1440
```

WARNINGS

Aborting `mediainit` is likely to leave the medium in a corrupt state, even if it was previously initialized. To recover, the initialization must be restarted.

DEPENDENCIES

Series 300/400

Partitioning of DDS tape media is not supported on HP-IB DDS drives.

Series 300/400 systems support various Amigo disk drives. Acceptable interleave factors for Amigo devices are as follows:

Device	Range	Default
HP 9895 SS/DS	1 - 29	2
HP 8290X	1 - 15	3
HP 9121	1 - 15	2
HP 9133V	na	9
HP 9133XV	na	9
HP 9134XV	na	9

Series 700/800

Except for SCSI devices, *pathname* must be a device special file whose minor number of the device being initialized has the diagnostic bit set (the diagnostic bit is the most significant bit in the minor number). For device special files with the diagnostic bit set, the section number is meaningless. The entire device is accessed.

For a device that contains multiple units on a single controller, each unit can be initialized independently from any other unit. It should be noted, however, that `mediainit` requires that there be no other processes accessing the device before initialization begins, regardless of which unit is being initialized. If there are accesses currently in progress, `mediainit` aborts. During the initialization process, `open()`

rejects all other accesses to the device being initialized, producing the error EACCES (see *open*(2)).

Series 800

Partitioning of DDS tape media (`-p` option) is not supported.

AUTHOR

`mediainit` was developed by HP.

SEE ALSO

lifinit(1), mkfs(1M), newfs(1M).

NAME

merge - three-way file merge

SYNOPSIS

`merge` [`-p`] *file1 file2 file3*

DESCRIPTION

`merge` combines two files that are revisions of a single original file. The original file is *file2*, and the revised files are *file1* and *file3*. `merge` identifies all changes that lead from *file2* to *file3* and from *file2* to *file1*, then deposits the merged text into *file1*. If the `-p` option is used, the result goes to standard output instead of *file1*.

An overlap occurs if both *file1* and *file3* have changes in the same place. `merge` prints how many overlaps occurred, and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, edit the result in *file1* and delete one of the alternatives.

This command is particularly useful for revision control, especially if *file1* and *file3* are the ends of two branches that have *file2* as a common ancestor.

EXAMPLES

A typical use for `merge` is as follows:

1. To merge an RCS branch into the trunk, first check out the three different versions from RCS (see *co*(1)) and rename them for their revision numbers: `5.2`, `5.11`, and `5.2.3.3`. File `5.2.3.3` is the end of an RCS branch that split off the trunk at file `5.2`.
2. For this example, assume file `5.11` is the latest version on the trunk, and is also a revision of the "original" file, `5.2`. Merge the branch into the trunk with the command:

   ```
   merge 5.11 5.2 5.2.3.3
   ```

3. File `5.11` now contains all changes made on the branch and the trunk, and has markings in the file to show all overlapping changes.
4. Edit file `5.11` to correct the overlaps, then use the `ci` command to check the file back in (see *ci*(1)).

WARNINGS

`merge` uses the *ed*(1) system editor. Therefore, the file size limits of *ed*(1) apply to `merge`.

AUTHOR

`merge` was developed by Walter F. Tichy.

SEE ALSO

diff3(1), diff(1), rcsmerge(1), co(1).

NAME

mesg - permit or deny messages to terminal

SYNOPSIS

mesg n
mesg y
mesg

DESCRIPTION

The command form **mesg n** forbids messages via **write** by revoking non-user write permission on the user's terminal (see *write*(1)). **mesg y** reinstates permission. **mesg** without any other argument reports the current state without changing it.

FILES

`/dev/tty*`

RETURN VALUE

mesg returns the following values:

0	Messages are receivable.
1	Messages are not receivable.
2	An error occurred.

SEE ALSO

write(1).

STANDARDS CONFORMANCE

mesg: SVID2, XPG2, XPG3

NAME
mkdir - make a directory

SYNOPSIS
`mkdir [-p] [-m` *mode* `]` *dirname* ...

DESCRIPTION
`mkdir` creates specified directories in mode 0777 (possibly altered by `umask` unless specified otherwise by a `-m` *mode* option (see *umask*(1)). Standard entries, `.` (for the directory itself) and `..` (for its parent) are created automatically. If *dirname* already exists, `mkdir` exits with a diagnostic message, and the directory is not changed.

Options
`mkdir` recognizes the following command-line options:

`-m` *mode* After creating the directory as specified, the file permissions are set to *mode*, which is a symbolic mode string as defined for `chmod` (see *chmod*(1)).

`-p` Intermediate directories are created as necessary. Otherwise, the full path prefix of *dirname* must already exist. `mkdir` requires write permission in the parent directory.

For each directory name in the pathname prefix of the *dirname* argument that is not the name of an existing directory, the specified directory is created using the current `umask` setting, except that the equivalent of `chmod u+wx` is done on each component to ensure that `mkdir` can create lower directories regardless of the setting of `umask`. Each directory name in the pathname prefix of the *dirname* argument that matches an existing directory is ignored without error. If an intermediate path component exists, but has permissions set to prevent writing or searching, `mkdir` fails with an error message. If the *dirname* argument (including pathname prefix) names an existing directory, `mkdir` fails with an error message.

If the `-m` option is used, the directory specified by *dirname* (excluding directories in the pathname prefix) is created with the permissions specified by *mode*.

Only `LINK_MAX` subdirectories can be created (see *limits*(5)).

EXTERNAL INFLUENCES
Environment Variables
LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `mkdir` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support
Single- and multi-byte character code sets are supported.

DIAGNOSTICS
`mkdir` returns exit code 0 if all directories were successfully made. Otherwise, it prints a diagnostic and returns non-zero.

EXAMPLES
Create directory `gem` beneath existing directory `raw` in the current directory:

```
mkdir raw/gem
```

Create directory path `raw/gem/diamond` underneath the current directory and set permissions on directory `diamond` to read-only for all users (`a=r`):

```
mkdir -p -m "a=r" raw/gem/diamond
```

which is equivalent to (see *chmod*(1)):

```
mkdir -p -m 444 raw/gem/diamond
```

If directories `raw` or `raw` and `gem` already exist, only the missing directories in the specified path are created.

SEE ALSO

rm(1), sh(1), umask(1).

STANDARDS CONFORMANCE

`mkdir`: SVID2, XPG2, XPG3, POSIX.2

NAME

mkfifo - make FIFO (named pipe) special files

SYNOPSIS

`mkfifo` [`-p`] [`-m` *mode*] *filename* ...

DESCRIPTION

`mkfifo` creates the FIFO special files named by its operand list. The operands are taken sequentially in the order specified and, if the user has write permission in the appropriate directory, the FIFO is created with permissions 0666 modified by the user's file mode creation mask (see *umask*(2)).

The specific actions performed are equivalent to calling

`mkfifo(`*filename*`, 0666)`

for each filename in the operand list (see *mkfifo*(2)).

Options

`mkfifo` recognizes the following command-line options:

`-m` *mode*	After creating the FIFO special file, set the permission bits of the new file to the specified *mode* value. The *mode* option argument is a symbolic mode string as defined in *chmod*(1).
`-p`	Create any missing intermediate path name components.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `mkfifo` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

RETURN VALUE

`mkfifo` returns zero if invoked with at least one operand and if all FIFO special files were created successfully. Otherwise, it prints a diagnostic message and returns non-zero.

EXAMPLES

The following command creates a FIFO special file named `peacepipe` in the current directory:

```
mkfifo peacepipe
```

SEE ALSO

chmod(1), umask(1), mknod(1M), mkfifo(3C).

STANDARDS CONFORMANCE

`mkfifo`: XPG3, POSIX.2

NAME

mkmf - make a makefile

SYNOPSIS

`mkmf` [`-acdeil`] [`-f` *makefile*] [`-F` *template*] [`-M` *language*] [*macroname*=*value* ...]

DESCRIPTION

`mkmf` creates a makefile that informs the `make` command how to construct and maintain programs and libraries (see *make*(1)). After gathering up all source code file names in the current working directory and inserting them into the makefile, `mkmf` scans source code files for included files and generates dependency information that is appended to the makefile. Source code files are identified by their file name suffixes. `mkmf` recognizes the following suffixes:

`.c`	C
`.C`	C++
`.f`	FORTRAN
`.h`	Include files
`.i`	Pascal include files
`.l`	Lex or Lisp
`.o`	Object files
`.p`	Pascal
`.r`	Ratfor
`.s`	Assembler
`.y`	Yacc

`mkmf` checks for an existing makefile before creating one. If no `-f` option is present, `mkmf` tries the makefiles `makefile` and `Makefile`, respectively.

After the makefile has been created, arbitrary changes can be made using a text editor. `mkmf` can also be used to re-edit the macro definitions in the makefile, regardless of changes that may have been made since it was created.

By default, `mkmf` creates a program makefile. To create a makefile that handles libraries, the `-l` option must be used.

Make Requests

Given a makefile created by `mkmf`, `make` recognizes the following requests:

`all`	Compile and load a program or library.
`clean`	Remove all object and core files.
`clobber`	Remove all files that can be regenerated.
`depend`	Update included file dependencies in a makefile.
`echo`	List the names of the source code files on standard output.
`extract`	Extract all object files from the library and place them in the same directory as the source code files. The library is not altered.
`index`	Print an index of functions on standard output.
`install`	Compile and load the program or library and move it to its destination directory.
`print`	Print source code files on standard output.
`tags`	Create a tags file for the `ex` editor (see *ex*(1) and *ctags*(1)), for C, Pascal, and Fortran source code files.
`update`	Recompile only if there are source code files that are newer than the program or library, link and install the program or library.

Several requests can be given simultaneously. For example, to (1) compile and link a program, (2) move the program to its destination directory, and (3) remove any unnecessary object files, use:

```
make install clean
```

Macro Definitions

`mkmf` understands the following macro definitions:

`CFLAGS`	C compiler flags. After searching for included files in the directory currently being processed, `mkmf` searches in directories named in `-I` compiler options and then in the `/usr/include` directory.
`COMPILESYSTYPE`	Location of `/usr/include`. If the `COMPILESYSTYPE` macro or environment variable is defined, `mkmf` searches for included files in `/$COMPILESYSTYPE/usr/include` instead of `/usr/include`.
CXXFLAGS	C++ compiler flags. After searching for included files in the directory currently being processed, `mkmf` searches in directories named in `-I` compiler options and then in the `/usr/include/CC` directory, followed by the `/usr/include` directory.
`DEST`	Directory where the program or library is to be installed.
`EXTHDRS`	List of included files external to the current directory. `mkmf` automatically updates this macro definition in the makefile if dependency information is being generated.
`FFLAGS`	Fortran compiler flags. After searching for included files in the directory currently being processed, `mkmf` searches in directories named in `-I` compiler options, then in the `/usr/include` directory.
`HDRS`	List of included files in the current directory. `mkmf` automatically updates this macro definition in the makefile.
`INSTALL`	Installation program name.
`LD`	Link editor name.
`LDFLAGS`	Link editor flags.
`LIBRARY`	Library name. This macro also implies the `-l` option.
`LIBS`	List of libraries needed by the link editor to resolve external references.
`MAKEFILE`	Makefile name.
`OBJS`	List of object files. `mkmf` automatically updates this macro definition in the makefile.
`PROGRAM`	Program name.
`SRCS`	List of source code files. `mkmf` automatically updates this macro definition in the makefile.
`SUFFIX`	List of additional file name suffixes for `mkmf` to know about.
`SYSHDRS`	List of included files found in the `/usr/include` directory hierarchy. `mkmf` automatically updates this macro definition in the makefile if dependency information is being generated. If `SYSHDRS` is omitted from the makefile, `mkmf` does not generate `/usr/include` dependencies.

Both these and any other macro definitions already within the makefile can be replaced by definitions on the command line in the form *macroname=value*. For example, to change the C compiler flags and the program name, type the following line:

```
mkmf "CFLAGS=-I../include -O" PROGRAM=mkmf
```

Note that macro definitions such as `CFLAGS` with blanks in them must be enclosed in double quote (") marks.

Environment

The environment is read by `mkmf`. All variables are assumed to be macro definitions with the exception of HDRS, `EXTHDRS`, `SRCS`, and `OBJS`. Environment variables are processed after command line macro definitions and the macro definitions in a *makefile*. The `-e` option forces the environment to override the macro definitions in a *makefile*.

File Name Suffixes

`mkmf` can recognize additional file name suffixes, or ignore ones that it already recognizes, by specifying suffix descriptions in the `SUFFIX` macro definition. Each suffix description takes the form *.suffix*`:`*tI* where *t* is a character indicating the contents of the file (`s` = source file, `o` = object file, `h` = header file, `x` = executable file) and *I* is an optional character indicating the include syntax for header files (`C` = C syntax, `C++` = C syntax plus the addition of `/usr/include/CC` as a standard search directory, `F` = Fortran and Ratfor syntax, `P` = Pascal syntax). The following list shows the default configuration for `mkmf`:

`.c:sC`	C
`.C:sC++`	C++
`.f:sF`	Fortran
`.h:h`	Include files
`.i:h`	Pascal include files
`.l:sC`	Lex or Lisp
`.o:o`	Object files
`.p:sP`	Pascal
`.r:sF`	Ratfor
`.s:s`	Assembler
`.y:sC`	Yacc

For example, to change the object file suffix to `.obj`, undefine the Pascal include file suffix, and prevent Fortran files from being scanned for included files, the `SUFFIX` macro definition could be:

```
SUFFIX = .obj:o .i: .f:s
```

Include Statement Syntax

The syntax of include statements for C, C++, Fortran, and Pascal source code are of the form:

C/C++:

`#include "`*filename*`"`
`#include` *filename*

where `#` must be the first character in the line.

Fortran:

```
$include '  filename '$
$INCLUDE '  filename '$
```

where `$` must be the first character in the line. Alternatively, the `$` can be omitted if the include statement starts in column 7. In either case the trailing `$` can be omitted.

Pascal:

```
$include '  filename '$
$INCLUDE '  filename '$
```

where `$` must be the first character in the line and the trailing `$` is optional.

User-Defined Templates

If `mkmf` cannot find a makefile within the current directory, it normally uses one of the standard makefile templates, `C.p` or `C.l`, in `/usr/lib/mf` unless the user has alternative `C.p` or `C.l` template files in a directory `$PROJECT/lib/mf` where `$PROJECT` is the absolute path name of the directory assigned to the `PROJECT` environment variable.

Options

`mkmf` recognizes the following options:

`-a`	Include source files beginning with a `.` in the makefile.
`-c`	Suppress "`creating` *makefile* `from` ..." message.
`-d`	Turn off scanning of source code for `include` files. Old dependency information is left untouched in the makefile.
`-e`	Environment variables override macro definitions within *makefile*s.
`-f` *makefile*	Specify an alternative *makefile* file name. The default file name is `Makefile`.
`-i`	Prompt the user for the name of the program or library and the directory where it is to be installed. If a carriage-return is typed in response to each of these queries, `mkmf` assumes that the default program name is `a.out` or the default library name is `lib.a`, and the destination directory is the current directory.

`-l`	Force the makefile to be a library makefile.
`-F` *template*	Specify an alternative makefile template path name. The path name can be relative or absolute.
`-M` *language*	Specify an alternative *language*-specific makefile template. The default language is C and the corresponding program and library makefile templates are `C.p` and `C.l`, respectively. **mkmf** looks for these templates in `/usr/lib/mf` or `$PROJECT/lib/mf`.

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

WARNINGS

The name of the makefile is included as a macro definition within the makefile and must be changed if the makefile is renamed.

Since executable files are dependent on libraries, standard library abbreviations must be expanded to full path names within the `LIBS` macro definition in the makefile.

Generated dependency information appears after a line in the makefile beginning with `###`. This line must not be removed, nor must any other information be inserted in the makefile below this line.

The name of a program or library must not conflict with any predefined target names in a makefile. It is especially important to avoid the the name `update` to prevent `make` from recursively executing itself an infinite number of times.

AUTHOR

mkmf was developed by the University of California, Berkeley.

FILES

`/usr/lib/mf/C.p`	Standard program makefile template
`/usr/lib/mf/C.l`	Standard library makefile template
`$PROJECT/lib/mf/C.p`	User-defined program makefile template
`$PROJECT/lib/mf/C.l`	User-defined library makefile template

SEE ALSO

ar(1), ctags(1), ld(1), make(1).

make tutorial in *Programming on HP-UX*.

"Automatic Generation of Make Dependencies", *Software-Practice and Experience*, Walden, K., vol. 14, no. 6, pp. 575-585, June 1984.

NAME

mkstr - extract error messages from C source into a file

SYNOPSIS

`mkstr` [`-`] *messagefile prefix file* ...

DESCRIPTION

`mkstr` examines a C program and creates a file containing error message strings used by the program. Programs with many error diagnostics can be made much smaller by referring to places in the file, and reduce system overhead in running the program.

`mkstr` processes each of the specified *file*s, placing a revised version of each in a file whose name consists of the specified *prefix* concatenated in front of the original name. A typical usage of *mkstr* would be

```
mkstr mystrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *mystrings* and revised copies of the source for these files to be placed in files whose names are prefixed with *xx*.

When processing the error messages in the source for transfer to the message file, `mkstr` searches for the string `error(` in the input file. Each time it is encountered, the C string starting after the leading quote is placed in the message file, followed by a null character and a new-line character. The null character terminates the message so that it can be easily used when retrieved, and the new-line character makes it possible to conveniently list the error message file (using `cat`, `more`, etc. — see *cat*(1) and *more*(1)) to review its contents.

The modified copy of the input file is identical to the original, except that each occurrence of any string that was moved to the error message file is replaced by an offset pointer usable by `lseek` to retrieve the message.

If the command line includes the optional `-`, extracted error messages are placed at the end of the specified message file (append) instead of overwriting it. This enables you to process individual files that are part of larger programs that have been previously processed by `mkstr` without reprocessing all the files.

All functions used by the original program whose names end in "error" that also can take a constant string as their first argument should be rewritten so that they search for the string in the error message file.

For example, a program based on the previous example usage would resemble the following:

```
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>

char errfile[] = "mystrings" ;

error(offset, a2, a3, a4)
int offset, a1, a2, a3;
{
    char msg[256];
    static int fd = -1;

    if (fd < 0) {
        fd = open(errfile, O_RDONLY);
        if (fd < 0) {
     perror(errfile);
     exit(1);
        }
    }

    if (lseek(fd, (off_t) offset, 0) || read(fd, msg, 256) <= 0) {
        printf("? Can't find error message in %s:\n", errfile);
        perror(errfile);
        exit(1);
    }
```

```
            printf(msg, a1, a2, a3);
        }
```

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of comments and string literals as single- and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `mkstr` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported within file names, comments, and string literals.

SEE ALSO

lseek(2), perror(3C), xstr(1).

BUGS

Strings in calls to functions whose names end in `error`, notably `perror()`, may be replaced with offsets by `mkstr`.

Calls to error functions whose first argument is not a string constant are left unmodified without warning.

NAME

mktemp - make a name for a temporary file

SYNOPSIS

`mktemp` [`-c`] [`-d` *directory_name*] [`-p` *prefix*]

DESCRIPTION

`mktemp` makes a name that is suitable for use as the pathname of a temporary file, and writes that name to the standard output. The name is chosen such that it does not duplicate the name of an existing file. If the `-c` option is specified, a zero-length file is created with the generated name.

The name generated by `mktemp` is the concatenation of a directory name, a slash (/), the value of the **LOGNAME** environment variable truncated to **{NAME_MAX}** – 6 characters, and the process ID of the invoking process.

The directory name is chosen as follows:

1. If the `-d` option is specified, *directory_name* is used.
2. Otherwise, if the **TMPDIR** environment variable is set and a string that would yield a unique name can be obtained by using the value of that variable as a directory name, this value is used.
3. Otherwise, if a string that would yield a unique name can be obtained using **/tmp** as the directory, **/tmp** is used.
4. Otherwise, **.** (current directory) is used.

If the `-p` option is specified, *prefix* is used instead of the value of the **LOGNAME** environment variable for name generation.

RETURN VALUE

`mktemp` returns zero on successful completion and non-zero if syntax, file access, or file creation errors were encountered or a unique pathname could not be generated.

SEE ALSO

mktemp(3C), umask(1).

NAME

mm, osdd - print documents formatted with the mm macros

SYNOPSIS

mm [*options*] [*files*]

osdd [*options*] [*files*]

DESCRIPTION

mm can be used to format and print documents using `nroff` and the mm text-formatting macro package (see *nroff*(1)). It has options to specify preprocessing by `tbl` and/or `neqn`, (see *tbl*(1) and *neqn*(1)), and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for `nroff` and mm are generated, depending on the options selected.

osdd is equivalent to the command mm `-mosd`.

Options

mm recognizes the following *options* and command-line arguments. Any other arguments or options (such as `-rC3`) are passed to `nroff` or to mm, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, mm prints a list of its options.

`-T`*term*	Specifies the type of output terminal; for a list of recognized values for *term*, type `help term2`. If this option is *not* used, mm uses the value of the shell variable **$TERM** from the environment (see *profile*(4) and *environ*(5)) as the value of *term* if **$TERM** is set; otherwise, mm uses `450` as the value of *term*. If several terminal types are specified, the last one is used.
`-12`	Indicates that the document is to be produced in 12-pitch. Can be used when **$TERM** is set to one of `300`, `300s`, `450`, and `1620`. (The pitch switch on the DASI 300 and 300s terminals must be manually set to `12` if this option is used.)
`-c`	Causes mm to invoke *col*(1); note that *col*(1) is invoked automatically by mm unless *term* is one of `300`, `300s`, `450`, `37`, `4000a`, `382`, `4014`, `tek`, `1620`, and `X`.
`-e`	Causes mm to invoke `neqn`.
`-t`	Causes mm to invoke `tbl`.
`-E`	Invokes the `-e` option of `nroff`.
`-y`	Causes mm to use the non-compacted version of the macros (see *mm*(5)). The default is to use the compacted version.

DIAGNOSTICS

mm sends the message `mm: no input file` if none of the arguments is a readable file and mm is not used as a filter.

EXAMPLES

Assuming that the shell variable **$TERM** is set in the environment to `450`, the two command lines below are equivalent:

```
mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3
```

mm reads the standard input when `-` is specified instead of any file names (mentioning other files along with `-` leads to disaster). This option allows mm to be used as a filter, as in this example:

```
cat dws | mm -
```

Hints

- mm invokes `nroff` with the `-h` option. With this option, `nroff` assumes that the terminal has tabs set every 8 character positions.
- Use the `-o`*list* option of `nroff` to specify ranges of pages to be output. Note, however, that mm, if invoked with one or more of the `-e`, `-t`, and `-` options, *together* with the `-o`*list* option of `nroff` may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
- If you use the `-s` option of `nroff` (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The `-s` option of `nroff` does not work with the `-c` option of mm, or

if mm automatically invokes `col` (see `-c` option above and `col(1)`).

- If you specify an incorrect output terminal type, mm produces (often subtle) unpredictable results. However, if you are redirecting output into a file, use the `-T37` option, then use the appropriate terminal filter when actually printing the formatted file.

SEE ALSO

col(1), env(1), nroff(1), tbl(1), profile(4), mm(5), term(5).

mm section in *Text Formatters User's Guide* .

NAME

more, page - file perusal filter for crt viewing

SYNOPSIS

`more` [`-n`] [`-cdflsu`] [*+linenumber*] [`+`*/pattern*] [*name* ...]

`page` [`-n`] [`-cdflsu`] [*+linenumber*] [`+`*/pattern*] [*name* ...]

REMARKS:

`pg` is preferred in some standards and has some added functionality, but does not support character highlighting (see *pg*(1)).

DESCRIPTION

`more` is a filter for examining continuous text, one screenful at a time, on a soft-copy terminal. It is quite similar to `pg`, and is retained primarily for backward compatibility. `more` normally pauses after each screenful, printing `--More--` at the bottom of the screen. To display one more line, press `Return`. To display another screenful, press the space bar. Other possibilities are described later.

`more` and `page` differ only slightly. `more` scrolls the screen upward as it prints the next page. `page` clears the screen and prints a new screenful of text when it prints a new page. Both provide one line of overlap between screenfuls.

name can be a filename or `-`, specifying standard input. `more` processes file arguments in the order given, and does not process standard input if file arguments are present, unless standard input is specifically identified by using the `-` filename argument (see EXAMPLES below).

`more` supports the Basic Regular Expression syntax (see *regexp*(5)).

`more` recognizes the following command line options:

`-`*n*	Use a window size of *n* lines instead of the default (*n* is an integer).
`-c`	Draw each page by beginning at the top of the screen, and erase each line just before drawing on it. This avoids scrolling the screen, making it easier to read while `more` is writing. This option is ignored if the terminal has no clear-to-end-of-line capability.
`-d`	Prompt user with the message `Press space to continue, q to abort` at the end of each screenful. This is useful if `more` is being used as a filter in some setting, such as a training class, where many users might be unsophisticated.
`-f`	Count logical lines, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter can generate escape sequences. These escape sequences contain characters that would ordinarily occupy screen positions, but which do not print when sent to the terminal as part of an escape sequence. Thus `more` might assume lines are longer than they really are, and fold lines erroneously.
`-l`	Do not treat ^L (form feed) specially. If this option is not given, `more` pauses after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.
`-s`	Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.
`-u`	Normally, `more` handles underlining and bold such as produced by `nroff` in a manner appropriate to the particular terminal: if the terminal supports underlining or has a highlighting (usually inverse-video) mode, `more` outputs appropriate escape sequences to enable underlining, else highlighting mode, for underlined information in the source file. If the terminal supports highlighting, `more` uses that mode information that should be printed in boldface type. The `-u` option suppresses this processing, as do the "ul" and "os" terminfo flags.
+linenumber	Start listing at *linenumber*.
`+`*/pattern*	Start listing two lines before the line matching the regular expression *pattern*.

If the program is invoked as *page* instead of `more`, the screen is cleared before printing each screenful (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

`more` uses terminfo descriptor files to determine terminal characteristics and to determine the default window size (see *term*(4)). On a terminal capable of displaying 24 lines, the default window size is 22 lines.

`more` uses the environment variable MORE to preset any flags desired. For example, to view files using the `-c` mode of operation, the shell command sequence

```
MORE='-c' ; export MORE
```

or the *csh* command

```
setenv MORE -c
```

causes all invocations of `more`, including invocations by programs such as *man* and *msgs*, to use this mode. The command sequence that sets up the MORE environment variable is usually placed in the *.profile* or *.cshrc* file.

If `more` is reading from a file, rather than a pipe, a percentage is displayed along with the `--More--` prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences that can be typed when `more` pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1):

i <space>	Display *i* more lines, (or another screenful if no argument is given).
^D	Display 11 more lines (a "scroll"). If *i* is given, the scroll size is set to *i*.
d	Same as ^D (control-D).
i`z`	Same as typing a space except that *i*, if present, becomes the new window size.
i`s`	Skip *i* lines and print a screenful of lines.
i`f`	Skip *i* screenfuls and print a screenful of lines.
`q` or `Q`	Exit from `more`.
`=`	Display the current line number.
`v`	Start up the editor *vi* at the current line.
`h`	Help command; give a description of all the `more` commands.
i / *expr*	Search for the *i*-th occurrence of the regular expression *expr*. If there are fewer than *i* occurrences of *expr* and the input is a file (rather than a pipe), the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters can be used to edit the regular expression. Erasing back past the first column cancels the search command.
i`n`	Search for the *i*-th occurrence of the last regular expression entered.
'	(single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.
`!`*command*	Invoke a shell with *command*. The characters `%` and `!` in *command* are replaced with the current file name and the previous shell command, respectively. If there is no current file name, `%` is not expanded. The sequences `\%` and `\!` are replaced by `%` and `!` respectively.
i`:n`	Skip to the *i*-th next file given in the command line (skips to last file if *n* does not make sense).
i`:p`	Skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, `more` goes back to the beginning of the file. If *i* does not make sense, `more` skips back to the first file. If `more` is not reading from a file, the bell is rung and nothing else happens.

`:f`	Display the current file name and line number.
`:q` or `:Q`	Exit from `more` (same as `q` or `Q`).
`.`	(dot) Repeat the previous command.

The commands take effect immediately; i.e., it is not necessary to press `Return`. Up to the time when the command character itself is given, the line-kill character can be used to cancel the numerical argument being formed. In addition, to redisplay the `--More--` (*xx*%), press the erase character.

The quit key (normally Ctrl-\) can be used at any time when output is being sent to the terminal. `more` stops sending output, and displays the usual `--More--` prompt. One of the above commands can then be entered in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

`more` sets the terminal *noecho* mode so that the output can be continuous. Thus, what you type does not show on the terminal, except for the `/` and `!` commands.

If the standard output is not a teletype, `more` is equivalent to *cat*(1), except that a header is printed before each file (if more than one is specified).

`more` supports the `SIGWINCH` signal, and redraws the screen in response to window size changes.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions.

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters, the classification of characters as printable, and the characters matched by character class expressions in regular expressions.

`LANG` determines the language in which messages are displayed.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `more` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

To view a simple file, use:

```
more filename
```

To preview *nroff* output, use a command resembling:

```
nroff -mm +2 doc.n | more -s
```

If the file contains tables, use:

```
tbl file | nroff -mm | col | more -s
```

To display file `stuff` in a fifteen line-window and convert multiple adjacent blank lines into a single blank line:

```
more -s -15 stuff
```

When mixing standard input and filename arguments, use `-` as follows:

Display `file1` from standard input followed by `file2`:

```
    cat file1 |more - file2
```

In this form, the `-` is missing, and only `file2` is displayed (standard input is ignored):

```
    cat file1 |more file2
```

FILES

`/usr/lib/more.help`	help file
`/usr/lib/terminfo/?/*`	compiled terminal capability data base

VARIABLES

MORE	Default paging mode.

AUTHOR

`more` was developed by the University of California, Berkeley.

SEE ALSO

csh(1), man(1), pg(1), sh(1), term(4), terminfo(4), environ(5), lang(5), regexp(5).

NAME

mt - magnetic tape manipulating program

SYNOPSIS

`mt` [`-t` *tapename*] *command* [*count*]

DESCRIPTION

`mt` is used to give commands to the tape drive. If *tapename* is not specified, the environment variable **TAPE** is used; if **TAPE** is not defined, the default drive is used.

`mt` winds the tape in the requested direction (forward or backward), stopping after the specified *count* EOF marks or records are passed. If *count* is not specified, one is assumed. Each EOF mark counts as one record. When winding backwards, the tape always stops at the BOT marker, regardless of the number remaining in *count*.

`mt` accepts the following *commands*:

eof	Write *count* EOF marks.
fsf	Forward space *count* files.
fsr	Forward space *count* records.
bsf	Backward space *count* files.
bsr	Backward space *count* records.
rew	Rewind tape.
offl	Rewind tape and go offline.
eod	Seek to end of data (DDS and QIC drives only).
smk	Write *count* setmarks (DDS drives only).
fss	Forward space *count* setmarks (DDS drives only).
bss	Backward space *count* setmarks (DDS drives only).

Spacing operations (back or forward space file or record) leave the tape positioned past the object being spaced to in the direction of motion. That is, backspacing a file leaves the the tape positioned before the file mark, forward spacing a file leaves the tape positioned after the file mark. This is consistent with all classical usage on tapes.

FILES

/dev/rmt/*	Raw magnetic tape interface
/dev/rmt/0mn	Default tape interface

WARNINGS

Only raw, no rewind, Berkeley-style devices can be specified.

It is possible to wind the tape beyond the EOT marker and off the end of the reel.

EXAMPLES

Rewind the tape associated with the device file **/dev/rmt/0mn**:

```
mt -t /dev/rmt/0mn rew
```

AUTHOR

`mt` was developed by the University of California, Berkeley.

SEE ALSO

dd(1), mt(7).

NAME

mv - move or rename files and directories

SYNOPSIS

mv [**-f** | **-i**] *file1 new_file*
mv [**-f** | **-i**] *file1* [*file2* ...] *dest_directory*
mv [**-f** | **-i**] *directory1* [*directory2* ...] *dest_directory*

DESCRIPTION

mv moves:

- *file1* to new or existing *new_file,*
- *file1* to existing *dest_directory,*
- Multiple files *file1*, *file2*, ... to existing *dest_directory,*
- directory subtree *directory1*, to new or existing *dest_directory*. or
- multiple directory subtrees *directory1*, *directory2*, ... to new or existing *dest_directory*.

Moving *file1* to *new_file* is used to relocate a file within the file system or to rename a file within a directory. When destination is a directory, one or more files are moved into that directory. If two or more files are moved, the destination must be a directory. When moving a single file to a new file, if *new_file* exists, its contents are destroyed.

If the access permissions of the destination *dest_directory* or existing destination file *new_file* forbid writing, **mv** asks permission to overwrite the file. This is done by printing the mode (see *chmod*(2) and Access Control Lists below), followed by the first letters of the words *yes* and *no* in the current native language, prompting for a response, and reading one line from the standard input. If the response is affirmative and the action is permissible, the operation occurs; if not, the command proceeds to the next source file, if any.

If *file1* is a file and *new_file* is a link to another file with other links, the other links remain and *new_file* becomes a new file. If *file1* is a file with links or a link to a file, the existing file or link remains intact, but the name is changed to *new_file* which may or may not be in the directory where *file1* resided, depending on directory pathnames used in the **mv** command. The last access and modification times of the file or files being moved remain unchanged.

Options

mv recognizes the following options:

-f Perform **mv** commands without prompting for permission. This option is assumed when the standard input is not a terminal.

-i Causes **mv** to write a prompt to standard output before moving a file that would overwrite an existing file. If the response from the standard input is affirmative, the file is moved if permissions allow the move.

Access Control Lists (ACLs)

If optional ACL entries are associated with *new_file*, **mv** displays a plus sign (+) after the access mode when asking permission to overwrite the file.

If *new_file* is a new file, it inherits the access control list of *file1*, altered to reflect any difference in ownership between the two files (see *acl*(5)).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG and **LC_CTYPE** determine the local language equivalent of y (for yes/no queries).

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **mv** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Rename a file in the current directory:

```
mv old_filename new_filename
```

Rename a directory in the current directory:

```
mv old_dirname new_dirname
```

Rename a file in the current directory whose name starts with a non-printing control character, **-**, *****, or some other character that is special to the shell (extra care may be required, depending on the situation):

```
mv ./ bad_filename new_filename
mv ./? bad_filename new_filename
mv ./* bad_filename new_filename
```

Move directory *sourcedir* and its contents to a new location (*targetdir*) in the file system (upon completion, a subdirectory named *sourcedir* resides in directory *targetdir*):

```
mv sourcedir targetdir
```

Move all files and directories (including links) in current directory to a new location underneath *targetdir*:

```
mv * targetdir
```

Move all files and directories (including links) in *sourcedir* to a new location underneath *targetdir* (*sourcedir* and *targetdir* are in separate directory paths):

```
mv sourcedir/* targetdir
```

WARNINGS

If *file1* and *new_file* exist on different file systems, **mv** copies the file and deletes the original. In this case the mover becomes the owner and any linking relationship with other files is lost. **mv** cannot carry hard links across file systems.

mv cannot be used to perform the following operations:

- Rename either the current working directory or its parent directory using the **.** or **..** notation,
- Rename a directory to a new name identical to the name of a file contained in the same parent directory.

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

DEPENDENCIES

NFS Access control lists of networked files are summarized (as returned in *st_mode* by *stat*(2)), but not copied to the new file. When using **mv** on such files, a **+** is not printed after the mode value when asking for permission to overwrite a file.

AUTHOR

mv was developed by AT&T, the University of California, Berkeley and HP.

SEE ALSO

cp(1), cpio(1), ln(1), rm(1), link(1M), lstat(2), readlink(2), stat(2), symlink(2), symlink(4), acl(5).

STANDARDS CONFORMANCE

mv: SVID2, XPG2, XPG3, POSIX.2

NAME

neqn - format mathematical text for nroff

SYNOPSIS

`neqn` [`-d`*xy*] [`-s` *n*] [`-f` *n*] [`-p` *n*] [*file* ...]

DESCRIPTION

`neqn` is a preprocessor for `nroff` for typesetting mathematical text on typewriter-like terminals. *nroff*(1) Usage is almost always of the following two forms or equivalent:

```
neqn  files|nroff|col
tbl  files|neqn|nroff|
```

If no files are specified (or if `-` is specified as the last argument), `neqn` reads from the standard input. A line beginning with `.EQ` marks the start of an equation. The end of an equation is marked by a line beginning with `.EN`. Neither of these lines is altered, which means that they can be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as `neqn` input. Delimiters can be set to characters *x* and *y* with the command-line argument `-d`*xy* or (more commonly) with `delim` *xy* between `.EQ` and `.EN`. The left and right delimiters can be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by `delim off` (see WARNINGS below). All text that is neither between delimiters nor between `.EQ` and `.EN` is passed through untouched.

Tokens within `neqn` are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces `{ }` are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces can be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and Superscripts

Subscripts and superscripts are produced using `sub` and `sup` as follows:

Source Text	**Result**
`x sub j`	x_j
`a sub k sup 2`	a_k^2
`e sup {x sup 2 + y sup 2}`	$e^{x^2+y^2}$

Fractions

Fractions are produced by using `over` :

Source Text	**Result**
`a over b`	$\frac{a}{b}$

Square Roots

`sqrt` produces square roots:

Source Text	**Result**
`1 over sqrt {ax sup 2+bx+c}`	$\frac{1}{\sqrt{ax^2+bx+c}}$

Upper and Lower Limits

The keywords `from` and `to` specify lower and upper limits:

Source Text	**Result**
`lim from {n -> inf } sum from 0 to n x sub i`	$\lim_{n \to \infty} \sum_0^n x_i$

Brackets and Braces

Left and right brackets, braces, and such, of proper height are made with `left` and `right`:

Source Text	Result
`left [ {x sup 2 + y sup 2} over alpha right ] ~=~ 1`	$\left[\frac{x^2+y^2}{\alpha} \right] = 1$

Legal characters after `left` and `right` are braces, brackets, bars, `c` and `f` for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket). A `left` *thing* need not have a matching `right` *thing*.

Vertical Piles

Vertical piles of *things* are made with `pile`, `lpile`, `cpile`, and `rpile`:

Source Text	Result
`pile {a above b above c}`	$\begin{matrix} a \\ b \\ c \end{matrix}$

Piles can have arbitrary numbers of elements; `lpile` left justifies, `pile` and `cpile` center (but with different vertical spacing), and `rpile` right justifies.

Matrices and Determinants

Matrices are made with `matrix`:

Source Text	Result
`left	{ matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } } } right |`	$\begin{vmatrix} x_i & 1 \\ y_2 & 2 \end{vmatrix}$

In addition, there is `rcol` for a right-justified column.

Diacritical Marks

Diacritical marks are made with `dot`, `dotdot`, `hat`, `tilde`, `bar`, `vec`, `dyad`, and `under`:

Source Text	Result
`x dot = f(t) bar`	$\dot{x} = \overline{f(t)}$
`y dotdot bar ~=~ n under`	$\bar{\ddot{y}} = \underline{n}$
`x vec ~=~ y dyad`	$\vec{x} = \overleftrightarrow{y}$

Point sizes and fonts can be changed with `size` *n* or `size` ±*n*, `roman`, `italic`, `bold`, and `font` *n*. Point sizes and fonts can be changed globally in a document by `gsize` *n* and `gfont` *n*, or by the command-line arguments `-s`*n* and `-f`*n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this can be changed by the command-line argument `-p`*n*.

Successive display arguments can be lined up. Place `mark` before the desired lineup point in the first equation; place `lineup` at the place that is to line up vertically in subsequent equations.

Shorthand Forms

Shorthand forms can be defined or existing keywords redefined with `define`:

`define` *thing % replacement %*

defines a new token called *thing* that is replaced by *replacement* whenever it appears thereafter. The % can be any character that does not occur in *replacement*.

Keywords such as `sum` ($\sum$), `int` ($\int$), `inf` (∞), and shorthands such as `>=` ($\geq$), `!=` ($\neq$), and `->` ($\rightarrow$) are recognized. Greek letters are spelled out in uppercase or lowercase as desired, as in `alpha` (α) or `GAMMA` (Γ). Mathematical words such as `sin`, `cos`, and `log` are made Roman automatically. `nroff` four-character escapes such as `\(dd` (‡) and `\(bu` (•) can be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with `nroff` when other methods fail. Details are given in the manuals cited below.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, **neqn** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.
Also see WARNINGS under *nroff*(1).

Good practice dictates that if a delimiter is specified in a file, the `delim off` directive should be included at the end of the file to prevent undesirable behavior when processing multiple files where a subsequent file may contain the delimiter character as part of regular text.

To properly display equations on terminal screens and other devices that do not support reverse line feeds, `nroff` output should be piped through *col*(1).

SEE ALSO

mm(1), nroff(1), tbl(1), mm(5).

Typesetting Mathematics-User's Guide, by B. W. Kernighan and L. L. Cherry.
New Graphic Symbols for EQN and NEQN, by C. Scrocca.

NAME

netstat - show network status

SYNOPSIS

netstat [**-Aan**] [**-f** *address_family*] [*system*] [*core*]
netstat [**-R**] [*system*] [*core*]
netstat [**-himnrs**] [**-f** *address_family*] [*system*] [*core*]
netstat [**-n**] [**-I** *interface*] *interval* [*system*] [*core*]

DESCRIPTION

netstat symbolically displays the contents of various network-related data structures. Output format varies according to options selected. The **netstat** command takes one of the three forms shown above:

- The first form of the command displays a list of active sockets for each protocol.
- The second form presents the contents of one of the other network data structures according to the option selected.
- The third form causes **netstat** to display updated packet traffic data on configured network interfaces. The display is updated at each *interval*.

Options are interpreted as follows:

-A Use the default display to show the address of any protocol control blocks associated with sockets. This option is used for debugging (does not show the X.25 programmatic access control blocks.)

-R Lists all socket names in the socket registry for NetIPC applications. **netstat -R** returns only NetIPC information, not BSD IPC ("Berkeley Sockets") information.

-a Use the default display to show the state of all sockets. Normally sockets used by server processes are not shown. (This option does not show the state of X.25 programmatic access sockets.)

-h Show the state of the IMP host table.

-i Show the state of auto-configured interfaces (interfaces statically configured into a system, but not located at boot time are not shown).

-I *interface* Show information about this interface only. This option is used with an *interval* as described below.

-m Show statistics recorded by memory management routines (the network manages a private pool of memory buffers).

-n Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically). This option can be used with any available display format.

-s Show per-protocol statistics.

-r Show the routing tables. If **-s** is also present, show routing statistics instead.

-f *address_family*
Limit statistics or address control block reports to those of the specified *address_family*. The following address families are recognized: **inet** for **AF_INET**, and **unix** for **AF_UNIX**.

The arguments, *system* and *core* allow substitutes for the defaults **/hp-ux** and **/dev/kmem**.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form *host*.*port* or *network*.*port* if a socket's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically using **gethostbyname()** and **getnetbyname()**, respectively (see *gethostbyname*(3N) and *getnetbyname*(3N)). If a symbolic name for an address is unknown, or if the **-n** option is specified, the address is displayed numerically according to the address family. For more information regarding the Internet "dot format", refer to *inet*(3N). Unspecified or "wildcard" addresses and ports appear as *****.

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit (mtu) are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route (`U` if `up`), whether the route is to a gateway (`G`), and whether the route was created dynamically by a redirect (`D`). Direct routes are created for each interface attached to the local host. The gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection-oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry identifies which network interface was used for the route.

When **`netstat`** is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. To replace the primary interface with another interface, use the **`-I`** option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

DEPENDENCIES

X.25:

`-A` and **`-a`** options do not list X.25 programmatic access information.

`-R` option is not supported over the X.25 link.

AUTHOR

`netstat` was developed by the University of California, Berkeley.

SEE ALSO

hosts(4), networks(4), gethostbyname(3N), getnetbyname(3N), protocols(4), services(4).

NAME

newform - change or reformat a text file

SYNOPSIS

`newform` [`-i` *tabspec*] [`-o` *tabspec*] [`-l` *n*] [`-b` *n*] [`-e` *n*] [`-c` *char*] [`-p` *n*] [`-a` *n*] [`-f`] [`-s`] [*files*]

DESCRIPTION

`newform` reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for `-s`, command line options can appear in any order, can be repeated, and can be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences such as `-e 15 -l 60` yield results different from `-l 60 -e 15`. Options are applied to all *files* on the command line.

Options

`newform` recognizes the following options:

`-i`*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* can be `--`, in which `newform` assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to `-8`. A *tabspec* of `-0` expects no tabs; if any are found, they are treated as `-1`.

`-o`*tabspec* Output tab specification: replaces spaces with tabs, according to the tab specifications given. The tab specifications are the same as for `-i`*tabspec*. If no *tabspec* is given, *tabspec* defaults to `-8`. A *tabspec* of `-0` means that no spaces will be converted to tabs on output.

`-l`*n* Set the effective line length to *n* characters. If *n* is not entered, `-l` defaults to 72. The default line length without the `-l` option is 80 characters. Note that tabs and backspaces are treated as single characters (use `-i` to expand tabs to spaces).

`-b`*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see `-l`*n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when `-b` with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -l1 -b7 file-name
```

The `-l1` must be used to set the effective line length shorter than any existing line in the file so that the `-b` option is activated.

`-e`*n*
Same as `-b`*n* except that characters are truncated from the end of the line.

`-c`*k*
Change the prefix/append character to *k*. Default character for *k* is a space.

`-p`*n*
Prefix *n* characters (see `-c`*k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

`-a`*n*
Same as `-p`*n* except characters are appended to the end of a line.

`-f`
Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* `-o` option. If no `-o` option is specified, the line which is printed contains the default specification of `-8`.

`-s`
Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a `*` and any characters to the right of it are discarded. The first tab is always

discarded.

An error message and program exit occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform -s -i -l -a -e file-name
```

RETURN VALUE

`newform` returns one of the following values upon completion:

0 No errors encountered.

1 An error occurred.

DIAGNOSTICS

All diagnostics are fatal.

`usage:` ...
: `newform` was called with a bad option.

`not -s format`
: There was no tab on one line.

`can't open file`
: Self-explanatory.

`internal line too long`
: A line exceeds 512 characters after being expanded in the internal work buffer.

`tabspec in error`
: A tab specification is incorrectly formatted, or specified tab stops are not ascending.

`tabspec indirection illegal`
: A *tabspec* read from a file (or standard input) must not contain a *tabspec* referencing another file (or standard input).

WARNINGS

`newform` normally only keeps track of physical characters; however, for the `-i` and `-o` options, `newform` keeps track of backspaces in order to line up tabs in the appropriate logical columns.

`newform` does not prompt the user if a *tabspec* is to be read from the standard input (by use of `-i--` or `-o--`).

If the `-f` option is used, and the last `-o` option specified was `-o--`, and was preceded by either a `-o--` or a `-i--`, the tab specification format line will be incorrect.

SEE ALSO

fspec(4), csplit(1), tabs(1).

NAME

newgrp - log in to a new group

SYNOPSIS

`newgrp` [`-`] [*group*]

DESCRIPTION

`newgrp` changes a user's group identification without changing the user ID, subject to the following conditions:

- User is a member of the new *group*, or
- If user is not a member of *group*, *group* has a password that is known to user. (If user is not a member of *group* and there is no group password, `newgroup` fails.)

The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs.

`newgrp` always gives the user a new shell which replaces the current shell (without creating a new process), regardless of whether `newgrp` terminates successfully or as a result of an error condition (such as an unknown group). The new shell is either:

- The shell specified for that user in `/etc/passwd`, or
- `/bin/sh` if the shell field in `/etc/passwd` is empty for that user.

Exported variables retain their values after invoking `newgrp`, but all unexported variables are either reset to their default value or set to null. Environment variables such as `PS1`, `PS2`, `PATH`, `MAIL`, and HOME are reset to default values unless they were previously exported by the system or explicitly exported by the user. For example, assume a user has redefined the primary prompt string `PS1` so that it no longer has the default value `$`, but `PS1` has not been exported. `newgrp`, whether successful or not, sets `PS1` to the default prompt string `$` as a result of the changed shell. To preserve existing variable names and values when starting new shells, use the built-in `export` command described in the *ksh*(1) or *sh*(1) manual entry (*csh*(1) does not have a built-in `export` command).

When `newgrp` is executed without any arguments, the group identification changes back to the group specified in the user's password file entry, and a new shell is created, again without starting a new process.

If the first argument to `newgrp` is `-`, the user's `.profile` file (or `.cshrc` if `csh` is specified for the user in `/etc/passwd`) is executed when the new shell is started, thus changing the environment to what would be expected if the user actually logged in again.

If the user is not listed as a member of *group* in file `/etc/group` and the group has a password, a password is required before `newgrp` can change the effective group ID.

`newgrp` fails if the user does not qualify as a group member for any of the following reasons:

- User cannot be found in the password file, either by login name or real user id, or
- User was not a member of the new group and:
  - *group* has a password which the user failed to provide, or
  - *group* has no password.

Since no new process is created when the new shell is started, exiting from the new shell has the same effect as exiting the shell from which `newgrp` was executed.

EXTERNAL INFLUENCES

International Code Set Support

Characters from the 7-bit USASCII code set are supported in group names (see *ascii*(5)).

DIAGNOSTICS

`Sorry:`	User does not qualify as a group member.
`Unknown group:`	Group name does not exist in `/etc/group`.
`Permission denied:`	If a password is required, it must come from a teletype port.

`You have no shell:`
Standard input is a non-tty file, causing the **`exec`** of the new shell to fail.

WARNINGS

There is no convenient way to enter a password into **`/etc/group`**.

Use of group passwords is not recommended because they, by their very nature, encourage poor security practices. Group passwords may be eliminated in future HP-UX releases.

Any shell variables that are not exported are lost.

FILES

`/etc/group`	system group file
`/etc/passwd`	system password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

STANDARDS CONFORMANCE

`newgrp`: SVID2, XPG2, XPG3

NAME

newmail - notify users of new mail in mailboxes

SYNOPSIS

newmail [**-i** *interval*] [**-w**] [*file-spec* ...]

DESCRIPTION

newmail monitors the user's incoming or specified mailbox. Without any options, **newmail** runs in the background at a default interval of 60 seconds to monitor the user's incoming mailbox.

Options

newmail recognizes the following options:

-i *interval*	This option changes the time interval between mailbox checks to the value specified, in seconds.
-w	This option runs the program within a window where it has a more succinct output format and also runs in foreground rather than background.

The basic operation is that the program checks the incoming or specified mailbox each *interval* seconds and lists any new mail that has arrived in any of the mailboxes, indicating the sender name, and the subject of the message.

The following message is produced when the program is initially started:

`Newmail started: folder` *foldername*

Each entry displayed can be in a number of different formats, depending on the mode of the program and the status of the message. If **newmail** is running in a window (**-w** option), the output resembles:

`New mail from` *sender name - subject of message*

or

`PRIORITY mail from` *sender name - subject of message*

where *sender name* is either the name of the person sending it, if available (the ARPA **From:** line), or some other brief indication of origin. If there is no subject, the message **(No Subject Specified)** is displayed. If **newmail** is checking more than one mailbox, output lines are prefixed by the *folder-name* or prefix string specified by *file-spec*.

If the message is a "priority" message (meaning that it has a field in the header **Priority:**), the line will read **PRIORITY mail** instead of **New mail**.

When running **newmail** without the **-w** option, the output format is modified so that it is suitable for display on an already active screen. **newmail** messages are prefixed with a pair of pointer characters as follows:

`>> New mail from` *sender name - subject of message*

or

`>> PRIORITY mail from` *sender name - subject of message*

In this case, output lines are also prefixed when monitoring multiple mailboxes.

file-spec is made up of two components: the *folder name* and the *prefix-string*, the latter of which can always be omitted. The format is *foldername=prefix-string*. Metacharacters such as +, =, and % which indicate the folder directory in the **elm** mailer (see *elm*(1)) is available to specify the folder name (see EXAMPLES).

newmail runs until the user logs out or explicitly kills it, and can internally reset itself if the mailbox shrinks in size and then grows again.

EXAMPLES

Here are some example invocations:

Check incoming mailbox every 60 seconds:

```
newmail
```

Check incoming mailbox every 15 seconds for new messages from `joe` or `root`.

```
newmail -i 15 joe root
```

Monitor the incoming mailbox for new messages from user `mary` and the folder in your *maildir* called `postmaster`. Prefix all messages from `mary` with the string `Mary`, and messages from `postmaster` (see *elm*(1)) with `POBOX`. Also, monitor folder `/tmp/mbox`:

```
newmail "mary=Mary" +postmaster=POBOX /tmp/mbox
```

WARNING

If the interval is set to less than 10 seconds, `newmail` issues a warning message that such short intervals are not recommended.

interval must be less than 2^{32} seconds because `newmail` uses `sleep()` (see *sleep*(3C)).

AUTHOR

`newmail` was developed by HP.

NAME

news - print news items

SYNOPSIS

news [**-a**] [**-n**] [**-s**] [*items*]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, **news** prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. **news** stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

Options

news recognizes the following options:

-a Print all items, regardless of currency. The stored time is not changed.

-n Report the names of the current items without printing their contents, and without changing the stored time.

-s Report how many current items exist without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If an interrupt is typed during the printing of a news item, printing stops and the next item is started. Another interrupt within one second of the first causes the program to terminate.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

FILES

/usr/news/*
$HOME/.news_time
/etc/profile

SEE ALSO

mail(1), profile(4), environ(5).

STANDARDS CONFORMANCE

news: SVID2, XPG2

NAME

nice - run a command at non-default priority

SYNOPSIS

`nice` [- *priority_change*] *command* [*command_args*]

DESCRIPTION

`nice` (name derived from being "nice" to other system users by running large programs at lower priority) executes *command* at a non-default CPU scheduling priority. Command-line arguments are as follows:

priority_change	Specifies the difference between the system nice value (relative priority) of the current (or parent) process and the actual system nice value at which *command* is to be run. An unsigned value increases the system nice value for *command*, causing it to run at lower priority; a negative value requires super-user privileges, and assigns a lower (higher priority) system nice value to *command*. If the value of *priority_change* would result in a system nice value outside the range 0 through 39, the corresponding limit value of 0 or 39 is used instead. Note that a positive *priority_change* (lower priority) has a single `-` option character before the numeric value; a negative (higher priority) *priority_change* has two; the option character followed by the minus sign: `--`. If *priority_change* is not specified, the default value of `10` (lower priority) is used.
command	Program, HP-UX command, user shell script, etc. to be executed at non-default priority. *command* can be run as a foreground or background process. If *command* is run as a background process, any nice *priority_change* made by the shell (`ksh` executes all background processes via `nice -4`) is in addition to that specified in the `nice` command line.
command_args	Any arguments recognized by *command*.

Process Priorities

All processes have an associated **system nice value** which is used to compute the instantaneous-priority of the process when it is scheduled to run. Normally, all processes inherit the system nice value of their parent process when they are spawned. The shell (`sh`, `csh`, `ksh`, etc.) can create a child process with a different priority than the current shell process by spawning the child process via the `nice` command. If the *priority_change* value is unsigned (positive), the child process is nicer (lower in priority) relative to the parent. If the *priority_change* value is negative, the child process runs at a higher priority with a greater share of available system resources. To spawn a higher priority child process, the parent process must be owned by a user who has the necessary privileges.

At boot-up, the system starts the `init` process at a system nice value of 20 (system default). On most systems, all processes (down to the login shells) inherit this priority. Starting from their individual login shell processes, users can alter the system nice value of descendent processes to as much as 39, or, with appropriate privileges, as little as 0. A system nice value of 0 establishes an extremely high priority, whereas a value of 39 indicates a very low priority.

Ordinary users can only increase the system nice value of any child process relative to the current process; i.e., *priority_change* must be a positive (unsigned) value, resulting in a lower priority. To start a child process at a lower system nice value (higher priority) than the current process, the user must have the appropriate privileges listed above, regardless of the relative nice-priority value desired.

For example, using the command

```
nice ksh
```

from a login shell whose current nice value is 20 spawns a sub-shell with a system nice value of 30. Attempting to use

```
nice --2 ksh
```

to spawn another subshell whose system nice value would be 28, is rejected (unless the user has the necessary privileges), even though the resulting system nice value would be less than the priority of the original login shell process.

The system nice value for current processes is listed under the `NI` column produced by the `ps -l` command (see *ps*(1)).

Background Processes

Foreground processes are run at same system nice value as the parent shell. Background processes spawned by `ksh` run at the equivalent of a `nice -4` by default. If a background process is started via `nice` from `ksh`, any *priority_change* specified in the `nice` command is added to default `nice -4`. Thus the command `nice 12` *command* `&` runs at a system nice value of 36 if executed from `ksh`.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`nice` returns the value returned by *command*.

EXAMPLES

The following examples assume the current process is running with a system nice value of 20 and `nice` is executed from the Korn shell (see *ksh*(1)).

Run a program named `prog` in the current directory at the default *priority_change* of 10 (system nice value of 30):

`nice ./prog` *prog_args*

From Korn shell (see *ksh*(1)), run the same program in the background using a system nice value of 36 (*priority_change*=12):

`nice -12 ./prog` *prog_args* `&`

As a user with appropriate privileges, run `prog` as a foreground process with a system nice value of 6:

`nice --14 ./prog` *prog_args*

WARNINGS

`csh` has a built-in `nice` command with syntax that is different from the syntax of the standard system `nice` command, and which behaves as follows:

`nice` +*priority_change*
: Increases the system nice value for the current shell by *priority_change*.

`nice` -*priority_change*
: Decreases the system nice value for the current shell by *priority_change*.

`nice`
: Sets the system nice value for the current shell to 39, making it very difficult to get any response from a busy system.

`nice` *command*
: Executes *command* in a sub-shell with a default *priority_change* of 4 from the system nice value for the current shell process.

`nice` *priority_change command ...*
: Executes *command* in a sub-shell at the specified *priority_change* from the system nice value for the current shell process.

SEE ALSO

nohup(1), renice(1), nice(2).

STANDARDS CONFORMANCE

`nice`: SVID2

NAME

nl - line numbering filter

SYNOPSIS

`nl` [`-b`*type*] [`-h`*type*] [`-f`*type*] [`-p`] [`-v`*start#*] [`-i`*incr*] [`-s`*sep*] [`-w`*width*] [`-n`*format*] [`-l`*num*] [`-d`*delim*] [*file*]

DESCRIPTION

`nl` reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

`nl` views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	Start of
\:\:\:	header
\:\:	body
\:	footer

Unless told otherwise, `nl` assumes the text being read is in a single logical page body.

Command options can appear in any order and can be intermingled with an optional file name. Only one file can be named. `nl` recognizes the following options:

`-b`*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meanings are:

`a`	number all lines;
`t`	number lines with printable text only;
`n`	no line numbering;
`p`*string*	number only lines that contain the regular expression specified in *string*. Basic Regular Expression syntax is supported (see *regexp*(5)).

The default *type* for logical page body is `t` (text lines numbered).

`-h`*type*
Same as `-b`*type* except for header. Default *type* for logical page header is `n` (no lines numbered).

`-f`*type*
Same as `-b`*type* except for footer. Default for logical page footer is `n` (no lines numbered).

`-p`
Do not restart numbering at logical page delimiters.

`-v`*start#*
start# is the initial value used to number logical page lines. Default is `1`.

`-i`*incr*
incr is the increment value used to number logical page lines. Default is `1`.

`-s`*sep*
sep is the character or characters used in separating the line number and the corresponding text line. Default *sep* is a tab.

`-w`*width*
width is the number of character columns to be used for the line number. Default *width* is `6`.

`-n`*format*
format is the line numbering format. Recognized values are:

`ln`	left justified, leading zeroes suppressed;
`rn`	right justified, leading zeroes suppressed;

rz right justified, leading zeroes kept.

Default *format* is **rn** (right justified).

-l*num*
num is the number of consecutive blank lines to be treated and numbered as a single line. For example, **-l3** results in every third adjacent blank line being numbered if the appropriate **-ha**, **-ba**, and/or **-fa** option is set. Default is **1**.

-d*xx*
The delimiter characters specifying the start of a logical page section can be changed from the default characters (**\:**) to two user-specified characters. If only one character is entered, the second character remains the default character (**:**). No space should appear between the **-d** and the delimiter characters. To define a backslash as the delimiter, use two backslashes.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the characters matched by character class expressions in regular expressions.

If **LC_COLLATE** or **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **nl** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Number **file1** starting at line number 10, using an increment of ten. The logical page delimiters are **!** and +:

```
nl -v10 -i10 -d!+ file1
```

SEE ALSO

pr(1), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

nl: SVID2, XPG2, XPG3

NAME

nljust - justify lines, left or right, for printing

SYNOPSIS

`nljust` [`-aclnt`] [`-e` *seq*] [`-j` *just*] [`-m` *mode*] [`-o` *order*] [`-r` *margin*] [`-w` *width*] [`-x` *ck*] [*file* ...]

DESCRIPTION

`nljust` formats for printing data written in languages with a right-to-left orientation. It is designed to be used with the `pr` and the `lp` commands (see *pr*(1) and *lp*(1)).

`nljust` reads the concatenation of input files (or standard input if none are given) and produces on standard output a right-to-left formatted version of its input. If `-` appears as an input file name, `nljust` reads standard input at that point. Use `--` to delimit the end of options.

`nljust` formats input files for all languages that are read from right to left. For languages that have a left-to-right orientation, the command merely copies input files to standard output.

Options

`nljust` recognizes the following options:

`-a`	Justify data for all languages, including those having a left-to-right text orientation. By default only right-to-left language data is justified. For all other languages, input files are directly copied to standard output.
`-c`	Select enhanced printer shapes for some Arabic characters. With this option, two-character combinations of laam and alif are replaced by a single character.
`-e` *seq*	Use *seq* as the escape sequence to select the primary character set. This escape sequence is used by languages that have too many characters to be accommodated by ASCII in a single 256-character set. In these cases, the *seq* escape sequence can be used to select the non-ASCII character set. The *escape* character itself (0x1b) is not given on the command line. Hewlett-Packard escape sequences are used by default.
`-j` *just*	If *just* is `l`, left justify print lines. If *just* is `r`, right-justify print lines starting from the (designated or default) print width column. The default is right justification.
`-l`	Replace leading spaces with alternative spaces. Some right-to-left character sets have a non-ASCII or alternative space. This option can be useful when filtering `pr -n` output (see *pr*(1)). With right justification, the `-l` option causes line numbers to be placed immediately to the right of the tab character. Without the `-l` option, right justification causes line numbers to be placed at the print-width column. By default, leading spaces are not replaced by alternative spaces.
`-m` *mode*	Indicate *mode* of any file to be formatted. Mode refers to the text orientation of the file when it was created (see *hpnls*(5) for more details). If *mode* is `l`, assume Latin mode. If *mode* is `n`, assume non-Latin mode. By default, mode information is obtained from the **LANGOPTS** environment variable.
`-n`	Do not terminate lines containing printable characters with a new-line. By default, print lines are terminated by new-lines.
`-o` *order*	Indicate data *order* of any file to be formatted. The text orientation of a file can affect the way its data is arranged (see *hpnls*(5) for more details). If *order* is `k`, assume keyboard order. If *order* is `s`, assume screen order. By default, order information is obtained from the **LANGOPTS** environment variable.
`-t`	Truncate print lines that do not fit the designated or default line length. Print lines are folded (that is, wrapped to next line) by default.
`-x` *ck*	Expand input tabs to column positions *k*+1, *2**k+*1, 3**k+*1,* etc. Tab characters in the input are expanded to the appropriate number of spaces. If *k* is 0 or is omitted, default tab settings at every eighth position is assumed. If *cd* (any non-digit character) is given, it is treated as the input tab character. The default for *c* is the tab character. `nljust` always expands input tabs. This option provides a way to change the tab character and setting. If this option is specified, at least one of the parameters *c* or *k* must be given.

`-r` *margin* Designate a number as the print *margin*. The print margin is the column where truncation or folding takes place. The print margin determines how many characters appear on a single line and can never exceed the print width. The print margin is relative to the justification. If the print margin is 80, folding or truncation occurs at column 80 starting from the right during a right justification. Similarly, folding or truncation occurs at column 80 starting from the left during a left justification. By default, the print margin is set to column 80.

`-w` *width* Designates a number as the print *width*. The print width is the maximum number of columns in the print line. Print width determines the start of text during a right justification. The larger the print width, the further to the right the text will start. By default, an 80-column print width is used.

EXTERNAL INFLUENCES

Environment Variables

The **LANGOPTS** environment variable determines the mode and order of the file. The syntax of **LANGOPTS** is [*mode*][*_order*]. *mode* describes the mode of a file where `l` represents Latin mode and `n` represents non-Latin mode. Non-Latin mode is assumed for values other than `l` and `n`. *order* describes the data order of a file where `k` is keyboard and `s` is screen. Keyboard order is assumed for values other than `k` and `s`. Mode and order information in **LANGOPTS** can be overridden from the command line.

The **LC_ALL** environment variable determines the direction of a language (left-to-right or right-to-left) and whether context analysis of characters is necessary.

The **LC_NUMERIC** environment variable determines whether a language has alternative numbers.

The **LANG** environment variable determines the language in which messages are displayed.

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Right justify `file1` on a 132-column printer with a print margin at column 80 (the default):

```
nljust -w 132 file1 | lp
```

Right justify `pr` output of `file2` with line numbers on a 132-column printer with a print margin at column 132:

```
pr -n file2 | nljust -w 132 -r 132 | lp
```

WARNINGS

If `pr` with line numbers (`-n` option) is piped to `nljust`, the separator character must be a tab (0x09).

It is the user's responsibility to ensure that the **LANGOPTS** environment variable accurately reflects the status of the file.

Mode and justification must be consistent. Only non-Latin-mode files can be right justified in a meaningful way. Similarly, only Latin-mode files can be safely left justified. If mode and justification do not match, the results are undefined.

If present, alternative numbers always have a left-to-right orientation.

AUTHOR

`nljust` was developed by HP.

SEE ALSO

forder(1), lp(1), pr(1), strord(3C), hpnls(5).

NAME

nlsinfo - display native language support information

SYNOPSIS

nlsinfo [**-acfhilmnstC**] [**-d** *n*] [**-e** *n*] [**-o** *n*] [**-r** *n1 n2*] [**-L** *language*]

Remarks:

nlsinfo is provided for historical reasons only. Use **locale** instead (see *locale*(1)).

DESCRIPTION

nlsinfo displays a list of installed Native Language Support (NLS) international environment languages, and displays information contained within the runtime locale associated with these languages (see *setlocale*(3C)). Commands and routines modified for NLS operation (see EXTERNAL INFLUENCES section in applicable manual entries) use the information contained within locales to process and display data according to the local language and customs of a user (see *hpnls*(5)). Unless overridden by the **-L** option, the runtime locale is determined by the locale category environment variables (those that begin with **LC_**) and the **LANG** environment variable.

By default, non-printable characters are displayed as 2-digit hexadecimal numbers unless they are included within strings, in which case the hexadecimal numbers are preceded by **\x**.

Options

The following options are recognized:

-l	Display the list of installed international environment languages.
-a	Display the **LC_ALL** locale category which contains all parts of the selected locale. Equivalent to specifying options **-csntmh**.
-c	Display the **LC_CTYPE** locale category that contains character classification and conversion information (see *ctype*(3C), *conv*(3C), and *nl_tools_16*(3C)).
-s	Display the **LC_COLLATE** locale category that contains collating (sorting) information (see *strcoll*(3C) and *strxfrm*(3C)).
-n	Display the **LC_NUMERIC** locale category which contains information regarding the radix character and thousands separator.
-t	Display the **LC_TIME** locale category which contains such information as the format of the date and time and the names of the days of the week (see *strftime*(3C)).
-m	Display the **LC_MONETARY** locale category which contains information for the formatting of monetary quantities.
-h	Display all other local custom information not included in the **LC_NUMERIC**, **LC_TIME**, and **LC_MONETARY** locale categories.
-i	Display a list of abbreviated "from" and "to" code set names used by **iconv** the code set conversion utility (see *iconv*(1)). Each "from" and "to" code set name pair represents a conversion supported by **iconv**.
-C	Display all the characters defined in the selected NLS environment's code set. The list of characters may be quite long for some Asian NLS environments.

If none of the options **l**, **a**, **c**, **s**, **n**, **t**, **m**, **h**, **i**, or **C** is specified, **-l** is used as a default.

-f	Place a form feed character before all category display headings.
-o *n*	Display non-printable characters as numbers using the base specified by *n*. If *n* is **o**, display non-printable characters as 3-digit octal numbers. If *n* is **x**, display non-printable characters as 2-digit hex numbers. If non-printable characters are included within strings, the octal or hex numbers are preceded by a backslash (\).
-e *n*	Display all characters (including printable characters) as hexadecimal or octal numbers. If *n* is **o**, display characters as 3-digit octal numbers. If *n* is **x**, display characters as 2-digit hex numbers. If characters are included within strings, the octal or hex numbers are preceded by a backslash (\).

`-d` *n* Display *n* lines between headings. By default 16 lines are displayed between headings.

`-r` *n1 n2* Display information only about element number *n1* through element number *n2* of the requested table(s), rather than all elements of each table. The parameters *n1* and *n2* can be hexadecimal (preceded by `0x`), octal (preceded by `0`), or decimal numbers. These parameters can also be characters, in which case each character is mapped to its value in the code set of the user's current locale (not the locale, if any, specified via the `-L` option). Characters that are digits should be quoted with the backslash character (\) to prevent them from being interpreted as numbers.

`-L` *language* Display information about the locale associated with *language* instead of the locale specified by the setlocale category and **LANG** environment variables. The parameter *language* should be one of the language names from the output of `nlsinfo -l` (ell) (the valid language names are also given in *lang*(5)).

EXTERNAL INFLUENCES

Environment Variables

The `LC_CTYPE` environment variable determines how option argument characters are mapped to a value in the code set.

The **LANG** environment variable determines the language in which messages and headings are displayed.

International Code Set Support

Single- and multi-byte-character code sets are supported with the exception of multi-byte-character file names.

WARNINGS

The format and content of the display of a particular table may change if the underlying NLS implementation is changed to support new features. Applications should not use the information displayed by *nlsinfo* as the basis for their operation, but rather should use the interface routines provided in LIBC (see *conv*(3C), *ctype*(3C), *langinfo*(3C), *nl_tools_16*(3C), *setlocale*(3C), *string*(3C) and *buildlang*(1M)).

AUTHOR

`nlsinfo` was developed by HP.

FILES

`/usr/lib/nls/config`
`/usr/lib/nls/$LANG/locale.def`

SEE ALSO

buildlang(1M), iconv(1), conv(3C), ctype(3C), langinfo(3C), nl_tools_16(3C), setlocale(3C), string(3C), environ(5), hpnls(5), lang(5).

NAME
nm - print name list of common object file.

SYNOPSIS
nm [*options*] [*file*]

REMARKS
This is a generic entry for a machine-dependent program. A specific entry is provided for each version. Refer to manual entry *nm_300*(1) for information about **nm** on Series 300 and 400 systems or *nm_800*(1) for information about **nm** on Series 700 and 800 systems.

SEE ALSO
nm_300(1), nm_800(1), crt0(3), end(3C).

EXTERNAL INFLUENCES
International Code Set Support
Single-byte character code sets are supported.

STANDARDS CONFORMANCE
nm: SVID2, XPG2, XPG3

NAME

nm - print name list of common object file

SYNOPSIS

nm [-degnoprsux] [*file* ...]

DESCRIPTION

nm prints the name list (symbol table) of each common object *file* in the argument list. If an argument is an archive, a listing for each common object file in the archive is produced. If no *file* is given, the symbols in a.out are listed.

Each symbol name is preceded by its value (zero if undefined) and one of the letters:

U	undefined,
A	absolute,
T	text segment symbol,
D	data segment symbol, or
B	bss segment symbol.

If the symbol is local (non-external), the type letter is in lowercase. If the symbol is a secondary definition, the type letter is followed by the letter S. The output is sorted alphabetically. Common symbols are indicated by a type of U and a value indicating the size.

Options

nm recognizes the following options:

-d	Print numeric values in decimal.
-e	Print only external (global) symbols. Same as -g.
-g	Print only global (external) symbols. Same as -e.
-n	Sort numerically rather than alphabetically.
-o	Precede each output line with the file or archive element name, rather than printing the file or archive element name only once. This option can be used to make piping to grep more meaningful (see *grep*(1)).
-p	Do not sort; print in symbol-table order.
-r	Sort in reverse order.
-s	Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next highest value). This difference is the value printed. This flag turns on -g and -n and turns off -u and -p.
-u	Print only undefined symbols.
-x	Print numeric values in hexadecimal (default).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

SEE ALSO

ar(1), crt0(3), end(3C), a.out_300(4), a.out_800(4), ar(4).

NAME

nm - print name list of object file

SYNOPSIS

nm [-hnoprxuvTV] file ...

DESCRIPTION

nm displays the symbol table of each object file *file*. *file* can be a relocatable or absolute common object file, or it can be an archive of relocatable or absolute common object files. For each symbol, at least the following information is printed:

Name	The name of the symbol.
Value	Its value expressed as an offset or an address depending on its storage class.
Scope	The scope of the symbol (undefined, static, sdef, or external). The "sdef" scope indicates an external symbol that is flagged as a secondary definition.
Type	The type of the symbol (code, data, common, absolute, etc.).
Subspace	The subspace to which the symbol belongs.

Options

The output of nm can be controlled using the following options:

-e	Print only external and static symbols. This is the normal behavior, so this option is ignored.
-f	Produce full output. This is the normal behavior, so this option is ignored.
-h	Do not display the output header data.
-n	Sort symbols by *name*, in ascending collation order, before they are printed (see Environment Variables below).
-o	Print the *value* and *size* of a symbol in octal instead of decimal.
-p	Produce easily parsable, terse output. Each symbol *name* is preceded by its value (blanks if undefined) and one of the letters U (undefined), A (absolute), T (text symbol), D (data symbol), B (bss symbol), or C (common symbol). If the symbol is local (non-external), the type letter is in lowercase. If the symbol is a secondary definition, the type letter is followed by the letter S.
-r	Prefix each output line with the name of the object file or archive.
-u	Print undefined symbols only.
-v	Sort symbols by *value* before they are printed.
-x	Print the *value* and *size* of a symbol in hexadecimal instead of decimal.
-T	By default, nm prints the entire name of the symbols listed. Since object files can have symbol names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The -T option causes nm to truncate every name that would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.
-V	Print the version of the nm command executing on the standard error output.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating order output by the -n option.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, nm behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Display which object files have undefined references for the symbol `foo`:

```
nm -rup *.o | grep foo
```

Display which object files have a definition for the text symbol foo:

```
nm -rp *.o | awk '{ if ($3 == "T" && $4 == "foo") { print $0 } }'
```

SEE ALSO

cc(1), ld(1), crt0(3), end(3C).

STANDARDS CONFORMANCE

nm: SVID2, XPG2, XPG3

NAME

nodename - assign a network node name or determine current network node name

SYNOPSIS

`nodename` [*name*]

DESCRIPTION

`nodename` used without any parameters returns the network node name for the system on which it is executed, and can be executed by any normal user. `nodename` used with a *name* parameter assigns *name* as the network node name for the system, and can be executed *only* by users with appropriate privileges.

name is the Network Services (NS) node name chosen for the system, and should be of the form *node*.*domain*.*organization*. *node*, *domain*, and *organization* can each be up to 16 alphanumeric, underscore, (_) or hyphen (-) characters; the first character must be alphabetic. *domain* and *organization* are arbitrary labels, and may be useful for grouping nodes and collections of nodes. However, they do not indicate anything about the structure of the network. (The *node*.*domain*.*organization* notation discussed here is not related in any way to Internet dot notation.)

The name assigned by *nodename* is used by the LAN diagnostic, *rlb*, the Probe proxy server, the NS/9000 NFT service, and any user-defined NetIPC application.

By convention, the *node* field of the node name should duplicate the *host name* field in the `/etc/hosts` file.

WARNINGS

Many types of networking services are supported on HP-UX, each of which uses a different assigned system name and naming convention. To ensure predictable system behavior, it is essential that system names (also called host names or node names) be assigned in such a manner that they do not create conflicts when the various networking facilities interact with each other.

The system does not rely on a single system name in a specific location, partly because different services use dissimilar name formats as explained below. System names are assigned by using the `uname -S`, `hostname`, and `nodename` *name* commands. In addition, the system name used in the HP Clustered Environment (called the **cnode name**) is assigned in the cluster configuration file `/etc/clusterconf`. System names are assigned as follows:

Nodename	Command/File	Format	Used By
NetIPC name	`nodename` *name*	*foo*[.*a*[.*b*]]	NS Services and NetIPC
Internet name	`hostname` *name*	*foo*[.*x*.*y*.*z* ...]	ARPA and NFS Services
UUCP name	`uname -S` *name*	*foo*	UUCP and related programs
cnode name	`/etc/clusterconf`	*foo*	Cluster server and clients

where *foo* represents the assigned system name (it is *strongly* recommended that *foo* be identical for all commands and locations) and the optional .*x*.*y*.*z* or .*a*.*b* follow the specified notation for the particular ARPA/NFS or NS/NetIPC environment.

Internet names are also frequently called hostnames or domain names (not to be confused with NFS domain names). Refer to *hostname*(5) for more information about Internet naming conventions.

Whenever the system name is changed in any file or by use of any of the above commands, it should also be changed in all other locations as well. Other files or commands in addition to those above (such as `/usr/lib/uucp/Permissions` if used to circumvent `uname`, for example) may contain or alter system names. To ensure correct operation, they should also use the same system name.

System names are normally assigned by the `/etc/rc` script at start-up, and should not be altered elsewhere.

DIAGNOSTICS

Messages indicate a user attempted to assign a node name without having appropriate privileges, in which case no new node name is assigned.

AUTHOR

`nodename` was developed by HP.

SEE ALSO

dscopy(1), hostname(1), proxy(1M), rlb(1M), ipcgetnodename(2), ipcsetnodename(2), hostname(5).

NAME
nohup - run a command immune to hangups, logouts, and quits

SYNOPSIS
`nohup` command [arguments]

DESCRIPTION
`nohup` executes *command* with hangups and quits ignored. If output is not redirected by the user, both standard output and standard error are sent to `nohup.out`. If `nohup.out` is not writable in the current directory, output is redirected to `$HOME/nohup.out`; otherwise, `nohup` fails.

If output from `nohup` is redirected to a terminal, or is not redirected at all, the output is sent to `nohup.out`.

EXTERNAL INFLUENCES
International Code Set Support
Single- and multi-byte character code sets are supported.

EXAMPLES
It is frequently desirable to apply `nohup` to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell script. To run the script using `nohup`:

```
nohup sh file
```

`nohup` features apply to the entire contents of *file*. If the shell script *file* is to be executed often, the need to type `sh` can be eliminated by setting execute permission on *file*. The script can also be run in the background with interrupts ignored (see *sh*(1)):

```
nohup file &
```

file typically contains normal keyboard command sequences that one would want to continue running in case the terminal disconnects, such as:

```
tbl ofile | eqn | nroff > nfile
```

WARNINGS
Be careful to place punctuation properly. For example, in the command form:

```
nohup command1; command2
```

`nohup` applies only to *command1*. To correct the problem, use the command form:

```
nohup (command1; command2)
```

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio -o <list >/dev/rmt/1m&
```

whereas

```
nohup cpio -o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file `errors`.

SEE ALSO
chmod(1), nice(1), sh(1), signal(5).

STANDARDS CONFORMANCE
`nohup`: SVID2, XPG2, XPG3, POSIX.2

NAME

nroff - format text

SYNOPSIS

`nroff` [*options*] *file* ...

DESCRIPTION

`nroff` is a text formatting program that interprets source text contained in *file* and prepares it for printing on typewriter-like devices and line printers. If *file* name is `-` or not specified, standard input is used as source text.

If the file contains plain text with no formatter requests, `nroff` uses default line lengths and page dimensions to produce readable output, outputting a blank line for each blank line encountered in the input, and filling and adjusting text to both margins. `nroff` ignores any lines in the source text that begin with a period (`.`) but are not valid `nroff` formatter requests.

`nroff` formatting capabilities are described in the tutorial cited below.

Source File Preparation

Document source file preparation is usually easier when text is coded using macro packages such as *mm*(1) which provide a high-level interface for headings, page footers, lists, and other features, rather than coding the file with inherently low-level `nroff` requests.

Options

`nroff` recognizes the following command-line *options*, which can appear in any order but must appear before the *file* argument:

`-o`*list*	Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *n*`-`*m* means pages *n* through *m*; an initial `-`*n* means from the beginning to page *n*; and a final *n*`-` means from *n* to the end. (See **WARNINGS** below.)
`-n`*n*	Number first generated page *n*.
`-s`*n*	Stop every *n* pages. `nroff` halts *after* every *n* pages (default *n*=1) to allow paper loading or changing, and resumes upon receipt of a line-feed or new-line (new-lines do not work in pipelines, such as with `mm`). When `nroff` halts between pages, an ASCII BEL is sent to the terminal.
`-r`*aN*	Set register *a* (which must have a one-character name) to *N*.
`-i`	Read standard input after *files* are exhausted.
`-q`	Invoke the simultaneous input-output mode of the `.rd` request.
`-z`	Print only messages generated by `.tm` (terminal message) requests.
`-m`*name*	Precede the input *files* with the non-compiled (ASCII text) macro file `/usr/lib/nls/`*LANG*`/tmac/tmac.`*name*, where *LANG* is the value of the **LANG** environment variable. If **LANG** is not set or `/usr/lib/nls/`*LANG*`/tmac/tmac.`*name* does not exist `/usr/lib/tmac/tmac.`*name* is used instead.
`-c`*name*	Precede the input *files* with the compiled macro files `/usr/lib/macros/cmp.[nt].d.`*name* and `/usr/lib/macros/ucmp.[nt].`*name*.
`-k`*name*	Compile the macros used in this invocation of `nroff`, placing the output in `d.`*name* in the current directory.
`-T`*name*	Prepare output for specified terminal. Known *name*s are `37` for the (default) TELETYPE Model 37 terminal, `tn300` for the GE TermiNet 300 (or any terminal without half-line capability), `300s` for the DASI 300s, `300` for the DASI 300, `450` for the DASI 450, `lp` for a (generic) ASCII line printer, `382` for the DTC-382, `4000A` for the Trendata 4000A, `832` for the Anderson Jacobson 832, `X` for a (generic) EBCDIC printer, `2631` for the Hewlett-Packard 2631 line printer, `klp` for a (generic) 16-bit character printer having ratio of 2 to 3 in 8-bit and 16-bit character width, and `lj` for Hewlett Packard PCL3 and newer laser printers.

`-e`	Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.
`-h`	Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every eight nominal character widths.
`-u`*n*	Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text as single and/or multi-byte characters.

LANG is used to determine the search path for the `-m` option. **LANG** also determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `nroff` behaves as if all internationalization variables are set to "C". See *environ*(5).

EXAMPLES

The following command prints the first five pages of the document whose `nroff` source file is *filename*:

`nroff -o-5` *filename*

Note that there should not be a space between the `o` and the `-` or the `-` and the `5`.

To print only pages 1, 3, and 4 type:

`nroff -o1,3,4` *filename*

WARNINGS

When `nroff` is used with the `-o`*list* option inside a pipeline, it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

FILES

`/usr/lib/macros/*`	standard macro files
`/usr/lib/term/*`	terminal driving tables for `nroff`
`/usr/lib/suftab`	suffix hyphenation tables
`/usr/lib/tmac/tmac.*`	standard macro files and pointers

SEE ALSO

col(1), mm(1), neqn(1), soelim(1), ul(1), man(5).
`nroff/troff` tutorial in the *Text Formatting Users Guide*.

NAME

nslookup - query name servers interactively

SYNOPSIS

nslookup [*-option* ...] *host-to-find* [*server*]
nslookup [*-option* ...] [- [*server*]]

DESCRIPTION

nslookup is a program to query Internet domain name servers. If a name server is not configured, **nslookup** uses NIS if it is configured. Otherwise the local host table, **/etc/hosts**, is used.

nslookup has two modes: interactive and non-interactive. Interactive mode allows the user to query a name server for information about various hosts and domains, or print a list of hosts in the domain. Non-interactive mode is used to to query a name server for information about one host or domain. When using NIS or the host table, only name and address lookups are possible.

Interactive mode is entered in the following cases:

- No arguments are given.
- The first argument is a hyphen (-). The optional second argument is a host name or Internet address of a name server. If there is no server running on that host, NIS or the host table is used.

Non-interactive mode is used when the name of the host to be looked up is given as the first argument. The optional second argument is a host name or Internet address of a name server.

The options listed under the **set** command below can be specified in the **.nslookuprc** file in the user's home directory if they are listed one per line. Options can also be specified on the command line if they precede the arguments and are prefixed with a hyphen. For example, to change the default query type to host information, and the initial timeout to 10 seconds, type:

```
nslookup -query=hinfo -timeout=10
```

Interactive Commands

Commands can be interrupted at any time by using the interrupt character. To exit, type a Ctrl-D (EOF) or type **exit**. To treat a built-in command as a host name, precede it with an escape character (\). When using NIS or the host table, only host names and Internet addresses are allowed as commands. An unrecognized command is interpreted as a host name.

host [*server*]
Look up information for *host* using the current default server or using *server* if specified. If *host* is an Internet address and the query type is A or PTR, the name of the host is returned. If *host* is a name and does not have a trailing period, one or more domains are appended to the name (this behavior depends on the state of the **set** options **domain**, **srchlist**, **defname**, and **search**). Answers from a name server's cache are labelled "non-authoritative."

server *domain*
lserver *domain*
Change the default server to *domain*. **lserver** uses the initial server to look up information about *domain* while **server** uses the current default server.

root
Changes the default server to the server for the root of the domain name space. Currently, the host **ns.nic.ddn.mil** is used (this command is a synonym for the **lserver**ns.nic.ddn.mil**)**. The name of the root server can be changed with the **set root** command.

finger [*name*] [> *filename*]
finger [*name*] [>> *filename*]
Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and returned address information (see the **set querytype=A** command). *name* is optional. > and >> can be used to redirect output in the usual manner.

ls [*option*] *domain* [> *filename*]
ls [*option*] *domain* [>> *filename*]
List the information available for *domain*, optionally creating or appending to *filename*. The default output contains host names and their Internet addresses. *option* can be one of

the following:

`-t` *querytype*	lists all records of the specified type (see *querytype* below).
`-a`	lists aliases of hosts in the domain (synonym for `-t CNAME`).
`-d`	lists all records for the domain (synonym for `-t ANY`).
`-h`	lists CPU and operating system information for the domain (synonym for `-t HINFO`).
`-s`	lists well-known services of hosts in the domain (synonym for `-t WKS`).

When output is directed to a file, `#` characters are printed for every 50 records received from the server.

`view` *filename*
Sorts and lists the output of previous `ls` command(s) using `more` (see *more*(1)).

`help`
`?` Prints a brief summary of commands.

`exit`
Exits the program.

`set` *keyword*[=*value*]
This command is used to change state information that affects the lookups. Valid keywords are:

`all` Prints the current values of the various options to `set`. Information about the current default server and host is also printed.

`cl[ass]`=*value*
Change the query class to one of:

`IN`	the Internet class.
`CHAOS`	the Chaos class.
`HESIOD`	the MIT Athena Hesiod class.
`ANY`	wildcard (any of the above).

The class specifies the protocol group of the information.
(Default = `IN`)

`[no]deb[ug]`
Turn debugging mode on. More information is printed about the packet sent to the server and the resulting answer.
(Default = nodebug)

`[no]d2`
Turn exhaustive debugging mode on. Essentially all fields of every packet are printed.
(Default = nod2)

`[no]def[name]`
If set, append the default domain name to a single-component lookup request (i.e., one that does not contain a period character).
(Default = `defname`)

`do[main]`=*name*
Change the default domain name to *name*. The default domain name is appended to a lookup request, depending on the state of the `defname` and `search` options. The domain search list contains the parents of the default domain if it has at least two components in its name. For example, if the default domain is `CC.Berkeley.EDU`, the search list is `CC.Berkeley.EDU` and `Berkeley.EDU`. Use the `set srchlist` command to specify a different list. Use the `set all` command to display the list.
(Default = value from hostname, `/etc/resolv.conf` or `LOCALDOMAIN`)

`[no]ig[nore]`
Ignore truncation errors.
(Default = noignore)

`q[uerytype]=`*value*
`ty[pe]=`*value*
Change the type of information returned from a query to one of:

A	Host's Internet address
ANY	All types of data
CNAME	Canonical name for an alias
GID	Group ID
HINFO	Host CPU and operating system type
MB	Mailbox domain name
MG	Mail group member
MINFO	Mailbox or mail list information
MR	Mail rename domain name
MX	Mail exchanger
NS	Name server for the named zone
PTR	Host name if the query is an Internet address, otherwise the pointer to other information.
SOA	Start of authority record
TXT	Text information
UID	User ID
UINFO	User information
WKS	Well-known service description

`po[rt]=`*value*
Change the default TCP/UDP name server port to *value*.
(Default = 53)

`[no]rec[urse]`
Tell the name server to query other servers if it does not have the information.
(Default = recurse)

`ret[ry]=`*number*
Set the number of retries to *number*. When a reply to a request is not received within a certain amount of time (changed with `set timeout`), the timeout period is doubled and the request is resent. The retry value controls how many times a request is resent before giving up.
(Default = 4)

`ro[ot]=`*host*
Change the name of the root server to *host*. This affects the `root` command.
(Default = `ns.nic.ddn.mil`)

[no]sea[rch]
If the lookup request contains at least one period but doesn't end with a trailing period, append the domain names in the domain search list to the request until an answer is received. See *hostname*(5).
(Default = `search`)

`srchl[ist]=`*name1/name2/...*
Change the default domain name to *name1* and the domain search list to *name1*, *name2*, etc. A maximum of 6 names separated by slashes (/) can be specified. For example,

```
set srchlist=lcs.MIT.EDU/ai.MIT.EDU/MIT.EDU
```

sets the domain to `lcs.MIT.EDU` and the search list to the three names. This command overrides the default domain name and search list of the `set domain` command. Use the `set all` command to display the list.
(Default = value based on hostname, `/etc/resolv.conf` or LOCALDOMAIN)

`t[imeout]`=*number*
Change the initial timeout interval for waiting for a reply to *number* seconds. Each retry doubles the timeout period.
(Default = 5 seconds)

`[no]v[c]`
Always use a virtual circuit when sending requests to the server.
(Default = `novc`)

DIAGNOSTICS

If the lookup request was not successful, an error message is printed. Possible errors are:

`Time-out`
The server did not respond to a request after a certain amount of time (changed with `set timeout`=*value*) and a certain number of retries (changed with `set retry`=*value*).

`No response from server`
No name server is running on the server machine.

`No records`
The server does not have resource records of the current query type for the host, although the host name is valid. The query type is specified with the `set querytype` command.

`Non-existent domain`
The host or domain name does not exist.

`Connection refused`
`Network is unreachable`
The connection to the name server could not be made at the present time.

`Server failure`
The name server found an internal inconsistency in its database and could not return a valid answer.

`Refused`
The name server refused to service the request.

`Format error`
The name server found that the request packet was not in the proper format.

AUTHOR

`nslookup` was developed by the University of California, Berkeley.

FILES

`/etc/resolv.conf`	initial domain name and name server addresses
`$HOME/.nslookuprc`	user's initial options

SEE ALSO

named(1M), resolver(3N), resolver(4), hostname(5),

RFC1034, RFC1035

NAME

od, xd - octal and hexadecimal dump

SYNOPSIS

`od` [`-v`] [`-A` *address_base*] [`-j` *skip*] [`-N` *count*] [`-t` *type_string*] ... [*file* ...]

`xd` [`-v`] [`-A` *address_base*] [`-j` *skip*] [`-N` *count*] [`-t` *type_string*] ... [*file* ...]

The following pre-POSIX usage is also supported.

`od` [`-bcdosx`] [*file*] [[+][`0x`]*offset*[`.`][`b`]]
`xd` [`-bcdosx`] [*file*] [[+][`0x`]*offset*[`.`][`b`]]

DESCRIPTION

`od` and C xd concatenate one or more input *file*s and write their contents to standard output in a user-specified format. If *file* is not specified, the standard input is used.

Options and Arguments

`od` and `xd` recognize the following options and command-line arguments:

`-A` *address_base* Specify the input offset base. *address_base* is a single character that defines which format the offset base is written in:

`d`	Decimal format.
`o`	Octal format.
`x`	Hexadecimal format.
`n`	Do not write the offset.

`-j` *skip*

Jump over *skip* bytes from the beginning of the input. `od` seeks past the first *skip* bytes in the concatenated input files. If the combined input is not at least *skip* bytes long, `od` writes a diagnostic message to standard error and exits with a non-zero exit status. By default, *skip* is interpreted as a *decimal* number. If *skip* has a leading `0x` or `0X`, it is interpreted as a *hexadecimal* number; a leading `0` indicates that *skip* is an *octal* number.

If the value of *skip* is followed by a `b`, `k`, or `m`, it is interpreted as a multiple of 512, 1024, or 1048576, respectively.

`-N` *count*

Format no more than *count* bytes of input.

By default, *count* is interpreted as a *decimal* number. A leading `0x` or `0X` indicates that *count* is a *hexadecimal* number; a leading `0` identifies an `octal` value.

If *count* bytes of input are not available (after successfully skipping if `-j`*skip* is specified), the input that is available is formatted.

`-t` *type_string*

type_string is a string defining the types to be used when writing the input data.

The string can contain any of the following type-specification characters:

`a`	named character ,
`c`	character ,
`d`	signed decimal ,
`f`	floating point ,
`o`	octal ,
`u`	unsigned decimal ,
`x`	hexadecimal ,

Type specification characters `d`, `f`, `o`, `u`, and `x` can be followed by an optional *unsigned decimal* integer specifying the number of bytes to be transformed by each instance of the output type, or by an optional `C`, `S`, `I`, or `L` indicating that the conversion should be applied to an item of type *char*, *short*, *int*, or *long*, respectively.

Type specification character `f` can be followed by an optional `F`, `D`, or `L` indicating that the conversion should be applied to an item of type *float*, *double*, or *long double*, respectively.

Multiple types can be concatenated within the same *type_string* and multiple `-t` options can be specified. Output lines are written for each type specified in the order in which the type specification characters appear.

-v
Write all input data. Without the `-v` option, any number of groups of output lines, that would be identical to the immediately preceding group of output lines (except for the byte offsets), are replaced with a line containing only an asterisk (*).

file
Pathname of one or more input files to be processed. If *file* is not specified, the standard input is used.

Input files can be any file type.

DESCRIPTION OF PRE-POSIX USAGE

`od` and `xd` dump *file* in one or more formats as selected by the first argument. If the first argument is missing, the default is `-o` for `od`; `-x` for `xd`. An offset field is inserted at the beginning of each line. For `od`, the offset is in octal, for `xd`, the offset is in hexadecimal.

Options

`od` and `xd` recognize the following format options:

`-b`	Interpret bytes in octal (hexadecimal).
`-c`	Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=`\0`, backspace=`\b`, form-feed=`\f`, new-line=`\n`, return=`\r`, tab=`\t`; others appear as 3-digit octal numbers.
`-d`	Interpret 16-bit words in decimal.
`-o`	Interpret 16-bit words in octal.
`-s`	Interpret 16-bit words in signed decimal.
`-x`	Interpret 16-bit words in hexadecimal.

file specifies which file is to be dumped. If *file* is not specified, the standard input is used.

offset specifies the offset in the file where dumping is to commence, and is normally interpreted as octal bytes. Interpretation can be altered as follows:

- *offset* must be preceded by + if the file argument is omitted.
- *offset* preceded by **0x** is interpreted in hexadecimal.
- *offset* followed by `.` is interpreted in decimal.
- *offset* followed by `b` is interpreted in blocks of 512 bytes.

Dumping continues until end-of-file.

EXAMPLES

Write hexadecimal bytes and the corresponding octal values to the standard output in blocks of 16 bytes in one line, by transforming the data from the input file `file1`:

```
od -tx1oC file1
```

The following commands write one line each of the types *character*, *signed decimal integer*, and *float*, in the order given, transforming 100 bytes of data starting from fifteenth byte offset in the file `file1`:

```
od -j14 -N100 -tc -tdfF file1

od -j0xe -N100 -tcd4fF file1
```

Write one line each of the types *unsigned integer*, *named character*, and *long double*, with the offsets written in hexadecimal and forcing a write, even on lines that are identical to the immediately preceding group of output lines:

```
od -v -Ax -tuafL file1
```

WARNINGS

When the output format is of floating-point type; i.e., when using the `-t fD`, `-t fL`, or `-t f` options:

- If the input bytes cannot be transformed into a valid floating point number, a floating point exception might occur. In that case, the output is printed as a string containing some non-numeric characters and program execution continues.
- When the number of input bytes used for transformation is set to 1 with the type specifier characters `d`, `o`, `u`, or `x`, only the least-significant seven bits of each byte are used.
- When one or more of the `-A`, `-j`, `-N`, or `-t` options is specified, an operand starting with the first character as a plus-sign (+) or the first character as numeric is interpreted as a file name.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the range of printable characters.

LANG determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `od` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported. Multi-byte data is displayed as multi-byte values.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occured.

SEE ALSO

adb(1).

STANDARDS CONFORMANCE

`od`: SVID2, XPG2, XPG3

NAME

on - execute command on remote host with environment similar to local

SYNOPSIS

`on` [`-i` | `-n`] [`-d`] *host* [*command* [*argument*] ...]

DESCRIPTION

`on` executes a command on a remote host, using an environment similar to that of the invoking user where:

host	specifies the name of the host on which to execute the command.
command	specifies the command to execute on *host*

If *command* is not specified, `on` starts a shell on *host*. *argument* ... is a list of arguments for *command*.

The user's environment variables are copied to the remote host, and the file system containing the user's current working directory is NFS mounted on the remote host (see *nfs*(7)). The command is executed on the remote host in the user's current working directory.

Commands using relative path names that reference file system objects within the user's current working file system have the same behavior as running the command on the client. The behavior of commands using relative path names that cross the file system boundary or commands using absolute path names depends on the organization of the remote host's file system.

Implicit and explicit use of environment variables may also cause a command's behavior to be dependent on the organization of the remote host's file system. For example, the `$PATH` environment variable usually contains absolute path names.

Standard input, output and error of the remote command are connected to the appropriate file descriptors on the client.

The remote execution daemon (`rexd`) does not allow `root` to execute a remote command.

The signals `SIGINT`, `SIGTERM`, and `SIGQUIT` are propagated to the remote command. `SIGTSTP` and `SIGSTOP` are ignored by the remote command. All other signals are delivered to the `on` command.

In order to execute a remote command, the remote host must be configured to execute `rexd` (see *rexd*(1M)).

Options

`on` recognizes the following options:

`-i` Interactive mode. This option is required for commands that must communicate with a terminal such as `vi`, `ksh`, or `more`. Terminal mode changes are propagated to the `rexd` server. The standard input for an interactive `on` command must be a tty device. The `-i` and `-n` options are mutually exclusive.

`-d` Debug mode. Print diagnostic messages during startup of the `on` command. These messages are useful for detecting configuration problems if the `on` command to a specific host is failing.

`-n` No input mode. This option causes the remote command to get end-of-file (EOF) when it reads from standard input, instead of connecting the standard input of the `on` command to the standard input of the remote command. The `-n` option is required when running commands in the background. The `-n` and `-i` options are mutually exclusive.

DIAGNOSTICS

`on: unknown host` *host*
The host name *host* was not found in the hosts database.

`on: cannot connect to server on` *host*
The host *host* is down, unreachable on the network, or not running `rexd`.

`on: can't find` *current_dir*
A problem occurred trying to find the user's current working directory (*current_dir*).

`on: can't locate mount point for` *current_dir*
A problem occurred trying to determine the mount point of the user's current working directory (*current_dir*).

`on: standard input (stdin) is not a tty`
The standard input (stdin) of the on command with the `-i` option is not a tty device.

on *server*`:` `rexd:` *message*
Errors that occur on the server *server* are propagated back to the client. These messages are documented in the DIAGNOSTICS section of *rexd*(1M).

AUTHOR

on was developed by Sun Microsystems, Inc.

SEE ALSO

exports(4), rexd(1M).

NAME

pack, pcat, unpack - compress and expand files

SYNOPSIS

`pack [-] [-f]` *name* ...

`pcat` *name* ...

`unpack` *name* ...

DESCRIPTION

`pack` attempts to store the specified files in a compressed form. Wherever possible, each input file *name* is replaced by a packed file *name*`.z` with the same ownership, modes, and access and modification times. The `-f` option forces packing of *name*. This is useful for causing an entire directory to be packed even if some of the files do not benefit. If `pack` is successful, *name* is removed. Packed files can be restored to their original form using `unpack` or `pcat`.

`pack` uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the `-` argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of `-` in place of *name* cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each `.z` file, it is usually not worthwhile to pack files smaller than three blocks unless the character frequency distribution is very skewed such as in printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

`pack` returns a value that is the number of files that it failed to compress.

No packing occurs if:

- The file appears to be already packed.
- The file name has more than 12 characters and the file system is configured as a short filename system.
- The file has links.
- The file is a directory.
- The file cannot be opened.
- The file is empty.
- No disk storage blocks will be saved by packing.
- A file called *name*`.z` already exists.
- The `.z` file cannot be created.
- An I/O error occurred during processing.

On short-filename systems, the last segment of the file name must contain no more than 12 characters to allow space for the appended `.z` extension. Directories cannot be compressed.

`pcat` does for packed files what *cat*(1) does for ordinary files, except that `pcat` cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*`.z` use:

```
pcat name.z
```

or simply:

```
pcat name
```

To make an unpacked copy (named *file*) of a packed file named *name*`.z` without destroying *name*`.z`) use the command:

```
pcat name >file
```

`pcat` returns the number of files it was unable to unpack. Failure may occur if:

- The file name (exclusive of the `.z`) has more than 12 characters;

- The file cannot be opened;
- The file does not appear to have been created by *pack*.

unpack expands files created by pack. For each file *name* specified in the command, a search is made for a file called *name*.z (or just *name* if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the reasons given for pcat, as well as for the following:

- A file with the "unpacked" name already exists;
- The unpacked file cannot be created.

Access Control Lists (ACLs)

pack retains all entries in a file's access control list when compressing and expanding it (see *acl*(5)).

DEPENDENCIES

NFS

Optional access control list entries of networked files are summarized (as returned in st_mode by stat(), but not copied to the new file (see *stat*(2)).

SEE ALSO

cat(1), compact(1), compress(1), acl(5).

STANDARDS CONFORMANCE

pack: SVID2, XPG2, XPG3

pcat: SVID2, XPG2, XPG3

unpack: SVID2, XPG2, XPG3

(Requires Optional X.25 Software)

NAME

padem - Pad emulation for X.25/9000 interface with PAD support

SYNOPSIS

`padem` [`-p` *profid*] [`-l` *level* [`-f` *filename*]] [`-v`] [`-g` *nn*] [`-r`] [`-u` *abc*] *address* | *symad*
`padem` [`-p` *profid*] [`-i` *programmatic_interface_name*] [`-l` *level* [`-f` *filename*]] [`-v`]

DESCRIPTION

`padem` is used to connect the local terminal with a remote host, whose X121 address is *address* or whose symbolic name is *symad*, by means of the HP X.25/9000 interface. If *address* or *symad* is specified, `padem` makes a connection with the remote host directly, along with the associated facility fields, and enters the *DATA TRANSFER* state. Otherwise, `padem` enters the *PAD COMMAND* state, indicated by its prompt **`X28>`** (provided X.3 parameter 6 is set to 5).

With minor exceptions (see CCITT X.3, X.28, X.29 RECOMMENDATIONS), `padem` implements all recommendations, as defined by CCITT and thus gives full PAD support for outgoing calls. `padem` operates in three modes:

- Data transfer mode, entered either at program invocation time (see SYNOPSIS above) or after the virtual circuit (VC) has been established (see Commands below),
- Command mode without connection, entered at program invocation,
- Command mode with connection which happens when the Escape-From-Data-Transfer-Mode byte is entered during the data transfer mode.

In data transfer mode, typed text is sent to the remote host, and text received from remote host is displayed (unless X.3 parameter 8 is set to 1). To issue a `padem` command signal when in data transfer mode, command mode must be re-entered by using the Escape From Data Transfer Mode key. In command mode, `padem` receives X.28 PAD commands from users (see X28 PAD Command Set below.)

If the VC is cleared at any time, either by the remote host or by the X.25 network, `padem` returns from data transfer mode back to command mode.

Options

`padem` supports the following options:

`-p` *profid*	*profid* is a profile identifier that will be loaded when `padem` is started. This profile is called the initial profile. The profile describes the X.3 configuration for the login session. If it is missing, the profile is taken from the **`/etc/x25/x29hosts`** file. X.3 parameter profiles are stored in file **`/etc/x25/x3config`**.
`-i` *programmatic_interface_name*	*programmatic_interface_name* is the name of the X.25 card through which communication is to be initialized. If this option is missing, the programmatic interface name specified by **name** in **`/etc/x25/x29hosts`** is used. The default programmatic interface name is the name for card **`/dev/x25_0`** if it is not present in the command line or in **`/etc/x25/x29hosts`**.
`-g` *nn*	Specifies the closed user group. It can be 0,1 or 2 digits.
`-r`	Reverse charging is requested.
`-u` *abc*	Specifies call user data. abc is a character string no longer than 12.
`-l` *level*	Logging severity level, can be 0,1,2,3. Default is 0.
`-f` *filename*	*filename* is the user specified filename for logging messages. The logfiles reside in the user's home directory; the full pathname is **$HOME/***filename***.***xxxxxx*. If this option is missing, `padem` uses the default logfile name **$HOME** **`/plog.`***xxxxxx*, where *xxxxxx* is the process ID running the `padem` session.
`-v`	Enable verbose output.

If an option is specified on the command line and is also defined in **`/etc/x25/x29hosts`** for padem configuration, the value entered on the command line overrides the one entered in **`/etc/x25/x29hosts`**.

Configuration

padem configuration can be controlled by managing two configuration files, /etc/x25/x29hosts and /etc/x25/x3config.

/etc/x25/x29hosts defines the configuration for padem for each remote host. A padem configuration entry is identified by the keyword pad_em followed by information on the remote host system beginning with { and ending with }. The information between the open and close brackets specifies the X.121 address of the remote host, the X.25 programmatic interface name to be used for call set-up, the reverse charge option for call set-up, the logging level for login sessions to the host system, and the profile of X.3 parameters to be used for terminal input/output control. A typical padem entry in /etc/x25/x29hosts resembles:

```
pad_em {
name                hptndxk0
remote_x121         4085551111
reverse_charge      enable
profile             0
logging             3
}
```

/etc/x25/x29hosts also defines the X.121 address and symbolic address mapping. This configuration is identified by the host_table entry in the file. Each line starts with remote DTE symbolic name followed by the the remote DTE's X.121 address. host_table{} is searched when users enter symbolic names, either on the command line or in the padem command mode. An typical host_table entry resembles:

```
host_table {
gale                4085551111
tornado             4085551113
typhoon             4085551115
}
```

/etc/x25/x3config provides the initial X.3 parameters for padem sessions. It contains the profile ID, the X.3 parameters, and their values for the PAD emulation session between the user's terminal and the remote system. The initial local X.3 parameters are loaded from the default profile id 0 defined in CCITT if the user does not specify a profile ID. Here is an example of X.3 configuration with profile ID 0:

```
hp_profile : 0 {
     1      1
     2      1
     3    127
     4      0
     5      1
     6      5
     7     21
     8      0
     9      0
     10     0
     11    14
     12     1
     13     0
     14     0
     15     1
     16     8
     17    24
     18     0
     19     1
     20     0
     21     0
     22     0
}
```

If **padem** is started with command line option parameters, the option given on the command line takes precedence over corresponding parameters in **/etc/x25/x29hosts**. If no options are specified, or if only some options are specified on the command line, **padem** matches the calling address to those listed in **/etc/x25/x29hosts** and gets those not specified on the command line from **/etc/x25/x29hosts**. If the option is not specified in either location, **padem** assigns a default value as follows:

name	programmatic interface name for **/dev/x25_0**
profile	default profile 0 values from above
logging	0
reverse_charge	disable

Commands

If no destination X.121 address or symbolic address is entered at the command line, **padem** enters *PAD COMMAND* state. PAD command signals are available in command mode and are grouped into X28 PAD command set, defined by CCITT X.28 recommendation, and extended command set, which adds some new functionality. If a destination X.121 address or a symbolic address is entered when **padem** is in command mode, **padem** enters data transfer mode. The syntax for entering a symbolic address in command mode is

|*symad*|

and *symad* must be present in the **host_table** set in **/etc/x25/x29hosts**.

X28 PAD Command Set

stat Display status information of a VC placed to a remote host. **padem** responds with **FREE** if no VC is established or with **ENGAGED** if there is a virtual circuit established.

par? List all X.3 parameters and their values. The response is a list of all parameters, each one displayed by its number, followed by **:**, followed by its current value.

set *n*:*v* [,*n*:*v*] Set the X.3 parameter *n* to value *v* (see CCITT RECOMMENDATIONS below). If there is a syntax error, **padem** responds with an **ERR** service signal and updates the diagnostic text. All changes take effect when a new VC is established or immediately if **padem** is already engaged.

set? *n*:*v* [,*n*:*v*] Set the X.3 parameter *n* to value *v* and display the current setting of all parameters. This command is a combination of the **set** and **par?** commands above.

int Transmit an interrupt to the remote host.

reset Reset a virtual circuit. When a virtual circuit is reset, all data received but still not read is lost, and all X.3 parameters are set back to their default values (as they were set before the virtual call was established).

clr Clear a virtual circuit. This command disconnects the VC from the remote host.

[**r**,**g**[*n*[*n*]]-]*x121_address*[*subaddress*][**P**/**D***d...d*]
Perform a virtual call to the remote host defined by its X.121 address. *x121_address* [*subaddress*] can be replaced with |*symad*| where *symad* is defined in the file **/etc/x25/x29hosts**. **P** specifies the packet size negotiation. **D** specifies the throughput class negotiation.

prof [*profile_id*] Interpret a prepared set of command signals from a *profile*. Profiles must be placed in the file **/etc/x25/x3config**.

Extended Command Set

quit Terminate **padem**. If the virtual call is established, it is cleared.

verbose Display more useful diagnostic text.

help Display a short summary of **padem** command signals.

list List all symbolic names for remote hosts accessible by the user

aprof List all profile IDs.

`lprof`	Display the last loaded profile ID.
`sleep` *s*	Suspend `padem` for *s* seconds (maximum 3600).
`ifname [def]`	`ifname` [*string*] Display or set X25 *pgm_access_name* to default or *string*.

X.3 Parameter Set

X.3 configuration parameters can specify terminal characteristics or input/output actions that the user wants `padem` to take on receipt of specific input from the user or X.29 packet from the remote PAD support.

Parameter	Recognized Values	Description
1	`0 1 32-126`	Character used for escaping from data transfer state
2	`0 1`	Controls echo of typed characters back to standard output
3	`0 1 2 4 8 16 32 64 126`	Selects data-forwarding character
4	`0-255`	Data forwarding timeout (value in twentieths of a second)
5	`0 1`	Flow Control of user's terminal by PAD
6	`0 1 5`	Control of PAD service signals
7	`0 1 2 4 5 8 16 21`	PAD action when break signal is received
8	`0 1`	Discard the output sending to user's terminal
9	`0 1-255`	Padding character after carriage return
10	`0 1-255`	Line folding
11	`0-18`	Line speed; baud rate of the user's terminal is read-only because the physical line speed cannot be changed
12	`0 1`	Flow control of PAD by user's terminal
13	`0,1,4,5,6,7`	Line-feed insertion after carriage return
14	`0 1-255`	Padding after line-feed
15	`0 1`	Editing
16	`0-127`	Delete character
17	`0-127`	Line-delete character
18	`0-127`	Line-display character
19	`0 1 2 8 32-126`	Editing PAD service signals
20	`0 1 2 4 8 16 32 64 128`	If parameter 2 is set, this parameter specified the echo mask
21		Not supported. Parity treatment is not considered for PAD emulation
22	`0-255`	Page wait

DIAGNOSTICS

Diagnostics on `padem` are provided by the [`-l` *level* [`-f` *filename*]] option or `logging` in `/etc/x25/x29hosts`.

WARNINGS

`padem` can only be used on switched virtual-circuit connections.

`padem` does not support local PAD parameters.

`padem` does not accept incoming calls. This functionality is provided by PAD support (see *x29server*(1M)).

DEPENDENCIES

The local system must have HP 9000 X.25 software installed and the remote host must have PAD support.

AUTHOR

`padem` was developed by HP.

FILES

`/etc/x25/x29hosts`
`/etc/x25/x3config`

SEE ALSO

x25init(1M), x25stat(1), x29server(1M), x29hosts(4), x3config(4).

Installing and Administering X.25 / 9000 ,
Troubleshooting X.25 / 9000 .
X.25: The PSN Connection

STANDARDS CONFORMANCE

CCITT has defined four PAD related recommendations. HP 9000 X.25 software already implements the first X.25 recommendation as well as X.28, X.29, and X.3 recommendations for incoming calls (PAD support). **padem** implements the last three recommendations for outgoing calls. **padem** conforms to the 1984 CCITT standards. The only exceptions to this rule are X.3 parameter 21 that is recognized by **padem** but whose value is ignored (not supported for PAD emulation).

NAME

passwd - change login password

SYNOPSIS

`passwd` [`-f` *file*] [*name*]

DESCRIPTION

`passwd` changes or installs a password associated with the login *name*. If *name* is omitted, `passwd` uses `getlogin()` to determine the invoking user's user name (see *getlogin*(3C)). An alternate password file can be chosen with the `-f` option. The user must have read and write permission for the file given with the `-f` option. The default password file is `/etc/passwd`.

Ordinary users can change only the password corresponding to their login *name*.

`passwd` prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered `passwd` checks to see if the old password has "aged" sufficiently. If "aging" is insufficient, the new password is rejected and `passwd` terminates; see *passwd*(4).

Assuming "aging" is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies differ, `passwd` repeats the cycle of prompting for the new password, at most twice.

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant.
- Characters must be from the 7-bit USASCII character set; letters from the English alphabet.
- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means uppercase and lowercase letters.
- Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase equivalent are are treated as identical.
- New passwords must differ from the old one by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase equivalent are are treated as identical.

A user whose effective user ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users can change any password; hence, `passwd` does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

EXTERNAL INFLUENCES

International Code Set Support

Characters from single-byte character code sets are supported in passwords.

FILES

`/etc/passwd`

SEE ALSO

id(1), login(1), su(1), crypt(3C), passwd(4).

STANDARDS CONFORMANCE

`passwd`: SVID2, XPG2

NAME

paste - merge same lines of several files or subsequent lines of one file

SYNOPSIS

`paste` *file1 file2* ...
`paste -d` *list file1 file2* ...
`paste -s` [`-d` *list*] *file1 file2* ...

DESCRIPTION

In the first two forms, `paste` concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns in a table and pastes them together horizontally (parallel merging). In other words, it is the horizontal counterpart of *cat*(1) which concatenates vertically; i.e., one file after the other. In the `-s` option form above, `paste` replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to standard output, so `paste` can be used as the start of a pipe, or as a filter if `-` is used instead of a file name.

`paste` recognizes the following options and command-line arguments:

`-d`	Without this option, the new-line characters of all but the last file (or last line in case of the `-s` option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).
list	One or more characters immediately following `-d` replace the default *tab* as the line concatenation character. The list is used circularly; i.e., when exhausted, it is reused. In parallel merging (that is, no `-s` option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list can contain the special escape sequences: `\n` (new-line), `\t` (tab), `\\` (backslash), and `\0` (empty string, not a null character). Quoting may be necessary if characters have special meaning to the shell. (For example, to get one backslash, use `-d"\\\\"`).
`-s`	Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with the `-d` option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
`-`	Can be used in place of any file name to read a line from the standard input (there is no prompting).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

List directory in one column:

```
ls | paste -d" " -
```

List directory in four columns

```
ls | paste - - - -
```

Combine pairs of lines into lines

```
paste -s -d"\t\n" file
```

NOTES

`pr -t -m`... works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS

`line too long`	Output lines are restricted to 1023 characters.
`too many files`	Except for the `-s` option, no more than **OPEN_MAX** – 3 input files can be specified (see *limits*(5)).

SEE ALSO

cut(1), grep(1), pr(1).

STANDARDS CONFORMANCE

`paste`: SVID2, XPG2, XPG3, POSIX.2

NAME

pathalias - electronic address router

SYNOPSIS

`pathalias` [`-ivcDf`] [`-l` *host*] [`-d` *link*] [`-t` *link*] [*files*]

DESCRIPTION

`pathalias` computes the shortest paths and corresponding routes from one host (computer system) to all other known, reachable hosts. `pathalias` reads host-to-host connectivity information on standard input or in the named *files*, and writes a list of host-route pairs on the standard output.

Options

`pathalias` recognizes the following options and command-line arguments:

- `-i` Ignore case: map all host names to lowercase. By default, case is significant.
- `-c` Print costs. Print the path cost (see below) before each host-route pair.
- `-v` Verbose. Report some statistics on the standard error output.
- `-D` Terminal domains. Domain members are terminal.
- `-f` First hop cost. The printed cost is the cost to the first relay in a path instead of the cost of the path itself; implies (and overrides) the `-c` option.
- `-l` *host* Set local host name to *host*. By default, `pathalias` discovers the local host name in a system-dependent way.
- `-d` *link* Declare a dead link, host, or network (see below). If *link* is of the form `host1!host2`, the link from host1 to host2 is treated as an extremely high cost (i.e., **DEAD**) link. If *link* is a single host name, that host is treated as dead and is used as an intermediate host of last resort on any path. If *link* is a network name, the network requires a gateway.
- `-t` *link* Trace input for link, host, or network on the standard error output. The form of *link* is as above.

The public domain version of `pathalias` includes two undocumented options that are briefly described in the Special Options section below.

Input Format

A line beginning with white space continues the preceding line. Anything following `#` on an input line is ignored.

A list of host-to-host connections consists of a "from" host in column 1, followed by white space, followed by a comma-separated list of "to' hosts, called *links*. A link may be preceded or followed by a network character to use in the route. Valid network characters are `!` (default), `@`, `:`, and `%`. A link (and network character, if present) may be followed by a "cost" enclosed in parentheses. Costs can be arbitrary arithmetic expressions involving numbers, parentheses, `+`, `-`, `*`, and `/`. Negative costs are prohibited. The following symbolic costs are recognized:

LOCAL	25	(local-area network connection)
DEDICATED	100	(high speed dedicated link)
DIRECT	200	(toll-free call)
DEMAND	300	(long-distance call)
HOURLY	500	(hourly poll)
EVENING	2000	(time restricted call)
DAILY	5000	(daily poll, also called POLLED)
WEEKLY	30000	(irregular poll)

In addition, **DEAD** is a very large number (effectively infinite), and **HIGH** and **LOW** are –5 and +5 respectively, for baud-rate or quality bonuses/penalties, and **FAST** is -80, for adjusting costs of links that use high-speed (9.6 Kbaud or more) modems. These symbolic costs represent an imperfect measure of bandwidth, monetary cost, and frequency of connections. For most mail traffic, it is important to minimize the number of hosts in a route, thus, *e.g.*, **HOURLY** is far greater than **DAILY** divided by 24. If no cost is given, a default of 4000 is used.

For the most part, arithmetic expressions that mix symbolic constants other than **HIGH**, **LOW**, and **FAST** make no sense. For example, if a host calls a local neighbor whenever there is work, and additionally polls every evening, the cost is `DIRECT`, *not* `DIRECT+EVENING`.

Some examples:

```
down         princeton!(DEDICATED), tilt,
             %thrash(LOCAL)
princeton    topaz!(DEMAND+LOW)
topaz        @rutgers(LOCAL+1)
```

If a link is encountered more than once, the least-cost occurrence dictates the cost and network character. Links are treated as bidirectional but asymmetric: for each link declared in the input, a DEAD reverse link is assumed.

If the "to" host in a link is surrounded by angle brackets, the link is considered *terminal*, and further links beyond this one are heavily penalized. For example, with input

```
seismo       <research>(10), research(100), ihnp4(10)
research     allegra(10)
ihnp4        allegra(50)
```

the path from `seismo` to `research` is direct, but the path from `seismo` to `allegra` uses `ihnp4` as a relay; not `research`.

The set of names by which a host is known by its neighbors is called its *aliases*. Aliases are declared as follows:

name=*alias*, *alias* ...

The name used in the route to or through aliased hosts is the name by which the host is known to its predecessor in the route.

Fully connected networks, such as the ARPANET or a local-area network, are declared as follows:

`net` = {*host, host, ...*}

The host-list can be preceded or followed by a routing character (`!` by default), and can be followed by a cost (4000 by default). The network name is optional; if not given, `pathalias` creates one.

```
etherhosts = {rahway, milan, joliet}!(LOCAL)
ringhosts = @{gimli, alida, almo}(DEDICATED)
= {etherhosts, ringhosts}(0)
```

The routing character used in a route to a network member is the one encountered when "entering" the network. See also the sections on *gateways* and *domains*.

Connection data can be given while hiding host names by declaring

`private` {*host, host, ...*}

`pathalias` does not generate routes for private hosts, but can produce routes through them. The scope of a private declaration extends from the declaration to the end of the input file in which it appears, or to a private declaration with an empty host list, whichever comes first. The latter scope rule offers a way to retain the semantics of private declarations when reading from the standard input.

Dead hosts, links, or networks can be presented in the input stream by declaring

`dead` {*arg*, ...}

where *arg* has the same form as the argument to the `-d` option.

To force a specific cost for a link, delete all prior declarations with

`delete` {*host1*!*host2*}

and declare the link as desired. To delete a host and all its links, use

`delete` {*host*}

Error diagnostics refer to the file in which the error was found. To alter the file name, use

`file` {*filename*}

Fine-tuning is possible by adjusting the weights of all links from a given host, as in

```
     adjust {host1, host-2(LOW), host3(-1)}
```

If no cost is given, a default of 4000 is used.

Input from compressed (and uncompressed) files can be piped into `pathalias` with the following script.

```
     for i in $*; do
             case $i in
             *.Z)    echo "file {'expr $i : ').Z''}
                     zcat $i ;;
             *)      echo "file {$i}"
                     cat $i ;;
             esac
             echo "private {}" done
```

Output Format

A list of host-route pairs is written to the standard output, where route is a string appropriate for use with `printf()` (see *printf*(3S)), *such* as

```
     rutgers      princeton!topaz!%s@rutgers
```

The `%s` in the route string should be replaced by the user name at the destination host (this task is normally performed by a mailer).

Except for *domains* (see below), the name of a network is never used in routes. Thus, in the earlier example, the path from `rahway` to `milan` would be `milan!%s`, not `etherhosts!milan!%s`.

Gateways

A network is represented by a pseudo-host and a set of network members. Links from the members to the network have the weight given in the input, while the cost from the network to the members is zero. If a network is declared dead, the member-to-network links are marked dead, which effectively prohibits access to the network from its members.

However, if the input also shows an explicit link from any host to the network, then that host can be used as a gateway (in particular, the gateway need not be a network member).

For example, suppose `CSNET` is declared dead on the command line and the input contains

```
     CSNET = {...}
     csnet-relay     CSNET
```

Then routes to `CSNET` hosts will use `csnet-relay` as a gateway.

Domains

A network whose name begins with `.` is called a domain. Domains are presumed to require gateways; i.e., they are `DEAD`. The route given by a path through a domain is similar to that for a network, but here the domain name is appended to the end of the name of the next host. Subdomains are permitted. For example:

```
     harvard      .EDU          # harvard is gateway to .EDU domain
     .EDU         = {.BERKELEY, .UMICH}
     .BERKELEY    = {ernie}
```

yields

```
     ernie        ...!harvard!ernie.BERKELEY.EDU!%s
```

Output is given for the nearest gateway to a domain; e.g., the example above gives

```
     .EDU          ...!harvard!%s
```

Output is given for a subdomain if it has a different route than its parent domain, or if all its ancestor domains are private.

If the `-D` option is given on the command line, `pathalias` treats a link from a domain to a host member of that domain as terminal. This property extends to host members of subdomains, etc., and discourages routes that use any domain member as a relay.

Special Options

The public domain version of `pathalias` includes two undocumented options that rewrite named files with intermediate data of limited usage. Here are brief descriptions:

`-g` *file*	Dump graph edges into *file* in the form *host*>*host* for simple connections and *host*@<tab>*host* for network connections (from hosts to networks only).
`-s` *file*	Dump shortest path tree into *file* in the form *host*<tab>[@]*host*[`!`]`(`*cost*`)`, including both connections from hosts to networks and from networks to hosts. This data may be useful for generating lists of one-way connections.

BUGS

The `-i` option should be the default.

The order of arguments is significant. In particular, `-i` and `-t` should appear early in the command line.

`pathalias` can generate hybrid (i.e., ambiguous) routes, which are abhorrent and most certainly should not be given as examples in a manual entry. Experienced mappers largely shun '@' when preparing input; this is historical, but also reflects UUCP's simplistic syntax for source routes.

Mixed-mode paths are ambiguous because the precedence of @ versus ! is not specified, varies from host to host, and is configurable. They should rarely be used.

Multiple @s in routes are prohibited by many mailers. To circumvent this restriction, mailers instead support the "magic %" rule, described below. When `pathalias` would otherwise generate a path containing multiple @s, it instead generates a path to which the "magic %" rule can be correctly applied.

Basically, the "magic %" rule for generating paths is "when constructing a path that would require multiple @s, replace all but the right-most @ with `%`.

When a mailer that supports the "magic %" rule receives a message that was routed to it via ..path..@host, it processes the route as follows:

1. Remove the trailing "@host" part of the route.
2. Examine the remaining route from right to left, proceeding to the next step when a "!" is seen. If a '%' is seen, change it to '@' and proceed to the next step immediately.
3. Continue processing the message using the modified route. If the modified route contains both '!' and '@' characters, the exact selection of the next host to route the message is governed by the specific precedence of '!' vs. '@' at this host.

   For example, if a host, `jazz.nonesuch.com`, received a message with a path `foo!joe%castle.hrh.gov.uk@jazz.nonesuch.com`, the mailer would convert the path to `foo!joe@castle.hrh.gov.uk`, and then forward it appropriately. If the host were configured such that '!' were of higher precedence than '@', the message would be forwarded to host `foo`, which would then deliver the message to `joe@castle.hrh.gov.uk`. If instead `jazz.nonesuch.com` were configured with '@' as higher in precedence, it would forward the message to host `castle.hrh.gov.uk`, which would then deliver it to `foo!joe`. (Clearly, `pathalias` could only correctly generate such a path if it knew the precedence at host `jazz.nonesuch.com`; since the database does not contain that information, such paths from `pathalias` should be viewed with suspicion.)

   The `-D` option suppresses insignificant routes to domain members. This is benign, perhaps even beneficial, but confusing, since the behavior is undocumented and somewhat unpredictable.

AUTHOR

`pathalias` was developed by Peter Honeyman and Steven M. Bellovin.

FILES

`newsgroup comp.mail.maps`	Likely location of some input files.

SEE ALSO

P.Honeyman and S.M. Bellovin, *PATHALIAS or The Care and Feeding of Relative Addresses*, in *Proc. Summer USENIX Conf.*, Atlanta, 1986.

NAME

pax - portable archive exchange

SYNOPSIS

pax [**-cimopuvy**] [**-f** *archive*] [**-s** *replstr*] [**-t** *device*] [*pattern* ...]
pax -r [**-cimnopuvy**] [**-f** *archive*] [**-s** *replstr*] [**-t** *device*] [*pattern* ...]
pax -w [**-adimuvy**] [**-b** *blocking*] [**-f** *archive*] [**-s** *replstr*] [**-t** *device*] [**-x** *format*] [*pathname* ...]
pax -rw [**-ilmopuvy**] [**-s** *replstr*] [*pathname* ...] *directory*

DESCRIPTION

pax reads and writes archive files that conform to the *Archive/Interchange File Format* specified in *IEEE Std. 1003.1-1988*. **pax** can also read, but not write, a number of other file formats in addition to those specified in the *Archive/Interchange File Format* description. Support for these traditional file formats, such as Version 7 **tar** and System V binary **cpio**-format archives, is provided for backward compatibility and to maximize portability.

Read/Write Options

Combinations of the **-r** and **-w** command-line options specify whether **pax** should read, write, or list the contents of the specified archive, or move the specified files to another directory as follows:

-w Write files and directories specified by *pathname* operands to the standard output together with the pathname and status information prescribed by the archive format used. If *pathname* is a directory, *pathname* refers to the files and (recursively) subdirectories of that directory. If *pathname* is not specified, standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied.

-r Read an archive file from the standard input. Only files whose names match any of the *pattern* operands are selected for extraction. Selected files are conditionally created and copied, relative to the current directory tree and subject to the options described below. By default, the owner and group of selected files is that of the invoking process. Permissions and modification times are the same as those in the archive.

Supported archive formats are automatically detected on input. The default output format is *ustar*, but can be overridden by the **-x** *format* option described below.

-rw Read the files and directories named in *pathname* and copy them to the destination *directory*. If *pathname* is a directory, *pathname* refers to the files and (recursively) subdirectories of that directory. If *pathname* is not specified, standard input is read to get a list of pathnames to copy, one pathname per line. In this case, only those pathnames appearing on the standard input are copied. The directory named by *directory* must exist and have the proper permissions before the copy can occur.

If neither the **-r** or **-w** options are given, **pax** lists the contents of the specified archive. In this mode, **pax** lists normal files; one per line, hard-link pathnames as

pathname == *linkname*

and symbolic link pathnames as

pathname -> *linkname*

where *pathname* is the name of the file being extracted, and *linkname* is the name of a file that appeared earlier in the archive.

If the **-v** option is specified, **pax** lists normal pathnames in the same format used by the **ls** command with the **-l** option. Hard links are shown as

<ls -l listing> == *linkname*

and symbolic links are shown as

<ls -l listing> -> *linkname*

pax is capable of reading and writing archives that span multiple physical volumes. Upon detecting an end-of-medium on an archive not yet completed, **pax** prompts the user for the next volume of the archive, and allows the user to specify the location of the next volume.

Other Options

`pax` recognizes the following additional options:

`-a`	Files specified by *pathname* are appended to the specified archive.
`-b` *blocking*	Block the output at *blocking* bytes per write to the archive file. A `k` suffix multiplies *blocking* by 1024, a `b` suffix multiplies *blocking* by 512 and an `m` suffix multiplies *blocking* by 1048576 (1 megabyte). If not specified, *blocking* is automatically determined on input and is ignored for `-rw`.
`-c`	Complement the match sense of the *pattern* operands.
`-d`	Intermediate directories not explicitly listed in the archive are not created. This option is ignored unless `-r` is specified.
`-f` *archive*	*archive* specifies the pathname of the input or output archive, overriding the default of standard input for `-r` or standard output for `-w`.
`-H`	Search hidden subdirectories (context dependent files or CDFs). Normally, only the CDF element matching the current context is backed up, without expanding the path name to show the actual element. For more information on CDFs, see *cdf*(4).
`-i`	Interactively rename files. Substitutions specified by `-s` options (described below) are performed before requesting the new file name from the user. A file is skipped if an empty line is entered and `pax` exits with an exit status of 0 if end-of-file is encountered.
`-l`	Files are linked rather than copied when possible.
`-m`	File modification times are not preserved.
`-n`	When `-r` is specified, but `-w` is not, *pattern* arguments are treated as ordinary file names. Only the first occurrence of each of these files in the input archive is read. `pax` exits with a zero exit status after all files in the list have been read. If one or more files in the list is not found, `pax` writes a diagnostic to standard error for each of the files and exits with a non-zero exit status. File names are compared before any of the `-i`, `-s`, or `-y` options are applied.
`-o`	Restore file ownership as specified in the archive. The invoking process must have appropriate privileges to accomplish this.
`-p`	Preserve access time of the input files after they have been copied.
`-s` *replstr*	File names are modified according to the substitution expression using the syntax of *ed*(1) as shown:

`-s/`*old*`/`*new*`/[gp]`

Any non-null character can be used as a delimiter (a `/` is used here as an example). Multiple `-s` expressions can be specified; the expressions are applied in the order specified terminating with the first successful substitution. The optional trailing `p` causes successful mappings to be listed on standard error. The optional trailing `g` causes the *old* expression to be replaced each time it occurs in the source string. Files that substitute to an empty string are ignored both on input and output.

`-t` *device*
device is an implementation-defined identifier that names the input or output archive device, overriding the default of standard input for `-r` and standard output for `-w`.

`-u`
Copy each file only if it is newer than an existing file with the same name. This implies `-a`.

`-v`
List file names as they are encountered. Produces a verbose table of contents listing on the standard output when both `-r` and `-w` are omitted; otherwise the file names are printed to standard error as they are encountered in the archive.

`-x` *format*
Specifies the output archive *format*. The input format, which must be one of the following, is automatically determined when `-r` is used. The supported formats are:

`cpio` Extended CPIO interchange format specified in *Extended CPIO Format* in *IEEE Std. 1003.1-1988*.

`ustar` Extended TAR interchange format specified in *Extended TAR Format* in *IEEE Std. 1003.1-1988*. This is the default archive format.

`-y`
Interactively prompt for the disposition of each file. Substitutions specified by `-s` options (described above) are performed before prompting the user for disposition. End-of-file or an input line starting with the character `q` causes `pax` to exit. Otherwise, an input line starting with anything other than `y` causes the file to be ignored. This option cannot be used in conjunction with the `-i` option.

Only the last of multiple `-f` or `-t` options is used.

When writing to an archive, standard input is used as a list of pathnames if *pathname* is not specified. The format is one pathname per line. Otherwise, the standard input is the archive file, which is formatted according to one of the specifications in *Archive/Interchange File format* in *IEEE Std. 1003.1-1988*, or some other implementation-defined format.

The user ID and group ID of the process, together with the appropriate privileges, affect the ability of `pax` to restore ownership and permissions attributes of the archived files. (See *format-reading utility* in `Archive/Interchange File Format` in *IEEE Std. 1003.1-1988*.)

The `-a`, `-c`, `-d`, `-i`, `-l`, `-p`, `-t`, `-u`, and `-y` options are provided for functional compatibility with the historical `cpio` and `tar` utilities. Option defaults were chosen based on the most common usage of these options. Therefore, some options have meanings that are different from those of the historical commands.

Operands

`pax` recognizes the following operands:

directory The destination directory pathname for copies when both the `-r` and `-w` options are specified. The directory must exist and be writable before the copy or an error results.

pathname A file whose contents are used instead of the files named on the standard input. When a directory is named, all of its files and (recursively) subdirectories are copied as well.

pattern A *pattern* is given in the standard shell pattern matching notation. If no *pattern* is specified, the default is `*` which selects all files.

RETURN VALUE

`pax` exits with one of the following values:

0	All files in the archive were processed successfully.
>0	`pax` aborted due to errors encountered during operation.

DIAGNOSTICS

If an error occurs, `pax` terminates immediately without processing any additional files on the command line or in the archive.

EXAMPLES

Copy the contents of the current directory to tape drive 0:

```
pax -w -f /dev/rmt0 .
```

Copy the contents of `olddir` to `newdir`:

```
mkdir  newdir
cd  olddir
pax -rw . newdir
```

Read the archive `pax.out` whose files are all rooted in `/usr` and extract the archive relative to the current directory:

```
pax -r -s ',//*usr//*,,' -f pax.out
```

WARNINGS

Special permissions may be required to copy or extract special files.

Device, user ID, and group ID numbers larger than 65 535 cause additional header records to be output. These records are ignored by some historical versions of *cpio*(1) and *tar*(1).

The archive formats described in *Archive/Interchange File Format* have certain restrictions that have been carried over from historical usage. For example, there are restrictions on the length of pathnames stored in the archive.

When getting a listing in `ls -l` style *tar*-format archives, link counts are listed as zero because the *ustar* archive format does not keep link count information.

AUTHOR

`pax` was developed by Mark H. Colburn and HP.

FILES

`/dev/tty`	used to prompt user for information when `-i` or `-y` options are specified.

SEE ALSO

cpio(1), find(1), tar(1), cpio(5), tar(5).

STANDARDS CONFORMANCE

`pax`: POSIX.2

(Requires Optional Pascal Compiler Software)

NAME
pc - Pascal compiler

SYNOPSIS
`pc` [*options*] *file* ...

REMARKS
This manual entry describes the generic HP Pascal compiler. Implementation dependencies for different architectures are noted as needed.

DESCRIPTION
`pc` is the HP standard Pascal compiler. It accepts several types of file arguments:

- Arguments whose names end with `.p` are taken to be Pascal source files. They are each compiled, and each corresponding object program or module(s) is left in the current directory in a file whose name is that of the source, with `.o` substituted for `.p`. The `.o` file is immediately deleted (leaving only the linked executable file) if only a single source is compiled and linked, if the `-C` option is specified, or if the source fails to compile correctly.
- All other file arguments, including those whose names end with `.o`, `.a`, or `.sl`, are passed on to the `ld` linker (see (*ld*(1)) to be linked into the final program.

Options and other arguments can be passed to the compiler through the `PCOPTS` environment variable as well as on the command line. The compiler picks up the value of `PCOPTS` and places its contents before any arguments on the command line. For example (in *sh*(1) notation),

```
PCOPTS=-v
export PCOPTS
pc -L prog.p
```

is equivalent to

```
pc -v -L prog.p
```

When set, the `TMPDIR` environment variable specifies a directory to be used for temporary files, overriding the default directories `/tmp` and `/usr/tmp`.

Options
`pc` recognizes the following options:

-A Produce warnings for the use of non-ANSI-Pascal features (same as `$ANSI ON$`).

-C Suppress code generation. No `.o` file is created and linking is suppressed. This is effectively a request for syntax/semantic checking only (same as `$CODE OFF$`).

-c Suppress linking and only produce object (`.o`) files from source files.

-D*name*=*bool*
-D*name* Define *name* as if by a `$SET` directive on the 0th line of the source file. *bool* can be either `TRUE` or `FALSE`; if no =*bool* is given, *name* is set to `TRUE`. Whether *name* and *bool* are uppercase or lowercase is insignificant. The order in which the compiler encounters `$SET`s (regardless of relative order on the command-line) is: `-D`*name*=*bool*, `+Q` *dfile*, source file. Hence, the compiler overrides `-D`*name*=*bool* with any subsequent duplicate `$SET`s, always taking the last one seen (with a warning). (The `-D` option is not available on Series 300/400 systems).

-g Generate additional information needed by a symbolic debugger, and ensure that the program is linked as required. See the appropriate implementation reference manual for more information on symbolic debugging support.

-G Prepare object files for profiling with `gprof` (see *gprof*(1)).

-I*dir* Add *dir* to the list of directories that are searched for `$INCLUDE` files whose names do not begin with `/`. The directory containing the source file is always searched first, followed by any directories specified with the `-I` option, then the current working directory, and finally the standard directory `/usr/include`.

-L Write a program listing to *stdout* (see the DEPENDENCIES section below for exceptions).

`-l`*x* Cause the linker to search the library **libx.sl** or **libx.a** in an attempt to resolve currently unresolved external references. Because a library is searched when its name is encountered, placement of a `-l` is significant. If a file contains an unresolved external reference, the library containing the definition must be placed *after* the file on the command line. See *ld*(1) for further details.

`-N` Cause the output file from the linker to be marked as unshareable (see **-n**). For details and system defaults, refer to the linker documentation (*ld*(1)).

`-n` Cause the output file from the linker to be marked as shareable (see **-N**). For details and system defaults, refer to the linker documentation (*ld*(1)).

`-o` *outfile* Name the output file from the linker *outfile* instead of `a.out`.

`-p` Prepare object files for profiling with `prof` (see *prof*(1)).

`-P` *lines* Specifies the number of lines (including any header or trailer) which should be listed per page of generated listing (same as **$LINES** *n* **$**).

`-Q` Cause the output file from the linker to be marked as not demand-loadable (see **-q**). For details and system defaults, refer to the linker documentation (*ld*(1)).

`-q` Cause the output file from the linker to be marked as demand loadable (see **-Q**). For details and system defaults, refer to the linker documentation (*ld*(1)).

`-s` Cause the output of the linker to be **strip**ped of symbol table information (see *ld*(1) and *strip*(1)) (this option is incompatible with symbolic debugging).

`-t` *x*,*name* Substitute or insert subprocess *x* with *name* where *x* is one or more of an implementation-defined set of identifiers indicating the subprocess(es). This option works in two modes:

- If *x* is a single identifier, *name* represents the full pathname of the new subprocess;
- If *x* is a set of (more than one) identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path name of the new subprocesses.

The values *x* can assume are:

c	compiler body (standard suffix is **pascomp**)
0	same as c
1	linker (standard suffix is `ld`)

-v
Enable verbose mode, producing a step-by-step description of the compilation process on *stderr*.

-w
Suppress warning messages (same as **$WARN OFF$**).

`-W` *x*,*arg1*[,*arg2*,...,*argN*]
Cause *arg1* through *argN* to be handed off to subprocess *x*. The *argi* are of the form:

-argoption[,*argvalue*],

where *argoption* is the name of an option recognized by the subprocess and *argvalue* is a separate argument to *argoption* where necessary. The values that *x* can assume are those listed under the `-t` option, as well as the value `d` (driver program), which has a special meaning explained below.

For example, the specification to pass the **-a archive** option to the linker to cause it to link against an archive library rather than a shared library would be:

```
-W l,-a,archive
```

The **-W d** option specification allows additional, implementation-specific options to be recognized and passed through the compiler driver to the appropriate compiler subprocesses. For example, on Series 300/400 systems:

```
-W d,-U
```

sends the option **-U** to the driver and compiler. Furthermore, a shorthand notation for this mechanism can be used by inserting + in front of the option name; as in

`+U`

which is equivalent to the previous option expression. Note that for simplicity this shorthand is applied to each implementation-specific option individually, and that the *argvalue* is no longer separated from the *argoption* by a comma (see `-W`).

`-Y`
Enable 16-bit Native Language Support when parsing string literals and comments (same as `$NLS_SOURCE ON$`). Note that 8-bit parsing is always supported.

`+a`
Cause the compiler to generate archived object (`.a`) files instead of simple object (`.o`) files. This allows source files containing multiple HP Pascal modules to be compiled such that each module can be linked independently. Use of this option is discouraged for portability reasons. It is provided to facilitate migration from Series 200 HP-UX releases prior to Release 5.1 (where `.a` files were always generated), to 5.1 and subsequent releases. Otherwise, it is recommended that modules needing separate linkability be placed in separate source files. To facilitate use of previously existing makefiles and scripts that depended on the archive generation, the `PCOPTS` environment variable can be used (e.g., `set PCOPTS="+a$PCOPTS"`).

`+Q` *dfile*
Cause *dfile* to be read before compilation of each source file. *dfile* must contain only compiler directives.

`+R`
Inhibit range checking (same as `$RANGE OFF$`).

EXTERNAL INFLUENCES

Environment Variables

`LANG` determines the language in which messages are displayed.

`PCOPTS` specifies options passed to the compiler.

`TMPDIR` specifies a directory to be used for temporary files.

International Code Set Support

Single- and multi-byte character code sets are supported only in strings and comments.

EXAMPLES

Series 700/800:

The following two commands create `one.o` and `two.o`, in succession, then link `one.o` and `two.o` with `/usr/lib/end.o` to create `prog`.

```
pc -cg one.p two.p
pc -g one.o two.o -o prog
```

To link object files that were compiled with `+DA1.1`, be sure to use the `+DA1.1` option to `pc` (in order to link `one.o` and `two.o` with the math libraries in `/lib/pa1.1` and `/usr/lib/pa1.1`):

```
pc +DA1.1 one.o two.o
```

An object file can conveniently be renamed using the `-c` and `-o`*outfile* options:

```
pc -g -c file.p -o ../obj/Debug/file.o
```

The following command causes level 3 optimization to be performed, dropping to level 1 optimization for those procedures containing more than 700 basic blocks.

```
pc +Obb700 +O3 file.p
```

DIAGNOSTICS

The diagnostics produced by `pc` are intended to be self-explanatory. Occasional messages may be produced by the linker.

A list of all compiler errors are contained in `/usr/lib/paserrs`.

If a listing is requested (`-L` option), errors are written to the listing file (*stdout*). If a listing is requested and if *stdout* and/or *stderr* has been redirected to something other than a terminal, errors are also written to *stderr*. If no listing is requested (no `-L` option), errors are written to *stderr*. This effectively guarantees that *stderr* will always receive error messages unless duplicate error messages would be printed on the terminal.

DEPENDENCIES

Series 300/400

The Series 300/400 compiler provides only limited support for shared libraries by allowing production of dynamic load libraries. Refer to the `+l` option below for more information.

The `-L` option writes a program listing to the file given in the `$LIST filename$` option in the source, instead of to *stdout*.

The Series 300/400 compiler does not support the **-D***name=bool* option.

The following options are implemented only on Series 300/400:

-T	Causes the compiler to produce a symbol table in the listing file as if the source file contains a **$TABLES ON$**. This option requires a **-L** option.
+A	Cause the compiler to always use 2-byte data and stack alignment rules instead of default 4-byte alignment rules for stacks and data objects exceeding four bytes. Refer to the reference manual for more details.
+M	Cause the compiler not to generate inline code for the MC68881 floating-point coprocessor. Library routines are referenced for math and intrinsic operations.
+l	Produces a supplementary symbol table to support the production of dynamic load libraries. Refer to the *Programming on HP-UX* and *HP Pascal Reference Manual* for a complete discussion of this type of library.
+bfpa	Generate code that uses the HP98248A floating-point accelerator card if it is installed at run-time. If the card is not installed, floating point operations are done on the MC68881 math coprocessor.
+ffpa	Generate code for the HP98248A floating point accelerator card. This code does not run unless the card is installed.
+U	Allow certain packed fields and elements to be passed by **VAR** (Same as **$ALLOW_PACKED ON$**).

Series 700/800

The following Series 300/400 options are ignored and warnings are issued: **+M**, **+bfpa**, and **+ffpa**.

The following options are implemented on Series 700 and 800 systems:

-y	Generate additional information needed by static analysis tools, and ensure that the program is linked as required for static analysis. This option is incompatible with optimization.
+z,+Z	Both of these options cause the compiler to generate position-independent code for use in building shared libraries. The options **-G** and **-p** are ignored if **+z** or **+Z** is used. Normally, **+z** should be used to generate position-independent code. However, when certain limits are exceeded, **+Z** is required to generate position-independent code. The **ld** linker (see *ld*(1)) issues the error mesage when building a shared library indicating when **+Z** is required. If both **+z** and **+Z** are specified, only the last one encountered is used. For a more complete discussion regarding position-independent code and these options, see the manual *Programming on HP-UX*. (**+z** is the same as **$SHLIB_CODE ON$**; **+Z** is the same as **$SHLIB_CODE 2$**).
-O	Perform level 2 optimizations.
-S	Output an assembly file. This file is named *filename***.s**, where *filename* is the basename of the source file.
+C	Convert MPE-style file names (*file*[*.group*[*.account*]]) into HPUX-style filenames ([[*account/*]*group/*]*file*) in compiler directives such as **$INCLUDE** and **$SYSINTR**. This option expects a strictly MPE-like directory structure; i.e. "group"-level directories under "account"-level directories, and source files only in "group"-level directories. See the Series 800 *HP Pascal Programmers' Guide* for details on the semantics of this option. Same as **$CONVERT_MPE_NAMES ON$**.

+N Turn off notes (same as `$NOTES OFF$`). Notes are also automatically turned off when **-w** is specified.

+O*arg* Perform optimizations selected by *arg*. There are two kinds of arguments to the **+O** optimization option. Those in the first group can have *arg* defined as:

- **1** Perform level 1 optimizations. These include branch optimizations, dead code elimination, instruction scheduling, and peephole optimization.
- **2** Perform level 2 optimizations. These include common subexpression elimination, constant folding, loop invariant code motion, coloring register allocation, and store-copy optimization. Level 2 optimizations are a superset of Level 1 optimizations. The **-O** option is equivalent to the **+O2** option.
- **3** Perform level 3 optimizations. These include (but are not limited to) interprocedural global optimizations. Level 3 optimizations are a superset of Level 2 optimizations.

Those in the second group can have *arg* defined as:

- **s** Suppress optimizations which tend to increase the generated code size. Currently, these optimizations include software pipelining and loop unrolling.
- **bb***num* Specify the maximum number of basic blocks allowed in a procedure which is to be optimized at level 2. If a procedure contains more than *num* basic blocks, level 1 optimization is performed for that procedure. The default value for *num* is 500 (same as `$OPTIMIZE 'BASIC_BLOCKS` *num* `'$`).

The arguments in the second group implicitly request level 2 optimizations, but an argument from the first group will override the implicit level 2, regardless of their relative positions on the command line.

+DA*model*
Generate object code for a specific version of the PA-RISC architecture. *model* can be either a model number (such as **750** for the HP9000 Model 750), or one of the following generic architecture specifications:

- **1.0** Generate object code suitable for all implementations of PA-RISC 1.0 or higher. This is the default for all Series 800 models.
- **1.1** Generate object code suitable for all implementations of PA-RISC 1.1. This is the default for all Series 700 models.

Object code generated for PA-RISC 1.1 will not execute on PA-RISC 1.0 implementations.

In addition, the **+DA** option allows the user to selectively link with PA-RISC 1.0 or PA-RISC 1.1 Math Libraries.

+DS*model*
Perform instruction scheduling appropriate for a specific implementation of the PA-RISC architecture. *model* can be either a model number (e.g., **870** for the HP9000 Model 870), or one of the following generic architecture specifications:

- **1.0** Perform generic scheduling tuned to a model representative of PA-RISC 1.0 implementations.
- **1.1** Perform generic scheduling tuned to a model representative of PA-RISC 1.1 implementations.

The default scheduling is based on the model number returned by **uname()** (see *uname*(2)).

This option affects only performance of the object code by scheduling the code based on the specific latencies of the target implementation. The resulting code will execute correctly on other implementations (subject to the **+DA** option above).

+FP*flags*
Specify how the run time environment for floating-point operations should be initialized at program start up. The default is that all behaviors are disabled. See *ld*(1) for specific values of *flags*. To

dynamically change these settings at run time, refer to *fpgetround*(3M).

FILES

file`.p`	input file (Pascal source file)
file`.o`	compiler-generated or other object file to be relocated at link time
`a.out`	linked executable output file
`/usr/lib/pascomp`	compiler
`/usr/lib/paserrs`	compiler error message file
`/lib/crt0.o`	runtime startup
`/lib/libc.a`	HP-UX system library (C-language library)
`/lib/libc.sl`	shared version of the HP-UX system library (C-language library)
`/usr/tmp/*`	temporary files used by the compiler

Series 300/400

file`.a`	optionally generated object archive file
`/lib/libm.a`	HP-UX math library
`/lib/libm.sl`	shared version of the HP-UX math library
`/usr/lib/escerrs`	Pascal escape codes
`/usr/lib/syserrs`	HP-UX system messages
`/usr/lib/ioerrs`	Pascal I/O results
`/lib/libpc.a`	Pascal run-time library
`/lib/libp/libpc.a`	profilable Pascal run-time library
`/usr/lib/libheap2.a`	Pascal run-time library, alternate heap manager
`/lib/libp/libheap2.a`	profilable Pascal run-time library, alternate heap manager

Series 700/800

`/lib/libm.a`	PA-RISC 1.0 Math Library
`/lib/libm.sl`	shared version of PA-RISC 1.0 Math Library
`/lib/pa1.1/libm.a`	PA-RISC 1.1 Math Library
`/lib/pa1.1/libm.sl`	shared version of PA-RISC 1.1 Math Library
`/usr/lib/nls/$LANG/pc_msgs.cat`	error message catalog
`/usr/include/pasesc.ph`	symbolic definitions of library escape codes
`/usr/lib/libcl.a`	PA-RISC 1.0 Pascal run-time library
`/usr/lib/libcl.sl`	shared version of PA-RISC 1.0 Pascal run-time library
`/usr/lib/pa1.1/libcl.a`	PA-RISC 1.1 Pascal run-time library
`/usr/lib/pa1.1/libcl.sl`	shared version of PA-RISC 1.1 Pascal run-time library
`/usr/lib/pasopts`	global compiler options file
`/usr/lib/sched.models`	processor implementation file
`/usr/lib/paslib`	default system module library

SEE ALSO

Texts and Tutorials

Programming in Pascal with Hewlett-Packard Pascal by Peter Grogono.
Programming on HP-UX.

Series 300/400

HP Pascal Reference Manual

Series 700/800

HP Pascal Reference Manual"
HP Pascal Programmers' Guide
HP-UX Floating-Point Guide

See *manuals*(5) for list of part numbers.

Program management and analysis tools:

softstatic(1)	static analysis tool (shipped with SoftBench software)

Profiling and debugging tools:

gprof(1)	display call graph profile data
prof(1)	display profile data
xdb(1)	invoke the FORTRAN, C, C++, and Pascal symbolic debugger

System tools:

ar(1)	create archived libraries
ld(1)	invoke the linker

Miscellaneous:

matherr(3M)	trap math errors
strip(1)	strip symbol and line number information from an object file

NAME

pg - file perusal filter for soft-copy terminals

SYNOPSIS

`pg` [`-`*number*] [`-p`*string*] [`-cefns`] [`+`*linenumber*] [`+/` *pattern*] [*file* ...]

Remarks

`pg` and `more` are both used in similar situations (see *more*(1)). Text highlighting features supported by `more` are not available from `pg`. However, `pg` has some useful features not provided by `more`.

DESCRIPTION

`pg` is a *text file* filter that allows the examination of *files* one screenful at a time on a soft-copy terminal. If `-` is used as a *file* argument, or `pg` detects NULL arguments in the comand line, the standard input is used. Each screenful is followed by a prompt. To display a new page, press **Return**. Other possibilities are enumerated below.

This command is different from other paginators such as `more` in that it can back up for reviewing something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, `pg` scans the `terminfo` data base for the terminal type specified by the environment variable **TERM** (see *terminfo*(4)). If **TERM** is not defined, terminal type **dumb** is assumed.

Options

`pg` recognizes the following command line options:

-*number*	*number* is an integer specifying the size (in lines) of the window that `pg` is to use instead of the default (on a terminal containing 24 lines, the default window size is 23).
`-p` *string*	Causes to use *string* as the prompt. If the prompt string contains a `%d`, the first occurrence of `%d` in the prompt is replaced by the current page number when the prompt is issued. The default prompt string is a colon (`:`).
`-c`	Home the cursor and clear the screen before displaying each page. This option is ignored if `clear_screen` is not defined in the `terminfo` data base for this terminal type.
`-e`	Causes `pg` to *not* pause at the end of each file.
`-f`	Normally, `pg` splits lines longer than the screen width, but some sequences of characters in the text being displayed (such as escape sequences for underlining) generate undesirable results. The `-f` option inhibits `pg` from splitting lines.
`-n`	Normally, commands must be terminated by a new-line character. This option causes an automatic end-of-command as soon as a command letter is entered.
`-s`	Causes `pg` to print all messages and prompts in standout mode (usually inverse video).
+*linenumber*	Start display at *linenumber*.
+`/`*pattern*`/`	Start up at the first line containing text that matches the regular expression *pattern*.

`pg` looks in the environment variable `PG` to preset any flags desired. For example, if you prefer to view files using the `-c` mode of operation, the Bourne-shell command sequence `PG='-c' ; export PG` or the C-shell command `setenv PG -c` causes all invocations of `pg`, including invocations by programs such as `man` and `msgs`, to use this mode. The command sequence to set up the `PG` environment variable is normally placed in the user `.profile` or `.cshrc` file.

The responses that can be typed when `pg` pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands that cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted either in pages or lines, depending on the command. A signed *address* specifies a point relative to the current page or line; an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

Perusal commands and their defaults are as follows:

(+1)*<newline>* or *<blank>*
: Displays one page. The address is specified in pages.

(+1) `l`
: With a relative address, `pg` simulates scrolling the screen, forward or backward, the number of lines specified. With an absolute address `pg` prints a screenful beginning at the specified line.

(+1) `d` or `^D`
: Simulates scrolling a half-screen forward or backward.

The following perusal commands take no *address*:

`.` or `^L`
: Typing a single period causes the current page of text to be redisplayed.

`$`
: Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The Basic Regular Expression syntax (see *regexp*(5)) is supported. Regular expressions must always be terminated by a new-line character, even if the `-n` option is specified.

i`/`*pattern*`/`
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i`^`*pattern*`^`
i`?`*pattern*`?`
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The `^` notation is useful for Adds 100 terminals which cannot properly handle the `?`.

After searching, `pg` normally displays the line found at the top of the screen. This can be modified by appending `m` or `b` to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix `t` can be used to restore the original situation.

`pg` users can modify the perusal environment with the following commands:

i`n`
: Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

i`p`
: Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

i`w`
: Display another window of text. If *i* is present, set the window size to *i*.

`s` *filename*
: Save the input in the named file. Only the current file being perused is saved. The white space between the `s` and *filename* is optional. This command must always be terminated by a new-line character, even if the `-n` option is specified.

`h`
: Help by displaying an abbreviated summary of available commands.

`q` or `Q`
: Quit *pg*.

`!`*command*
: *command* is passed to the shell, whose name is taken from the `SHELL` environment variable. If this is not available, the default shell is used. This command must always be terminated by a new-line character, even if the `-n` option is specified.

To cause `pg` to stop sending output and display the prompt at any time when output is being sent to the terminal, press the quit key (normally Ctrl-\) or the interrupt (break) key. Any one of the above commands can then be entered in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, `pg` is functionally equivalent to `cat` (see *cat*(1)), except that a header is printed before each file if more than one file is specified.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If **LC_COLLATE** or **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **pg** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLEs

To use **pg** when reading system news:

```
news | pg -p "(Page %d):"
```

WARNINGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using **pg** as a filter with another command that changes the terminal I/O options (such as *crypt*(1)), terminal settings may not be restored correctly.

While waiting for terminal input, **pg** responds to **BREAK**, **DEL**, and **^** by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, because an interrupt is likely to terminate the other commands in the pipeline.

Users of **more** will find that the **z** and **f** commands are available, and that the terminal **/**, **^**, or **?** can be omitted from the pattern search commands.

FILES

/usr/lib/terminfo/?/*	terminal information data base
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

crypt(1), grep(1), more(1), terminfo(4), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

pg: SVID2, XPG2, XPG3

(Requires Optional LAN/X.25 Software)

NAME

ppl - point-to-point serial networking

SYNOPSIS

`ppl` [`-o` | `-i`] [`-t` *tty*] [`-v`] [`-r` *rtprio*] [*remote_host*]

DESCRIPTION

`ppl` extends the Internet Protocol (IP) network over serial lines. It permits the use of hardwired, dial-out, or dial-in serial (tty) lines. `ppl` establishes a connection over the serial line, then runs an **encapsulation protocol** to transfer packets between the IP network and the serial line. Once started, `ppl` can only be stopped by loss of the serial line (i.e. carrier failure indication from a modem), or by sending it a `SIGTERM` signal.

Currently only the SLIP (Serial Line Internet Protcol) and ASLIP (Abbreviated SLIP) encapsulation protocols are available.

`ppl` can be configured in a number of ways, and behaves according to how it is configured and the command line options used. Configuration is controlled by configuration files: `ppl.remotes` , `ppl.ipool` , and `ppl.users` . `ppl` can be configured to share UUCP serial lines.

Options

`-i` Run the encapsulation protocol on the terminal device from which the program is invoked. This is the default if neither `-o` nor `-i` is specified.

`-o` The terminal on which the data pump is started is determined from configuration files, UUCP, or the `-t` option.

`-t` *devicename*
Specifies the *devicename* on which the encapsulation protocol is started, *only* if a device name is not specified in configuration file `ppl.remotes` for the remote host. This option only applies when used with the `-o` option.

`-v` Be verbose. This option is only available to the super-user. Verbose debugging information is displayed on the terminal and placed in the log. Secure information is included.

`-r` *rtprio* Set real-time priority during data pump operation. *rtprio* can be a decimal number between 0 and 127. By default, `ppl` runs at a real-time priority of 110.

remote_host specifies the remote host to communicate with. Either a name or an Internet address (in dot format) can be supplied. This must be the last argument on the command line. If none is supplied (with the `-i` option), the users login name is used to find one in the `ppl.users` file.

Dedicated Connections

Dedicated connections are lines that are dedicated to *ppl*. There is no getty/login running on the local or remote end of the serial line. Such a line can be either hard-wired, dial-out, or dial-in.

Alternatively, `ppl` can run on lines that are used for other purposes; specifically, a getty/login on incoming lines, or shared with UUCP on outgoing lines.

Dial-Out Connections

To dial out from an HP-UX system, use a command resembling:

`ppl -o` *remote_host*

to establish a connection to a remote host. Any progress or error messages are directed to the users standard error. `ppl` accesses the `ppl.remotes` file to obtain information needed to establish the connection. If necessary, it also uses UUCP files to select a line, acquire the phone number, and control the modem. Note that `ppl` uses various files created and maintained by the UUCP system, but it does not use the UUCP facility to handle inter-system communication.

The UUCP line is selected as follows:

1. If the entry in configuration file `ppl.remotes` does not contain a tty device name, `ppl` looks in the UUCP `Systems` file for a tty device (and speed), based on the `ppl.remotes` entry UUCP *system* name or *remote host* name.

2. `ppl` then searches for the tty device name and speed in the UUCP `Devices` file. This information is used for modem dialing instructions.

After a successful connection is made, `ppl` makes itself into a "daemon" by detaching itself from the users tty group. The user gets back a shell prompt, and the `ppl` program runs in the background. The user can then perform network commands to the remote host.

Dial-In Connections

To make a dial-in connection to an HP-UX system, dial in on a serial line to the HP-UX system, and log in as usual. Once logged in, the line can be converted from log-in to network use by running `ppl`. The `ppl` program uses the login name to search `ppl.users` for a remote hostname. The remote hostname is used to access the `ppl.remotes` file to obtain all of the configuration information needed to start the protocol. `ppl` then starts the desired protocol. Once the protocol starts, the users' local machine can begin packet transfers with `ppl`. `ppl` never (voluntarily) exits, and therefore never returns a prompt.

Only one invocation of `ppl` is allowed to each remote host Internet address.

A unique local Internet address must be associated with each invocation of `ppl`. This can be specified in the `ppl.remotes` file, or it can be picked at run time from a pool of Internet addresses given in `ppl.ipool`.

Log Files

`ppl` keeps a log of its activities in `/usr/spool/ppl/log`.

Another log can optionally be kept for creating bills or keeping track of serial line usage. If the file `/usr/spool/ppl/bill` exists, `ppl` adds an entry to this file each time it gracefully exits. The contents of the billing file are in human readable ASCII form. The file is also program readable, suitable for export to various databases. It contains the following fields, separated by `;` : bill end time, bill start time, Local inetaddr, Remote inetaddr, login user name, tty name, and protocol.

The current status of all invocations of `ppl` are kept updated in file `/usr/spool/ppl/ptmp` if the file exists.

Lock Files

`ppl` creates two lock files for each invocation of the program. These lock files serve three purposes:

- They lock access to the Internet network address by encoding the Internet address (in hex) as part of the file name.
- The files contain various details about the connection.
- The files can be executed (by owner or root) as a shell script. When executed, they send a signal to *ppl*, causing it to exit gracefully.

Whenever `ppl` exits gracefully, it removes these lock files and releases resources.

If `ppl` uses a UUCP line, it observes and creates UUCP lock file protocol.

Gateway Packets

ASLIP is an abbreviated header enhancement to SLIP. It provides throughput economy in non-gateway applications. One end of an ASLIP line must act as a "client"; the other end as a "server". Typically the client is a personal workstation; the server a larger machine.

The client sends non-gateway packets using ASLIP and gateway packets using SLIP. The server adapts its operation to the behavior of the client. ASLIP operation must be specified in the `ppl.remotes` file.

FILES

`/usr/lib/ppl/ppl.remotes`	all options for each remote connection
`/usr/lib/ppl/ppl.ipool`	pool of local internet addresses that can be used
`/usr/lib/ppl/ppl.users`	translates users login name to a remote host
`/usr/spool/ppl/PPL.`*xxxx*	termination command file and lock file for local Internet address
`/usr/spool/ppl/ppl.`*xxxx*	termination command file and lock file for remote Internet address
`/usr/spool/ppl/log`	audit trail and log file
`/usr/spool/ppl/bill`	billing file (optional)
`/usr/spool/ppl/ptmp`	status file (optional)

AUTHOR

`ppl` was developed by HP.

WARNINGS

The log and bill files grow without bound.

SEE ALSO

pplstat(1), ppl.remotes(4), ppl.ipool(4), ppl.users(4), ppl.ptmp(4),
Using Serial Line IP Protocols, UUCP tutorial in *Remote Access Users Guide*.

NAME

pplstat - give status of each invocation of *ppl*(1)

SYNOPSIS

`pplstat [-l][-n]`

DESCRIPTION

`pplstat` displays connection information for each current invocation of `ppl` (see *ppl*(1)). The tty device file, internet addresses, user who invoked `ppl`, and other useful information are given. `pplstat` displays each record in the `ptmp` file that is not marked idle (see *ptmp*(4)).

Information can be displayed in two formats. By default, an abbreviated table format is given with one entry per line resembling the following:

```
User    TTY     When         Protocol     Local      Remote
guest   tty0    030 18:30    SLIP         packard    hewlett
```

The `-l` option causes a more verbose, multi-line format resembling:

```
log_name            guest
rhost_name          hewlett
system_name         packard
Linetaddr           packard
Rinetaddr           15.255.136.4
netmask             255.0.0.0
tty                 /dev/tty0
ni                  /dev/ni0
status              SLIP
killfile (r)        /usr/spool/ppl/ppl.0fff8804
killfile (l)        /usr/spool/ppl/PPL.0fff8806
start_time          Tue Jan 30 18:30:32 1990
```

Options

`pplstat` recognizes the following options:

`-l`	Produce a long-form display.
`-n`	Print host and network addresses in Internet dotted decimal notation.

FILES

`/usr/spool/ppl/ptmp(4)`

AUTHOR

`pplstat` was developed by HP.

SEE ALSO

ppl(1), ptmp(4).

NAME

pr - print files

SYNOPSIS

`pr` [*options*] [*files*]

DESCRIPTION

`pr` prints the named files on the standard output. If *file* is `-`, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the `-s` option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until `pr` has completed printing.

Options

The following *options* can be used singly or combined in any order:

`+`*k*	Begin printing with page *k* (default is 1).
`-`*k*	Produce *k*-column output (default is 1). This option should not be used with `-m`. The options `-e` and `-i` are assumed for multi-column output.
`-a`	Print multi-column output across the page. This option is appropriate only with the `-k` option.
`-m`	Merge and print all files simultaneously, one per column (overrides the -*k* and `-a` options).
`-d`	Double-space the output.
`-e`*ck*	Expand *input* tabs to character positions *k*+1, 2×*k*+1, 3×*k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
`-i`*ck*	In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2×*k*+1, 3×*k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).
`-n`*ck*	Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of `-m` output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
`-w`*k*	Set the width of a line to *k* character positions (default is 72 for equal-width, multi-column output; no limit otherwise). Width specifications are only effective for multi-columnar output.
`-o`*k*	Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
`-l`*k*	Set the length of a page to *k* lines (default is 66). f *k* is less than what is needed for the page header and trailer, the `-t` option is in effect; that is, header and trailer lines are suppressed in order to make room for text.
`-h`	Use the next argument as the header to be printed instead of the file name.
`-p`	Pause before beginning each page if the output is directed to a terminal (`pr` rings the bell at the terminal and waits for a **Return**).
`-F`	Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
`-f`	Same as `-F`. Provided for backwards compatibility.

`-r`	Print no diagnostic reports on failure to open files.
`-t`	Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
`-s`*c*	Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text and the arguments associated with the `-e`, `-i`, `-n`, and `-s` options as single and/or multi-byte characters.

`LC_TIME` determines the format and contents of date and time strings.

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` or `LC_TIME` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `pr` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`pr` returns the following values upon completion:

0	Successful completion.
>0	One or more of the input *files* do not exist or cannot be opened.

EXAMPLES

Print `file1` and `file2` as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write `file1` on `file2`, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr -e9 -t <file1 >file2
```

Print `file1` in default format with non-blank lines numbered down the left side:

```
nl file1 |pr
```

FILES

`/dev/tty`	to suspend messages

SEE ALSO

cat(1), lp(1), nl(1), ul(1).

STANDARDS CONFORMANCE

`pr`: SVID2, XPG2, XPG3, POSIX.2

NAME

praliases - print system-wide sendmail aliases

SYNOPSIS

`praliases` [`-f` *file*] [key ...]

DESCRIPTION

`praliases` prints out the contents of the alias data base used by `sendmail` to resolve system-wide mail aliases. The alias data base is built with the command `newaliases` or `/usr/lib/sendmail -bi`. See *sendmail*(1M).

If the `-f` option is specified, `praliases` accesses the alias database built from *file* with the command

`newaliases -oA` *file*

Otherwise, `praliases` accesses the database built from the default alias file, `/usr/lib/aliases`.

Note that `praliases` accesses the database, not the alias file itself. If the alias file has changed since the alias database was last built, naturally the output of `praliases` cannot match the contents of the alias file.

Each *key* argument, if any, is looked up in the alias database. `praliases` prints out the aliases to which each key expands in the form:

key `:` *mailing list*

where mailing list can be a comma-separated list of addresses to which the key resolves.

DIAGNOSTICS

key`: No such key`
key was not found in the alias database.

EXAMPLES

```
$ praliases root postmaster no_user
root: jan_user
postmaster: joe_user
no_user: No such key
```

The output reveals that `root` is aliased to `jan_user`, `postmaster` is aliased to `joe_user`, and that there is no alias for the key `no_user`.

AUTHOR

`praliases` was developed by the University of California, Berkeley.

FILES

`/usr/lib/aliases`	default alias file
`/usr/lib/aliases.dir`	default alias database
`/usr/lib/aliases.pag`	

SEE ALSO

sendmail(1M).

NAME

prealloc - preallocate disk storage

SYNOPSIS

`prealloc` *name size*

DESCRIPTION

`prealloc` preallocates at least *size* bytes of disk space for an ordinary file *name*, creating the file if *name* does not already exist. The space is allocated in an implementation-dependent fashion for fast sequential reads and writes of the file.

`prealloc` fails and no disk space is allocated if *name* already exists and is not an ordinary file of zero length, if insufficient space is left on disk, or if *size* exceeds the maximum file size or the file size limit of the process (see *ulimit*(2)). The file is zero-filled.

DIAGNOSTICS

`prealloc` returns one of the following values upon completion:

0	Successful completion.
1	*name* already exists and is not an ordinary file of zero length.
2	There is insufficient room on the disk.
3	*size* exceeds file size limits.

EXAMPLES

The following example preallocates 50 000 bytes for the file **`myfile`**:

```
prealloc myfile 50000
```

WARNINGS

Allocation of file space is highly dependent on current disk usage. A successful return does not indicate how fragmented the file actually might be if the disk is approaching its capacity.

AUTHOR

`prealloc` was developed by HP.

SEE ALSO

prealloc(2), ulimit(2).

NAME

printenv - print out the environment

SYNOPSIS

printenv [*name*]

DESCRIPTION

printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

RETURN VALUE

If a *name* is specified and it is not defined in the environment, **printenv** returns 1; otherwise it returns zero.

SEE ALSO

sh(1), environ(5), csh(1).

NAME

printf - format and print arguments

SYNOPSIS

`printf` *format* [*arg* ...]

DESCRIPTION

`printf` writes formatted arguments to the standard output. The *arg* arguments are formatted under control of the *format* operand.

format is a character string patterned after the formatting conventions of *printf*(3C), and contains the following types of objects:

characters Characters that are not *escape sequences* or *conversion specifications* (as described below) are copied to standard output.

escape sequences These are interpreted as non-graphic characters:

`\a`	alert
`\b`	backspace
`\c`	print line without appending a new-line
`\f`	form-feed
`\n`	new-line
`\r`	carriage return
`\t`	tab
`\v`	vertical tab
`\'`	single quote character
`\\`	backslash
\n	the 8-bit character whose ASCII code is the 1-, 2-, 3-, or 4-digit octal number *n*, whose first character must be a zero.

conversion specification
Specifies the output format of each argument (see below).

Arguments following *format* are interpreted as strings if the corresponding format is either `c` or `s`; otherwise they are treated as constants.

Conversion Specifications

Each conversion specification is introduced by the percent character `%`. After the `%` character, the following can appear in the sequence indicated:

flags Zero or more *flags*, in any order, which modify the meaning of the conversion specification. The flag characters and their meanings are:

`-`	The result of the conversion is left-justified within the field.
`+`	The result of a signed conversion always begins with a sign, `+` or `-`.
<space>	If the first character of a signed conversion is not a sign, a space character is prefixed to the result. This means that if the space flag and `+` flag both appear, the space flag is ignored.
`#`	The value is to be converted to an "alternate form". For `c`, `d`, `i`, `u`, and `s` conversions, this flag has no effect. For `o` conversion, it increases the precision to force the first digit of the result to be a zero. For `x` or `X` conversion, a non-zero result has `0x` or `0X` prefixed to it. For `e`, `E`, `f`, `g`, and `G` conversions, the result always contains a radix character, even if no digits follow the radix character. For `g` and `G` conversions, trailing zeros are not removed from the result, contrary to usual behavior.

field width
An optional string of decimal digits to specify a minimum *field width*. For an output field, if the converted value has fewer characters than the field width, it is padded on the left (or right, if the left-adjustment flag, `-` has been given) to the field width.

precision
The *precision* specifies the minimum number of digits to appear for the `d`, `o`, `i`, `u`, `x`, or `X` conversions

(the field is padded with leading zeros), the number of digits to appear after the radix character for the `e` and `f` conversions, the maximum number of significant digits for the `g` conversion, or the maximum number of characters to be printed from a string in s conversion. The precision takes the form of a period `.` followed by a decimal digit string. A null digit string is treated as a zero.

conversion characters
A *conversion character* indicates the type of conversion to be applied:

`d`,`i`, `o`,`u`, `x`,`X`	The integer argument is printed a signed decimal (`d` or `i`), unsigned octal (`o`), unsigned decimal (`u`), or unsigned hexadecimal notation (`x` and `X`). The `x` conversion uses the numbers and letters `0123456789abcdef`, and the `X` conversion uses the numbers and letters `0123456789ABCDEF`. The *precision* component of the argument specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it is expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of 0 is no characters.
`f`	The floating-point number argument is printed in decimal notation in the style `[-]`*ddd* `r`*ddd*, where the number of digits after the radix character, `r`, is equal to the *precision* specification. If the *precision* is omitted from the argument, six digits are output; if the *precision* is explicitly 0, no radix appears.
`e`,`E`	The floating-point-number argument is printed in the style `[-]`*d* `r`*ddd* `e`±*dd*, where there is one digit before the radix character, and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is 0, no radix character appears. The `E` conversion character produces a number with `E` introducing the exponent instead of `e`. The exponent always contains at least two digits. However, if the value to be printed requires an exponent greater than two digits, additional exponent digits are printed as necessary.
`g`,`G`	The floating-point-number argument is printed in style `f` or `e` (or int style `E` in the case of a `G` conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted; style `e` is used only if the exponent resulting from the conversion is less than `-h` or greater than or equal to the precision. Trailing zeros are remove from the result. A radix character appears only if it is followed by a digit.
`c`	The first character of the argument is printed.
`s`	The argument is taken to be a string, and characters from the string are printed until the end of the string or the number of characters indicated by the *precision* specification of the argument is reached. If the precision is omitted from the argument, it is interpreted as infinite and all characters up to the end of the string are printed.
`%`	Print a `%` character; no argument is converted.
`b`	Similar to the `s` conversion specifier, except that the string can contain backslash-escape sequences which are then converted to the characters they represent.

In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of *arg* as single and/or multi-byte characters.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, *printf* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`printf` exits with one of the following values:

0 Successful completion;

>0 Errors occurred. The exit value is increased by one for each error that occurred up to a maximum of 255.

DIAGNOSTICS

If an argument cannot be converted into a form suitable for the corresponding conversion specification, or for any other reason cannot be correctly printed, a diagnostic message is printed to standard error, the argument is output as a string form as it was given on the command line, and the exit value is incremented.

EXAMPLES

The following command prints the number 123 in octal, hexadecimal and floating point formats in their alternate form

```
printf "%#o, %#x, %#X, %#f, %#g, %#e\n" 123 123 123 123 123 123
```

resulting in the following output

```
0173, 0x7b, 0X7B, 123.000000, 123.000, 1.230000e+02
```

Print the outputs with their corresponding field widths and precision:

```
printf "%.6d, %10.6d, %.6f, %.6e, %.6s\n" 123 123 1.23 123.4 MoreThanSix
```

resulting in the following output

```
000123,     000123, 1.230000, 1.234000e+02, MoreTh
```

SEE ALSO

echo(1), printf(3S).

STANDARDS CONFORMANCE

`printf`: POSIX.2

NAME

prmail - print out mail in the incoming mailbox file

SYNOPSIS

`prmail` [*user* ...]

DESCRIPTION

`prmail` prints the mail which waits for you or the specified user in the incoming mailbox file. The mailbox file is not disturbed.

`prmail` is functionally similar to the command:

```
cat /usr/mail/mailfile | more
```

or

```
cat /usr/mail/mailfile | pg
```

depending upon the setting of the user's `PAGER` environment variable

FILES

`/usr/mail/*`	incoming mailbox files

AUTHOR

`prmail` was developed by the University of California, Berkeley.

SEE ALSO

from(1), mail(1).

NAME

prof - display profile data

SYNOPSIS

`prof [-tcan][-ox][-g][-z][-h][-s][-m` *mdata* `][`*prog*`]`

DESCRIPTION

`prof` interprets a profile file produced by `monitor()` (see *monitor*(3C)). The symbol table in the object file *prog* (`a.out` by default) is read and correlated with a profile file (`mon.out` by default). For each external text symbol, the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options `t`, `c`, `a`, and `n` determine the type of sorting of the output lines:

`-t`	Sort by decreasing percentage of total time (default).
`-c`	Sort by decreasing number of calls.
`-a`	Sort by increasing symbol address.
`-n`	Sort by symbol name in ascending collation order (see Environment Variables below).

The mutually exclusive options `o` and `x` specify the printing of the address of each symbol monitored:

`-o`	Print each symbol address (in octal) along with the symbol name.
`-x`	Print each symbol address (in hexadecimal) along with the symbol name.

The following options can be used in any combination:

`-g`	Include non-global symbols (static functions).
`-z`	Include all symbols in the profile range (see *monitor*(3C)), even if associated with zero number of calls and zero time.
`-h`	Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)
`-s`	Print a summary of several of the monitoring parameters and statistics on the standard error output.
`-m` *mdata*	Use file *mdata* instead of `mon.out` as the input profile file.

A program creates a profile file if it has been loaded using the `cc -p` option (see *cc*(1)). This option to the `cc` command arranges for calls to `monitor()` at the beginning and end of execution (see *monitor*(3C)). It is the call to the `monitor` command at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the `-p` option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable `PROFDIR`. If `PROFDIR` does not exist, `mon.out` is produced in the directory current when the program terminates. If `PROFDIR=string`, `string/pid.progname` is produced, where *progname* consists of argv[0] with any path prefix removed, and *pid* is the program's process ID. If `PROFDIR=nothing`, no profiling output is produced.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating order output by the `-n` option.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `prof` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

WARNINGS

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely, however.

Only programs that call **`exit()`** (see *exit*(2)) or return from **`main`** cause a profile file to be produced, unless a final call to **`monitor()`** is explicitly coded.

The use of the `cc -p` option to invoke profiling imposes a limit of 600 functions that can have call counters established during program execution. For more counters, call `monitor()` directly. If this limit is exceeded, other data is overwritten and the `mon.out` file is corrupted. The number of call counters used is reported automatically by the `prof` command whenever the number exceeds 5/6 of the maximum.

FILES

`mon.out`	for profile
`a.out`	for namelist

SEE ALSO

cc(1), exit(2), profil(2), crt0(3), end(3C), monitor(3C).

STANDARDS CONFORMANCE

`prof`: SVID2, XPG2

NAME

protogen - ANSI C function prototype generator

SYNOPSIS

`protogen` [*options*] *files*

DESCRIPTION

`protogen` is an experimental tool that helps convert old style C code to ANSI C by generating prototypes for function declarations. `protogen` does not expand macros or remove `#ifdef` preprocessor directives; it only alters the declarations of functions. This approach retains the original form and content as much as possible. However, it also imposes restrictions on the file being parsed. The restrictions are:

- The program compiles with no errors.
- Macro expansions cannot contain braces, parentheses, or semicolons. `protogen` uses these symbols to determine where functions begin and end. For example, a program that uses the following macros will not be prototyped correctly.

  ```
  #define BEGIN {
  #define END   }
  ```

- No preprocessor directives are allowed within a function declaration. For example, the following function will not be prototyped and a warning will be issued.

  ```
  foo(a,b
  #ifdef EXTENDPARAMS
  ,c,d
  #endif
  )
  char a,b
  #ifdef EXTENDPARAMS
  ,c,d
  #endif
  ;
  ```

- No more than one macro is used as a type specifier in a declaration.
  For example, the following function will not be prototyped.

  ```
  #define EXTERNTYPE extern int
  #define SPECIALTYPE register auto
  EXTERNTYPE SPECIALTYPE foo(){}
  ```

- Programs that use erratic macro substitutions will not be prototyped correctly. For example, when prototyping the following:

  ```
  #define X int foo
  #define Y a,b
  X(Y){ ...
  ```

`protogen` produces:

```
int X(int Y){...
```

Code that violates any of these assumptions can be preprocessed by `cc` using the `-P` option before being prototyped (see *cc*(1)). Use of `protogen` on such code may cause the offending function to be ignored and no prototype generated, the issuance of a grammar conflict message or other warning, the incorrect prototype to be generated, or, when extremely erratic macro substitutions are used, `protogen` may produce unpredictable results. In any case, `protogen` does not alter the original C source file.

When invoking `protogen`, use the desired options and then a list of files to be prototyped. `protogen` then prototypes each file individually, and appends all prototypes to a common file specified by the `-h` option or to the default file `prototypes.h`. All files containing functions that do not have prototypes result in the creation of a new file with the functions prototyped. Thus, if the following files need new prototype declarations, their resulting modifications would appear under the result file name in the current directory:

Source File		Result File
filename.c	→	*filename*.a.c
filename.h	→	*filename*.a.h
filename	→	*filename*.a.c

If `protogen` encounters an `#include` directive and the file name is enclosed in angle brackets it is considered a system file and no modified result file is produced. The file is scanned for `#defines` and `#ifdef` directives. If, however, the file name is enclosed in double quotes, it is scanned for prototype generation. If new prototypes are added, the modifications are produced in a result file following the same naming conventions.

Once the new result files have been created, each should be compared to the original using the `diff` command (see *diff*(1)). Manual inspection of each file should show that only function declarations have been altered. If any of the above assumptions where violated, they show up in the results of the `diff` operation. Next the `prototypes.h` file should be included in the proper location of each result file and compiled using the ANSI C compiler. Once all files have been checked and compile successfully, they should be renamed and passed through `protogen` if other functions inside `#if` sections still exist.

Options

`protogen` recognizes the following options:

-D *name=def*	
-D *name*	Defines name as if by a `#define` preprocessor directive (see *cpp*(1)).
-H *nnn*	Change cpp's internal macro definition table to be *nnn* bytes in size. The macro symbol table is increased proportionately. The default table size is at least 128,000 bytes. This option serves to eliminate `too many defines` and `too much defining` errors.
-I *dir*	Changes the algorithm for searching for `#include` files whose names do not begin with `/` to look in *dir* before looking in the standard directories. See *cpp*(1) for more information on the `-I` option.
-U *name*	Removes any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular processor. See *cpp*(1) for a list of possible predefined symbols.
-h *file*	Specifies header output file for prototypes. If this option is not given, the default output file `prototypes.h` is used.
-w	Forces all prototype definitions to be widened to the standard C default promotions. All char, unsigned char, short, and unsigned short declarations are promoted to int. All floats are converted to double.

EXAMPLES

Let this be a sample C program to be prototyped, where `NODE`, `RecordTypeHP300`, and `RecordTypeOther` are defined in `def.h`.

```
/*********************************/
/* File: sample.c */
/*********************************/
#include "def.h"
#ifdef HP300
NODE *foo(a,b,c)
RecordTypeHP300 c;
#else
NODE *foo(a,b,c)
RecordTypeOther c;
#endif
{/* function statements */}

foo2(a,b)
long a;
char *b;
{/* function statements */}
```

Since there are two possible paths through the program, depending on the existence of the define HP300, `protogen` requires two passes to prototype the program:

```
$protogen -D HP300 -h sample.h sample.c
prototyping sample.c
no change   def.h
generating  sample.a.c
```

`protogen` has now prototyped the first declaration of `foo` and `foo2`. The results are placed in `sample.a.c`. The source file and result file should then be compared using the `diff` command and manually inspected. After checking the contents of the file, it should be moved to a temporary file and prototyped for the second path.

```
$mv sample.a.c temp.c
$protogen -U HP300 -h sample.h temp.c
prototyping temp.c
no change def.h
generating temp.a.c
```

The contents of `temp.a.c`:

```
/********************************/
/* File: sample.c */
/********************************/
#include "def.h"
#ifdef HP300
NODE *foo(int a,
    int b,
    RecordTypeHP300 c )
#else
NODE *foo(int a,
    int b,
    RecordTypeOther c )
#endif
    {/* function statements */}
int foo2(long a ,
    char *b )
    {/* function statements */}
```

After inspecting the file for differences, the prototype header file should be inserted into the apropiate location of the source file. Line 4 would be the proper position for it in `temp.a.c`. Next, look at the header file.

The contents of sample.h:

```
#if defined(HP300)
/*
Options -DHP300
*/

NODE *foo(int a,
    int b,
    RecordTypeHP300 c );

int foo2(long a ,
    char *b );

#endif

#if !defined(HP300)
/*
Options -UHP300
*/

NODE *foo(int a,
    int b,
        RecordTypeOther c );

    int foo2(long a ,
        char *b );

    #endif
```

Note that the include file has two sections for each run. Only the section with the matching defines is used. This allows easy conversion of programs that require several passes through `protogen`. However, it also causes multiple declarations of the same function as with `foo2()`. Therefore, it might be more desirable to place the function prototypes in manually.

Once the prototypes are in place, try to compile the new source with the ANSI C compiler. When the program compiles and executes properly, back up the old source and replace it with the new ANSI source code.

Common Problems When adding Prototypes

For large programs, a common include file of all the prototypes decreases abstraction between files and may introduce naming conflicts. Many definitions and types may also have to be added to the include file in order for the prototypes to be valid. It is therefore very important to prototype programs together only if they make several calls to each other or in closely related groups. Use `protogen` to generate special prototype header files for programs that are in different groups but make frequent calls to a common group of functions. In most cases it is desirable to manually replace all empty declarations with their new prototypes and not introduce any new header files.

The `-w` option can be used to widen parameters to their default promotions in function declarations. It is necessary to remove empty declarations for functions that do not use default promotion types and were not prototyped with the `-w` option. If any such declarations do exist or they use typedefs that contain nonstandard promoted types, the following error message from `cc` is issued:

```
function prototype for func-name must contain
parameters compatible with default argument
promotions when used with an empty declaration
```

`protogen` is simply a scanner and performs no semantical analysis. It does not look at typedefs and does not alter them when given the `-w` widen option. Typedefs that need to be widened, must be done manually.

Functions that have variable arguments must be fixed manually, using the ellipsis notation.

DIAGNOSTICS

Error messages are sent to standard error, and report warnings and grammar conflicts. Warnings are intended to be self-explanatory. Grammar conflicts occur due to violations in the predefined assumptions `protogen` was designed for. If a grammar conflict occurs, try preprocessing.

SEE ALSO

cpp(1), cc(1).

NAME

prs - print and summarize an SCCS file

SYNOPSIS

`prs` [`-d`[*dataspec*]] [`-r`[*SID*]] [`-e`] [`-l`] [`-c`] [`-a`] *file* ...

DESCRIPTION

`prs` prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user-supplied format. If a directory is named, `prs` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`), and unreadable files are silently ignored. If a name of `-` is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to `prs`, which can appear in any order, consist of options and file names.

All the described options apply independently to each named file:

`-d`[*dataspec*] Used to specify the output data specification. *dataspec* is a string consisting of SCCS file *data keywords* (see Data Keywords below) interspersed with optional user-supplied text.

`-r`[*SID*] Used to specify the *S*CCS *ID*entification (*SID*) string of a delta for which information is desired. If no *SID* is specified, the *SID* of the most recently created delta is assumed. If an *SID* is specified, it must agree exactly with an *SID* in the file (that is, the *SID* structure used by `get` does not work here — see *get*(1)).

`-e` Requests information for all deltas created *earlier* than and including the delta designated via the `-r` option or the date given by the `-c` option.

`-l` Requests information for all deltas created *later* than and including the delta designated via the `-r` option or the date given by the `-c` option.

`-c`[*cutoff*] Cutoff date-time, in the form

YY[MM[DD[HH[MM[SS]]]]]

Units omitted from the date-time default to their maximum possible values. Thus, `-c7502` is equivalent to `-c750228235959`. One or more non-numeric characters can be used to separate the various 2-digit segments of the cutoff date (for example `-c77/2/2 9:22:25`).

`-a`

Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, that is, delta type = *D*, deltas. If the `-a` option is not specified, information is provided for existing deltas only.

If no option letters (or only `-a`) are given, `prs` prints the file name using the default *dataspec* and the `-e` option. This produces information on all deltas.

Data Keywords

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword can appear in a *dataspec*.

The information printed by `prs` consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. Escapes can be used as follows:

backslash	\\	form feed	\f
backspace	\b	new-line	\n
carriage-return	\r	single quote	\′
colon	\:	tab	\t

The default *dataspec* is:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

SCCS File Data Keywords

Keyword	Data Item	File Section	Value	Fmt
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	**:Li:/:Ld:/:Lu:**	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	**:R:.:L:.:B:.:S:**	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	**:Dy:/:Dm:/:Dd:**	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	**:Th:::Tm:::Ts:**	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq # of deltas incl, excl, ign	"	**:Dn:/:Dx:/:Dg:**	S
:Dn:	Deltas included (seq #)	"	**:DS: :DS: ...**	S
:DB:	Deltas excluded (seq #)	"	**:DS: :DS: ...**	S
:Dg:	Deltas ignored (seq #)	"	**:DS: :DS: ...**	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User name	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	**:R: ...**	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	**:R:**	S
:CB:	Ceiling boundary	"	**:R:**	S
:Ds:	Default SID	"	**:I:**	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	**:Z::M:\t:I:**	S
:A:	A form of *what*(1) string	N/A	**:Z::Y: :M: :I::Z:**	S
:Z:	*what*(1) string delimiter	N/A	**@(#)**	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

*** :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:**

If no option letters (or only `-a`) are given, `prs` prints the file name, using the default *dataspec*, and the `-e` option; thus, information on all deltas is produced.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of dataspec as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `prs` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

DIAGNOSTICS

Use *help*(1) for explanations.

EXAMPLES

```
prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file
```

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case* (when no specifications for selecting or printing are given)

```
prs s.file
```

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only option argument allowed to be used with the *special case* is the `-a` option.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).
SCCS User's Guide in *Programming on HP-UX*.

STANDARDS CONFORMANCE

`prs`: SVID2, XPG2, XPG3

NAME

ps, cps - report process status

SYNOPSIS

ps [**-edafl**] [**-t** *termlist*] [**-p** *proclist*] [**-u** *uidlist*] [**-g** *grplist*]

cps [**-edafl**] [**-t** *termlist*] [**-p** *proclist*] [**-u** *uidlist*] [**-g** *grplist*]

DESCRIPTION

ps prints certain information about active processes. If not options are specified, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

In the HP Clustered environment, **cps** can be used to obtain a cluster wide process listing. A separate listing is produced for each member of the cluster, preceded by the cluster member's name. **cps** only reports on other members of the HP Cluster when the **e**, **d**, **a**, **t**, **p**, **u**, or **g** options have been specified.

options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

Options

ps and **cps** recognize the following options:

-e Print information about all processes.

-d Print information about all processes, except process group leaders.

-a Print information about all processes, except process group leaders and processes not associated with a terminal.

-f Generate a *full* listing. (See below for meaning of columns in a full listing.)

-l Generate a *long* listing. See below.

-t *termlist* Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers can be specified in one of two forms: the device's file name (such as **tty04**) or if the device's file name starts with **tty**, just the digit identifier (such as **04**).

-p *proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.

-u *uidlist* Restrict listing to data about processes whose real user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID is printed unless the **-f** option is used, in which case the login name is printed.

-g *grplist* Restrict listing to data about processes whose process group leaders are given in *grplist*.

The column headings and the meaning of the columns in a **ps** listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear. **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:

0	swapped;
1	in core;
2	system process;
4	locked in core (e.g., for physical I/O);
10	being traced by another process;
20	another tracing flag;

S (l)
The state of the process:

0	non-existent;
S	sleeping;
W	waiting;
R	running;

I	intermediate;
Z	terminated;
T	stopped;
X	growing.

`UID` (f,l)
The real user ID number of the process owner; the login name is printed under the **-f** option.

`PID` (all)
The process ID of the process; it is possible to kill a process if you know this datum.

`PPID` (f,l)
The process ID of the parent process.

`C` (f,l)
Processor utilization for scheduling.

`PRI` (l)
The priority of the process; higher numbers mean lower priority.

`NI` (l)
Nice value; used in priority computation.

`ADDR` (l)
The memory address of the process, if resident; otherwise, the disk address.

`SZ` (l)
The size in blocks of the core image of the process.

`WCHAN` (l)
The event for which the process is waiting or sleeping; if blank, the process is running.

`STIME` (f)
Starting time of the process. The starting date is printed instead if the elapsed time is greater than 24 hours.

`TTY` (all)
The controlling terminal for the process.

`TIME` (all)
The cumulative execution time for the process (reported in the form "min:sec").

`COMD` (all)
The command name; the full command name and its arguments are printed under the **-f** option. This field is renamed `COMMAND` except when the **-l** option is specified.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>** (see "zombie process" in *exit*(2)).

The time printed in the **STIME** field is the time when the process was forked, *not* the time when it was modified by **exec()**.

ps prints the command name (and arguments, if the **-f** option is specified) given at the time of process creation.

To make **ps** output safer to display and easier to read, all control characters in the **COMD** field are mapped to "visible" equivalents. These are of the form ^*C*, where the original character was in the range 0 through 037 and ^*C* is that value plus 040.

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, **ps** behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Generate a full listing of all processes currently running on your machine:

```
ps -ef
```

To check if a certain process exists on the machine such as `cron` (a clock daemon) for example, check the far right column for `cron`.

FILES

`/etc/passwd`	supplies UID information
`/etc/ps_data`	internal data structure
`/dev`	searched to find terminal ("tty") names

SEE ALSO

kill(1), nice(1), acctcom(1M), exec(2), fork(2), exit(2).

WARNINGS

Things can change while `ps` is running; the picture it gives is only a snapshot in time. Some data printed for defunct processes is irrelevant.

If two special files for terminals are located at the same select code, they are reported in the order in which they appear in `/dev`; not in alphabetical order.

The `-m` option, which was available in release 7.0, is no longer available. This option caused `ps` to print the command name and arguments that resided within the address space of the process.

The flags field printed out with the `-l` option no longer contains a flag indicating that a process is currently being swapped out. A process is now considered swapped out as soon as the swapping operation begins.

STANDARDS CONFORMANCE

`ps`: SVID2, XPG2, XPG3

(Requires Optional ALLBASE/SQL Software)

NAME

psqlc, psqlpas, psqlfor, psqlcbl - preprocess ALLBASE/SQL source programs written in C, Pascal, FORTRAN and COBOL

SYNOPSIS

C Preprocessing:

`psqlc -s` [`-i` *sourcefilename*] [`-p` *sqloutfilename*`.c`]
`psqlc` *DBEnvirnomentName* [`-o` *ownername*] [`-m` *modulename*] [`-d` [`-r`]]
[`-i` *sourcefilename*] [`-p` *sqloutfilename*`.c`]
`psqlc -D` [`-i` *sourcefilename*] [`-p` *sqloutfilename*`.c`]

Pascal Preprocessing:

`psqlpas -s` [`-i` *sourcefilename*] [`-p` *sqloutfilename*`.p`]
`psqlpas` *DBEnvirnomentName* [`-o` *ownername*] [`-m` *modulename*] [`-d` [`-r`]]
[`-i` *sourcefilename*] [`-p` *sqloutfilename*`.p`]
`psqlc -D` [`-i` *sourcefilename*] [`-p` *sqloutfilename*`.p`]

FORTRAN Preprocessing:

`psqlfor -s` [`-i` *sourcefilename*] [`-p` *sqloutfilename*`.f`]
`psqlfor` *DBEnvirnomentName* [`-o` *ownername*] [`-m` *modulename*] [`-d` [`-r`]]
[`-i` *sourcefilename*] [`-p` *sqloutfilename*`.f`]

COBOL Preprocessing:

`psqlcbl -s` [`-i` *sourcefilename*] [`-p` *sqloutfilename*`.cbl`]
`psqlcbl` *DBEnvirnomentName* [`-o` *ownername*] [`-m` *modulename*] [`-d` [`-r`]]
[`-i` *sourcefilename*] [`-p` *sqloutfilename*`.cbl`]

REMARKS

The ALLBASE/SQL product must be installed on the system before `psqlc`, `psqlpas`, `psqlfor`, or `psqlcbl` can be used.

DESCRIPTION

`psqlc`, `psqlpas`, `psqlfor`, and `psqlcbl` invoke the C, Pascal, FORTRAN, and COBOL preprocessors, respectively, for programmatically accessing an ALLBASE/SQL relational database environment (DBEnvironment). `psqlc`, `psqlpas`, `psqlfor`, and `psqlcbl` can be executed by all system users.

Options

`-s` Specify that the preprocessor checks only the syntax of embedded SQL commands.

`-D` Specify that all static SQL statements are to be converted to dynamic SQL statements. Dynamic statements remain dynamic and no module is created.

DBEnvironmentName
Identify the DBEnvironment in which a module is stored.

`-o` *ownername* Associate the stored module with a DBEUserID, a classname, or a groupname. You can specify an *ownername* for the module only if you have DBA authority in the DBEnvironment where the module is created, or if you are the current owner of the module. If not specified, the default *ownername* is your DBEUserID.

`-m` *modulename* Assign a name to the stored module. *modulenames* must follow the rules governing ALLBASE/SQL basic names as described in the *ALLBASE/SQL Reference Manual*. If no *modulename* is specified, the preprocessor uses the PROGRAM Statement name as the *modulename* or, for C programs, the source file name.

`-d` Delete any module currently stored in the DBEnvironment by the *modulename* and *ownername* specified in the options list. If this option is not specified, the module is retained and all existing run authorities for that module are preserved.

`-r` Revoke all existing run authorities for a module when a program being preprocessed already has a stored module. The `-r` option cannot be specified unless `-d` is also specified. If the `-r` option is not specified, all existing run authorities for that module are preserved.

`-i` *sourcefilename* Identify the name of the input file containing the source code being preprocessed. If *sourcefilename* is not specified, a default file `sqlin` is assumed. It is recommended but not required that the source file name have a filename suffix of `.sql`.

`-p` *sqloutfilename*	Identify the name of the output file containing the preprocessor generated code. If *sqloutfilename* is not specified, the preprocessor generated code is written to a file with the same name as the source file name but with an appended filename suffix indicating the language of the source code: `.c` (C), `.p` (Pascal), `.f` (FORTRAN), or `.cbl` (COBOL).

EXTERNAL INFLUENCES

Environment Variables

For preprocessors **psqlc**, **psqlpas**, and **psqlfor**, LANG determines the language in which messages are displayed.

International Code Set Support

For preprocessors **psqlc**, **psqlpas**, and **psqlfor**, single- and multi-byte character code sets are supported.

DEPENDENCIES

Series 300/400

The **psqlcbl** preprocessor is not currently supported.

AUTHOR

psqlc, **psqlpas**, **psqlfor**, and **psqlcbl** were developed by HP.

FILES

/usr/lib/sqldaemon	Cleanup daemon program file.
/usr/bin/psqlc	C preprocessor program file.
/usr/bin/psqlpas	Pascal preprocessor program file.
/usr/bin/psqlfor	FORTRAN preprocessor program file.
/usr/bin/psqlcbl	COBOL preprocessor program file.
/usr/lib/hpsqlproc	ALLBASE/SQL program file.
/usr/lib/hpsqlcat	ALLBASE/SQL message catalog file.
/usr/lib/libsql.a	Run time routine library file.
/usr/lib/nls/$LANG/hpsqlcat	Localized ALLBASE/SQL message catalog file.

SEE ALSO

ALLBASE/SQL C Application Programming Guide.
ALLBASE/SQL Pascal Application Programming Guide.
ALLBASE/SQL FORTRAN Application Programming Guide.
ALLBASE/SQL COBOL Application Programming Guide.
ALLBASE/SQL COBOL Release F.0 Application Programming Bulletin for HP-UX.

NAME

ptx - permuted index

SYNOPSIS

`ptx` [*options*] [*input* [*output*]]

DESCRIPTION

`ptx` generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted (see *sort*(1) and Environment Variables below). Finally, the sorted lines are rotated so the keyword comes at the middle of each line. `ptx` output is in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where `.xx` is assumed to be an `nroff` or `troff` macro provided by the user, or provided by the `mptx` macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

`-f`	Fold uppercase and lowercase letters for sorting.
`-t`	Prepare the output for the phototypesetter by using a line length of 100.
`-w` *n*	Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for `nroff` and 100 for `troff`.
`-g` *n*	Use the next argument, *n*, as the number of characters that `ptx` will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.
`-o` *only*	Use as keywords only the words given in the *only* file.
`-i` *ignore*	Do not use as keywords any words given in the *ignore* file. If the `-i` and `-o` options are missing, use `/usr/lib/eign` as the *ignore* file.
`-b` *break*	Use the characters in the *break* file to separate words. Tab, new-line, and space characters are *always* used as break characters. Punctuation characters are treated as part of the word in the absence of this option.
`-r`	Take any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

LC_CTYPE determines the default break characters.

If **LC_COLLATE** or **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `ptx` behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single-byte character code sets are supported.

WARNINGS

Line length counts do not account for overstriking or proportional spacing.

Lines containing tildes (~) are botched because `ptx` uses that character internally.

FILES

```
/usr/lib/eign
/bin/sort
```

`/usr/lib/tmac/tmac.ptx`

SEE ALSO

nroff(1), mm(5).

NAME

pwd - working directory name

SYNOPSIS

`pwd [-H]`

DESCRIPTION

`pwd` prints the path name of the working (current) directory. The `-H` option causes `pwd` to reveal hidden directories (context-dependent files) in the path name and append a plus sign (+) to them. See *cdf*(4).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `pwd` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`Cannot open ..`
`Read error in ..`
Possible file system trouble; contact system administrator.

`pwd: cannot access parent directories`
Current directory has been removed (usually by a different process). Use `cd` command to move to a valid directory (see *cd*(1)).

EXAMPLES

This command lists the path of the current working directory. If your home directory were `/mnt/staff` and the command `cd camp/nevada` were executed from the home directory, typing `pwd` would produce the following:

```
/mnt/staff/camp/nevada
```

AUTHOR

`pwd` was developed by AT&T and HP.

SEE ALSO

cd(1), cdf(4).

STANDARDS CONFORMANCE

pwd: SVID2, XPG2, XPG3, POSIX.2

NAME

pwget, grget - get password and group information

SYNOPSIS

`pwget` [`-n` *name* | `-u` *uid*]

`grget` [`-n` *name* | `-g` *gid*]

DESCRIPTION

`pwget` and `grget` locate and display information from `/etc/passwd` and `/etc/group`.

The standard output of `pwget` contains lines of colon-separated password information whose format is the same as that used in the `/etc/passwd` file (see *passwd*(4)).

The standard output of `grget` contains lines of colon-separated group information whose format is the same as that used in the `/etc/group` file (see *group*(4)).

With no options, `pwget` and `grget` get all entries with `getpwent()` or `getgrent()` respectively, (see *getpwent*(3C) and *getgrent*(3C)), and output a line for each entry found.

Options

When an option is given, only a single entry is printed.

The options for *pwget* are:

`-n` *name*	Output the first entry that matches *name* using `getpwnam()` (see *getpwent*(3C)).
`-u` I *Ouid*	Output the first entry that matches *uid* using `getpwuid()` (see *getpwent*(3C)).

The options for `grget` are:

`-n` *name*	Output the first entry that matches *name* using `getgrnam()` (see *getgrent*(3C)).
`-g` *gid*	Output the first entry that matches *gid* using `getgrgid()` (see *getgrent*(3C)).

NETWORKING FEATURES

NFS

If Network Information System (NIS) is in use, these commands provide password and group information based on the NIS version of the password and group databases in addition to the local password and group files.

RETURN VALUE

These commands return 0 upon success, 1 when a specific search fails, and 2 upon error.

DEPENDENCIES

NFS:

WARNING: If the Network Information System network database is in use and the NIS client daemon (`ypbind`) is not bound to a NIS server daemon (see *ypserv*(1M)), these utilities will wait until such a binding is established. These commands can be terminated in this condition by sending a `SIGINT` signal to the process (see *kill*(1)).

See *ypmatch*(1), and *ypserv*(1M).

AUTHOR

`pwget` and `grget` were developed by HP.

FILES

`/etc/group`	local group data file
`/etc/passwd`	local password data file

SEE ALSO

getgrent(3C), getpwent(3C), group(4), passwd(4).

NAME

quota - display disk usage and limits

SYNOPSIS

`quota` [`-v`] [*user*]...

DESCRIPTION

`quota` displays *user*'s disk usage and limits. Default for *user* is the name used when logging in. Only users with appropriate privileges can use the optional *user* argument to view the limits of other users.

If no options are specified, `quota` displays only warnings about mounted file systems where usage exceeds limits.

Options

`-v`	display *user*'s quota and usage statistics, whether they exceed limits or not. Note that no usage statistics exist if no quota is set.

AUTHOR

Disk Quotas were developed by the University of California, Berkeley, Sun Microsystems, Inc., and HP.

FILES

`/quotas`	quota file at the file system root
`/etc/mnttab`	list of currently mounted filesystems

SEE ALSO

quota(5).

NAME

ratfor - rational Fortran dialect

SYNOPSIS

`ratfor` [*options*] [*file* ...]

DESCRIPTION

`ratfor` converts a rational dialect of Fortran into ordinary irrational Fortran. `ratfor` provides control flow constructs essentially identical to those in C:

statement grouping:
```
    { statement; statement; statement }
```

decision-making:
```
    if (condition) statement [else statement]
    switch (integer value) {
          case  integer:          statement
            ...
         [default:]               statement
    }
```

loops:
```
    while (condition) statement
    for (expression; condition; expression) statement
    do limits statement
    repeat statement [until (condition) ]
    break
    next
```

and some syntactic sugar to make programs easier to read and write:

free form input:
  multiple statements per line and automatic continuation of lines

comments:
  # this is a comment.

compiler directives:
  directives beginning with a dollar sign (**$**) in column one are passed through to the compiler unchanged.

translation of relationals:
  >, **>=**, etc., become **.GT.**, **.GE.**, etc.

return expression to caller from function:
  `return` (expression)

define:
  `define` *name replacement*

include:
  `include` *file*

Options

The following options are supported:

- `-h` Cause quoted strings to be turned into Hollerith constructs as, for example, `6Hstring`.
- `-C` Copy comments to the output and format them neatly.
- `-6`*c* Normally, continuation lines are marked with an **&** in column 1. The `-6`*c* option defines *c* as the continuation character and places it in column 6.

`ratfor` is best used with `f77` (see *f77*(1)). Options can be passed to `ratfor` from `f77` by using the `f77 -Wr` command-line option.

SEE ALSO

f77(1).

B. W. Kernighan and P. J. Plauger, *Software Tools,* Addison-Wesley, 1976.

NAME

rcp - remote file copy

SYNOPSIS

Copy Single File:

`rcp` [`-p`] *source_file1 dest_file*

Copy Multiple Files:

`rcp` [`-p`] *source_file1* [*source_file2* ...] *dest_dir*

Copy One or More Directory Subtrees:

`rcp` [`-p`] -r *source_dir1* [*source_dir2* ...] *dest_dir*

Copy Files and Directory Subtrees:

`rcp` [`-p`] -r *file_or_dir1* [*file_or_dir2* ...] *dest_dir*

DESCRIPTION

`rcp` (remote copy) copies files, directory subtrees, or a combination of files and directory subtrees from one or more systems to another. In many respects, it is very similar to the `cp` command. (see *cp*(1).

The user must have read access to files being copied, and read and search (execute) permission on all directories in the directory path.

Command-Line Arguments

`rcp` recognizes the following command-line arguments:

source_file
source_dir
Name of existing file or directory on local or remote machine that is to be copied to destination. Source file and directory names are constructed as follows:

user_name@*hostname*:*pathname*/*filename*

or

user_name@*hostname*:*pathname*/*dirname*

Component parts of file and directory names are described below. If multiple existing files and/or directory subtrees are specified (*source_file1*, *source_file2*, ..., etc.), the destination must be a directory. Shell filename expansion is allowed on both local and remote systems. Multiple files and directory subtrees can be copied from one or more systems to a single destination directory with a single command.

dest_file
Name of destination file. If hostname and pathname are not specified, the existing file is copied into a file named *dest_file* in the current directory on the local system. If *dest_file* already exists and is writable, the existing file is overwritten. Destination filenames are constructed the same way as source files except that filename expansion characters cannot be used.

dest_dir
Name of destination directory. If hostname and pathname are not specified, the existing file is copied into a directory named *dest_dir* in the current directory on the local system. If *dest_dir* already exists in the specified directory path (or current directory if not specified), a new directory named *dest_dir* is created underneath the existing directory named *dest_dir*. Destination directory names are constructed the same way as source directory tree names except that filename expansion characters cannot be used.

file_or_dir
If a combination of files and directories are specified for copying (either explicitly or by filename expansion), only files are copied unless the `-r` option is specified. If the `-r` option is present, all files and directory subtrees whose names match the specified *file_or_dir* name are copied.

`-p`
Preserve (duplicate) modification times and modes (permissions) of source files, ignoring the current setting of the `umask` file creation mode mask. If this option is specified, `rcp` preserves the sticky bit only if the target user is super-user.

If the `-p` option is not specified, `rcp` preserves the mode and owner of *dest_file* if it already exists; otherwise `rcp` uses the mode of the source file modified by the `umask` on the destination host. Modification and access times of the destination file are set to the time when the copy was made.

`-r`
Recursively copy directory subtrees rooted at the source directory name. If any directory subtrees are to be copied, `rcp` recursively copies each subtree rooted at the specified source directory name to directory *dest_dir*. If *source_dir* is being copied to an existing directory of the same name, `rcp` creates a new directory *source_dir* within *dest_dir* and copies the subtree rooted at *source_dir* to *dest_dir*/*source_dir*. If *dest_dir* does not exist, `rcp` creates it and copies the subtree rooted at *source_dir* to *dest_dir*.

Constructing File and Directory Names

As indicated above, file and directory names contain one, two, or four component parts:

user_name	Login name to be used for accessing directories and files on remote system.
hostname	Hostname of remote system where directories and files are located.
pathname	Absolute directory pathname or directory pathname relative to the login directory of user *user_name*.
filename	Actual name of source or destination file. Filename expansion is allowed on source file names.
dirname	Actual name of source or destination directory subtree. Filename expansion is allowed on source directory names.

Each *file* or *directory* argument is either a remote file name of the form *hostname*:*path*, or a local file name (with a `/` before any `:`). *hostname* can be either an official host name or an alias (see *hosts*(4)). If *hostname* is of the form *ruser@rhost*, *ruser* is used on the remote host instead of the current user name. An unspecified *path* (i.e. *hostname*`:`) refers to the remote user's login directory. If *path* does not begin with `/`, it is interpreted relative to the remote user's login directory on *hostname*. Shell metacharacters in remote *paths* can be quoted with backslash (\\), single quotes (`' '`), or double quotes (`" "`), so that they will be interpreted remotely.

`rcp` does not prompt for passwords. The current local user name or any user name specified via *ruser* must exist on *rhost* and allow remote command execution via *remsh*(1) and *rcmd*(3). *remshd*(1M) must be executable on the remote host.

Third-party transfers in the form:

```
rcp ruser1@rhost1:path1 ruser2@rhost2:path2
```

are performed as:

```
remsh rhost1 -l ruser1 rcp path1 ruser2@rhost2:path2
```

Therefore, for a such a transfer to succeed, *ruser2* on *rhost2* must allow access by *ruser1* from *rhost1* (see *hosts.equiv*(4)).

WARNINGS

`rcp` is confused by any output generated by commands in a `.cshrc` file on the remote host (see *csh*(1)).

Copying a file to itself, for example:

```
rcp path 'hostname':path
```

may produce inconsistent results. The current HP-UX version of `rcp` simply copies the file over itself. However, some implementations of `rcp`, including some earlier HP-UX implementations, corrupt the file. In addition, the same file may be referred to in multiple ways, for example, via hard links, symbolic links, NFS, or in an HP-UX clustered environment. It is not guaranteed that `rcp` will correctly copy a file over itself in all cases.

Implementations of `rcp` based on the 4.2BSD version (including the implementations of `rcp` prior to HP-UX 7.0) require that remote users be specified as *rhost.ruser*. If the first remote host specified in a third party transfer (*rhost1* in the example below) uses this older syntax, the command must have the form:

```
rcp ruser1@rhost1:path1 rhost2.ruser2:path2
```

since the target is interpreted by *rhost1*. A common problem that is encountered is when two remote files are to be copied to a remote target that specifies a remote user. If the two remote source systems, *rhost1* and *rhost2*, each expect a different form for the remote target, the command:

```
rcp rhost1:path1 rhost2:path2 rhost3.ruser3:path3
```

will certainly fail on one of the source systems. Perform such a transfer using two separate commands.

AUTHOR

`rcp` was developed by the University of California, Berkeley.

SEE ALSO

cp(1), ftp(1), remsh(1), remshd(1M), rcmd(3), hosts(4), hosts.equiv(4).

`ftp` chapter in *Using ARPA Services*.

NAME

rcs - change RCS file attributes

SYNOPSIS

`rcs` [*options*] *file* ...

DESCRIPTION

`rcs` creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For `rcs` to work, the user's login name must be on the access list, except if the access list is empty, if the user is the owner of the file or the superuser, or if the `-i` option is present.

The user of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. `rcs` creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, `rcs` always creates a new file. On successful completion, `rcs` deletes the old one and renames the new one. This strategy makes links to RCS files useless.

Files ending in `,v` are RCS files; all others are working files. If a working file is given, `rcs` tries to find the corresponding RCS file first in directory `./RCS`, then in the current directory, as explained in *rcsintro*(5).

Options

`rcs` recognizes the following options:

`-a`*logins*	Appends the login names appearing in the comma-separated list *logins* to the access list of the RCS file.
`-A`*oldfile*	Appends the access list of *oldfile* to the access list of the RCS file.
`-c"`*string*`"`	Sets the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword `$Log$` during check out (see *co*(1)). This is useful for programming languages without multi-line comments. During `rcs -i` or initial `ci`, the comment leader is guessed from the suffix of the working file.
`-e`[*logins*]	Erases the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, the entire access list is erased.
`-i`	Creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, `rcs` tries to place it first into the subdirectory `./RCS`, then into the current directory. If the RCS file already exists, an error message is printed.
`-l`[*rev*]	Locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with `ci` or `rcs -u` (see below).
`-L`	Sets locking to `strict`. Strict locking means that the owner of an RCS file is not exempt from locking for check in. This option should be used for files that are shared.
`-n`*name*[`:`[*rev*]]	Associates the symbolic name *name* with the branch or revision *rev*. `rcs` prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic name is associated with the latest revision on the trunk. If `:`*rev* is omitted, the symbolic name is deleted.
`-N`*name*[`:`[*rev*]]	Same as `-n`, except that it overrides a previous assignment of *name*.
`-o`*range*	Deletes ("obsoletes") the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form *rev1-rev2* means revisions *rev1* to *rev2* on the same branch, *-rev* means from the beginning of the branch containing *rev* up to and including *rev*, and rev- means from revision *rev* to the head of the branch containing *rev*. None of the outdated revisions can have branches or locks.
`-q`	Quiet mode; diagnostics are not printed.
`-s`*state*[`:`*rev*]	Sets the state attribute of the revision *rev* to *state*. If *rev* is omitted, the latest revision on the trunk is assumed. If *rev* is a branch number, the latest revision on that

branch is assumed. Any identifier is acceptable for *state*. A useful set of states is `Exp` (for experimental), `Stab` (for stable), and `Rel` (for released). By default, `ci` sets the state of a revision to `Exp`.

`-t`[*txtfile*]
: Writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, `rcs` prompts the user for text supplied from the standard input, terminated with a line containing a single `.` or Ctrl-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the `-i` option is present, descriptive text is requested even if `-t` is not given. The prompt is suppressed if the standard input is not a terminal.

`-u`[*rev*]
: Unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is omitted, the latest lock held by the user is removed. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single `.` or Control-D.

`-U`
: Sets locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for check in. This option should *not* be used for files that are shared. The default (`-L` or `-U`) is determined by the system administrator.

Access Control Lists (ACLs)

Do not add optional ACL entries to an RCS file, because they are deleted when the file is updated. The resulting access modes for the new file might not be as desired.

DIAGNOSTICS

The RCS filename and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful; 1 if unsuccessful.

EXAMPLES

Add the names `jane`, `mary`, `dave`, and `jeff` to the access list of RCS file `vision,v`:

```
rcs -ajane,mary,dave,jeff vision
```

Set the comment leader to *tab* * for file `vision`:

```
rcs -c'tab*' vision
```

Associate the symbolic name `sso/6_0` with revision `38.1` of file `vision`:

```
rcs -Nsso/6_0:38.1 vision
```

Lock revision `38.1` of file `vision,v` so that only the locker is permitted to check in (see *ci*(1)) the next revision of the file. This command prevents two or more people from simultaneously revising the same file and inadvertently overwriting each other's work.

```
rcs -l38.1 vision,v
```

AUTHOR

`rcs` was developed by Walter F. Tichy>

SEE ALSO

co(1), ci(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4), acl(5), rcsintro(5).

NAME

rcsdiff - compareRCS revisions

SYNOPSIS

`rcsdiff` [**-bcefhn**] [`-r`*rev1*] [`-r`*rev2*] *file* ...

DESCRIPTION

`rcsdiff` compares two revisions of each given RCS file and creates output very similar to `diff` (see *diff*(1)). A file name ending in `,v` is an RCS file name, otherwise it is a working file name. `rcsdiff` derives the working file name from the RCS file name and vice versa, as explained in *rcsintro*(5). Pairs consisting of both an RCS and a working file name can also be specified.

`rcsdiff` recognizes the following options:

`-b` Same as described in *diff*(1);

`-e` Same as described in *diff*(1);

`-f` Same as described in *diff*(1);

`-h` Same as described in *diff*(1);

`-n` Generate an edit script of the format used by RCS.

`-c`[*n*]
Generate a diff with lines of context. The default is to present 3 lines of context. To change, specify *n*; for example, `-c10` gives 10 lines of context.

`-c` modifies the output format slightly from the normal *diff*(1) output. The "context" output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen `*` (asterisks). Lines removed from *file1* are marked with `-` (dashes); those added to *file2* with `+` (pluses). Lines that are changed from one file to the other are marked in both files with `!` (exclamation marks).

If both *rev1* and *rev2* are omitted, `rcsdiff` compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what was changed since the last check-in.

If *rev1* is given, but *rev2* is omitted, `rcsdiff` compares revision *rev1* of the RCS file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, `rcsdiff` compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* can be given numerically or symbolically.

EXAMPLES

Compare the latest trunk revision of RCS file `f.c,v` and the contents of working file `f.c`:

```
rcsdiff f.c
```

AUTHOR

`rcsdiff` was developed by Walter F. Tichy.

SEE ALSO

ci(1), co(1), diff(1), ident(1), rcs(1), rcsmerge(1), rlog(1), rcsfile(4), rcsintro(5).

NAME
rcsmerge - merge RCS revisions

SYNOPSIS
`rcsmerge -r`*rev1* [`-r`*rev2*] [`-p`] *file*

DESCRIPTION
`rcsmerge` incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If `-p` is given, the result is printed on the standard output; otherwise the result overwrites the working file.

A file name ending in `,v` is an RCS file name; otherwise it is a working file name. `rcsmerge` derives the working file name from the RCS file name and vice versa, as explained in *rcsintro*(5). A pair consisting of both an RCS and a working file name can also be specified.

rev1 cannot be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* can be given numerically or symbolically.

`rcsmerge` prints a warning if there are overlaps, and delimits the overlapping regions as explained for the `-j` option of *co*(1). The command is useful for incorporating changes into a checked-out revision.

EXAMPLES
Suppose you have released revision 2.8 of `f.c`. Assume furthermore that you just completed revision 3.4 when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file `f.c` and execute:

```
rcsmerge -p -r2.8 -r3.4 f.c >f.merged.c
```

Then examine `f.merged.c`. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute `co -j`:

```
ci -r2.8.1.1 f.c
co -r3.4 -j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in `f.c`:

```
rcsmerge -r2.8 -r2.4 f.c
```

Note the order of the arguments, and that `f.c` is overwritten.

WARNINGS
`rcsmerge` does not work for files that contain lines with a single `.`.

AUTHOR
`rcsmerge` was developed by Walter F. Tichy.

SEE ALSO
ci(1), co(1), merge(1), ident(1), rcs(1), rcsdiff(1), rlog(1), rcsfile(4).

NAME
read - read a line from standard input

SYNOPSIS
`read [-r] var ...`

DESCRIPTION
`read` reads a single line from standard input. The line is split into fields as when processed by the shell (refer to shells in SEE ALSO); the first field is assigned to the first variable *var*, the second field to the second variable *var*, and so forth. If there are more fields than there are specified *var* operands, the remaining fields and their intervening separators are assigned to the last *var*. If there are more *var*s than fields, the remaining *var*s are set to empty strings.

The setting of variables specified by the *var* operands affect the current shell execution environment.

Standard input to `read` can be redirected from a text file.

Since `read` affects the current shell execution environment, it is usually provided as a normal shell special (built-in) command. Thus, if it is called in a subshell or separate utility execution environment similar to the following, it does not affect the shell variables in the caller's environment:

```
(read foo)
nohup read ....
find . -exec read ... ;
```

Options and Arguments
`read` recognizes the following options and command-line arguments:

`-r`	Do not treat a backslash character in any special way. Consider each backslash to be part of the input line.
var	The name of an existing or non-existing shell variable.

EXTERNAL INFLUENCES
Environment Variables
`IFS` determines the internal field separators used to delimit fields.

RETURN VALUE
`read` exits with one of the following values:

0	Successful completion.
>0	End-of-file was detected or an error occurred.

EXAMPLES
Print a file with the first field of each line moved to the end of the line.

```
while read -r xx yy
do
        printf "%s %s \n" "$yy" "$xx"
done < input_file
```

SEE ALSO
csh(1), ksh(1), sh-posix(1), sh(1).

STANDARDS CONFORMANCE
`read`: SVID2, XPG2, XPG3, POSIX.2 FIPS

NAME

readmail - read mail from specified mailbox

SYNOPSIS

`readmail` [`-p`] [`-n`] [`-f` *filename*] [`-h`]
`readmail` [`-p`] [`-n`] [`-f` *filename*] [`-h`] *number-list*
`readmail` [`-p`] [`-n`] [`-f` *filename*] [`-h`] *pattern*

DESCRIPTION

`readmail` is a program that gives you the functionality of the *mailx*(1) `~r` command from the editor of your choice. There are three different ways to use the program.

- When creating a reply to a message from within *elm*(1), `readmail` without any arguments includes a summary of the headers and the body of the message being replied to. If the you are not currently editing a message, `readmail` returns an error.
- To include certain messages, specify them by listing their ordinal locations in the mail file (i.e., their "message numbers") up to 25 at a time. The meta-number `$` is understood to mean the last message in the mailfile. Similarly, `*` is understood to represent every message in the file (i.e., 1 2 3 4 5 ... $)
- Specify a pattern that occurs in one of the messages as a way of including it. This pattern can be typed in directly (no quotes) if the words are separated by a single space in the actual message. The pattern matching is uppercase/lowercase sensitive, so `Hello` and `hello` are not equivalent.

Other options are:

`-f` *folder*	Use the file specified for the operations specified instead of the incoming mailbox.
`-h`	Include the entire header of the matched message or messages when displaying their text (default is to display the `From:` `Date:` and `Subject:` lines only).
`-n`	Exclude all headers. This is used mostly for extracting files mailed and such.
`-p`	Put form-feeds (Control-L) between message headers. This is useful when printing sets of messages.

EXAMPLES

To use `readmail` from within `vi` (see *vi*(1)) to include the text of the current message at the end of the edit buffer, use the command:

```
!!readmail
```

(when you press the second `!`, the editor puts you at the bottom of the screen with the `!` prompt).

Define a *csh*(1) alias similar to:

```
alias rd 'readmail $ | page'
```

This can be used in conjunction with a program such `newmail` to peruse mail as it arrives without waiting for a mail system to start up (see *newmail*(1))

AUTHOR

`readmail` was developed by HP.

FILES

`/usr/mail/`*username*
incoming mailbox

`$HOME/.elm/readmail`
temporary file for *elm*(1)

SEE ALSO

elm(1), mailx(1), newmail(1), vi(1).

NOTE

For performance reasons when the program is given a list of message numbers to display, they are sorted into ascending order. Thus "1 3 2" produces the same output as "1 2 3".

NAME

remsh - execute from a remote shell

SYNOPSIS

remsh *host* [**-l** *username*] [**-n**] *command*
host [**-l** *username*] [**-n**] *command*

rexec *host* [**-l** *username*] [**-n**] *command*

DESCRIPTION

remsh connects to the specified *host* and executes the specified *command*. The host name can be either the official name or an alias as understood by **gethostbyname()** (see *gethostent*(3N) and *hosts*(4)). **remsh** copies its standard input (**stdin**) to the remote command, and the standard output of the remote command to its standard output (**stdout**), and the standard error of the remote command to its standard error (**stderr**). Hangup, interrupt, quit, terminate, and broken pipe signals are propagated to the remote command. **remsh** exits when the sockets associated with **stdout** and **stderr** of the remote command are closed. This means that **remsh** normally terminates when the remote command does (see *remshd*(1M)).

By default, **remsh** uses the following path when executing the specified *command*:

```
:/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin
```

remsh uses the default remote login shell with the **-c** option to execute the remote command. If the default remote shell is *csh*, csh sources the remote **.cshrc** file before the command. **remsh** cannot be used to run commands that require a terminal interface (such as **vi**) or commands that read their standard error (such as **more**). In such cases, use **rlogin** or **telnet** instead (see *rlogin*(1) and *telnet*(1)).

The remote account name used is the same as your local account name, unless you specify a different remote name with the **-l** option. This remote account name must be equivalent to the originating account; no provision is made for specifying a password with a command. For more details about *equivalent* hosts and how to specify them, see *hosts.equiv*(4). The files inspected by **remshd** on the remote host are **/etc/hosts.equiv** and **$HOME/.rhosts** (see *remshd*(1M)).

If *command,* is not specified, instead of executing a single command, you will be logged in on the remote host using **rlogin** (see *rlogin*(1)). Any **rlogin** options typed in the command line are transmitted to **rlogin**. If *command* is specified, options specific to **rlogin** are ignored by **remsh**.

By default, **remsh** reads its standard input and sends it to the remote command because **remsh** has no way to determine whether the remote command requires input. The **-n** option redirects standard input to **remsh** from **/dev/null**. This is useful when running a shell script containing a **remsh** command, since otherwise remsh may use input not intended for it. The **-n** option is also useful when running **remsh** in the background from a job control shell, **/bin/csh** or **/bin/ksh**. Otherwise, **remsh** stops and waits for input from the terminal keyboard for the remote command. **/bin/sh** automatically redirects its input from **/dev/null** when jobs are run in the background.

Host names for remote hosts can also be commands (linked to **remsh**) in the directory **/usr/hosts**. If this directory is specified in the **$PATH** environment variable, you can omit **remsh**. For example, if **remotehost** is the name of a remote host, **/usr/hosts/remotehost** is linked to **remsh**, and if **/usr/hosts** is in your search path, the command

```
remotehost command
```

executes **command** on **remotehost**, and the command

```
remotehost
```

is equivalent to

```
rlogin remotehost
```

The **rexec** command, a link to **remsh**, works the same as **remsh** except that it uses the **rexec()** library routine and **rexecd** for command execution (see *rexec*(3N) and *rexecd*(1M)). **rexec** prompts for a password before executing the command instead of using **hosts.equiv** for authentication. It should be used in instances where a password to a remote account is known but there are insufficient permissions for **remsh**.

EXAMPLES

Shell metacharacters that are not quoted are interpreted on the local host; quoted metacharacters are interpreted on the remote host. Thus the command line:

```
remsh otherhost cat remotefile >> localfile
```

appends the remote file `remotefile` to the local file `localfile`, while the command line

```
remsh otherhost cat remotefile ">>" otherremotefile
```

appends `remotefile` to the remote file `otherremotefile`.

If the remote shell is `/bin/sh`, the following command line sets up the environment for the remote command before executing the remote command:

```
remsh otherhost . .profile 2>&- \; command
```

The `2>&-` throws away error messages generated by executing `.profile` when *stdin* and *stdout* are not a terminal.

The following command line runs `remsh` in the background on the local system, and the output of the remote command comes to your terminal asynchronously:

```
remsh otherhost -n command &
```

The background `remsh` completes when the remote command does.

The following command line causes `remsh` to return immediately without waiting for the remote command to complete:

```
remsh otherhost -n "command 1>&- 2>&- &"
```

(See *remshd*(1M) and *sh*(1)). If your login shell on the remote system is *csh*, use the following form instead:

```
remsh otherhost -n "sh -c \"command 1>&- 2>&- &\""
```

RETURN VALUE

If `remsh` fails to set up the secondary socket connection, it returns 2. If it fails in some other way, it returns 1. If it fully succeeds in setting up a connection with `remshd`, it returns 0 once the remote command has completed. Note that the return value of `remsh` bears no relation to the return value of the remote command.

DIAGNOSTICS

Besides the errors listed below, errors can also be generated by the library functions `rcmd()` and `rresvport()` which are used by `remsh` (see *rcmd*(3N)). Those errors are preceded by the name of the library function that generated them. `remsh` can produce the following diagnostic messages:

`rlogin: ...`
: Error in executing `rlogin` (`rlogin` is executed when the user does not specify any commands to be executed). This is followed by the error message specifying why the execution failed.

`shell/tcp: Unknown service`
: The "shell" service specification is not present in the `/etc/services` file.

`Can't establish stderr`
: `remsh` cannot establish secondary socket connection for `stderr`.

<system call> `:` ...
: Error in executing system call. Appended to this error is a message specifying the cause of the failure.

`There is no entry for you (user ID` *uid*`) in /etc/passwd`
: Check with the system administrator to see if your entry in the password file has been deleted by mistake.

WARNINGS

For security reasons, the `/etc/hosts.equiv` and `.rhosts` files should exist, even if empty, and should be readable and writable only by the owner. Note also that all information, including any passwords asked for, is passed unencrypted between the two hosts.

If `remsh` is run with an interactive command it hangs.

DEPENDENCIES

`remsh` is the same service as `rsh` on BSD systems. The name was changed due to a conflict with the existing System V command `rsh` (restricted shell).

AUTHOR

`remsh` was developed by the University of California, Berkeley.

FILES

`/usr/hosts/*`
for version of the command invoked only with hostname

SEE ALSO

rlogin(1), remshd(1M), rexecd(1M), gethostent(3N), rcmd(3N), rexec(3N), hosts.equiv(4), hosts(4).

NAME

renice - alter priority of running processes

SYNOPSIS

`renice` [`-n` *priority_change*] [`-g`|`-p`|`-u`] *id* ...

DESCRIPTION

`renice` alters the system nice value (relative system scheduling priority) of one or more running processes specified by *id*. The new system nice value is equal to 20 + *priority_change*, and is limited to the range from 0 through 39. If *priority_change* is a negative value, priority is increased provided the user has appropriate privileges. See *nice*(1) for an explanation of system nice values.

Except for users with appropriate privileges, only the process owner can alter process priority, and can only monotonically increase the system nice value within the range 20 to 39 (this prevents overriding any current administrative restrictions). Users with appropriate privileges can alter the priority of any process and set the system nice value for the process to any value in the range 0 through 39 where 20 is the default system nice value for ordinary user processes. System nice values greater than 20 run at lower relative priority; values less than 20 run at higher relative priority.

Each specified *id* parameter is interpreted as a process ID, process group ID, or user name, depending on which option is used. Using `renice` on a process group causes all processes in that process group to have their scheduling priority altered. `renice` on a user changes the scheduling priority of all processes owned by that user. If no `-g`, `-p`, or `-g` option is specified, *id*s are interpreted as process IDs by default.

Options

`renice` recognizes the following options:

`-g`	Interpret each ID as a process group ID. Only users with appropriate privileges can use this option.
`-n` *increment*	Change the system nice value of each affected process to 20 + *increment*. If *increment* is negative, the system nice value is 20 minus the absolute value of *increment*. Only users with appropriate privileges can increase scheduling priority. If this option is not used, *priority_change* defaults to 10.
`-p`	Interpret each ID as a process ID. This is the default option if none is specified.
`-u`	Interpret each ID as a user name. Only users with appropriate privileges can use this option for user IDs other than their own.

RETURN VALUES

`renice` returns a 0 when successful, and a non-zero value when unsuccessful.

EXTERNAL INFLUENCES

Single-byte character code sets are supported.

DIAGNOSTICS

`renice` reports the old and new priority (system nice value) of the affected process(es) if the operation requested completed successfully. Otherwise, an error message is displayed to indicate the reason for failure.

EXAMPLES

Use `renice` default values to decrease the priority of process `923`. `renice` defaults to `-p` option, and *priority_change* defaults to `10`, setting the process to a system nice value of 30:

```
renice 923
```

Change the system nice value for all processes owned by user `john` and user `123` to 33 (*priority_change*=13):

```
renice -n 13 -u john 123
```

Change the system nice value of all processes in process group 20 to `10` (lowering the system nice value requires appropriate privileges).

```
renice -n -10 -g 20
```

WARNINGS

Users who do not have the appropriate privileges cannot increase scheduling priorities of their own processes, even if they were the one who decreased the priority in the first place.

FILES

`/etc/passwd`	to map user names to user ID's

SEE ALSO

nice(1), getpriority(2), nice(2).

NAME

rev - reverse lines of a file

SYNOPSIS

rev [*file*] ...

DESCRIPTION

rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, *rev* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

rlog - print log messages and other information on RCS files

SYNOPSIS

`rlog` [*options*] *file* ...

DESCRIPTION

`rlog` prints information about RCS files. Files ending in `,v` are RCS files; all others are working files. If a working file is given, `rlog` tries to find the corresponding RCS file first in directory `./RCS`, then in the current directory, as explained in *rcsintro*(5).

`rlog` prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, `rlog` prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, `rlog` prints complete information. The options below restrict this output.

Options

`rlog` recognizes the following options:

`-d`*dates*	Print information about revisions whose check-in date and time fall within the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2* (inclusive). A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d*, *d1*, and *d2* are in the free format explained in *co*(1). Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.
`-h`	Print only RCS file name, working file name, head, access list, locks, symbolic names, and suffix.
`-l`[*lockers*]	Print information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.
`-L`	Ignore RCS files that have no locks set; convenient in combination with `-R`, `-h`, or `-l`.
`-r`*revisions*	Print information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1-rev2* means revisions *rev1* to *rev2* on the same branch, *-rev* means revisions from the beginning of the branch up to and including *rev*, and *rev*- means revisions starting with *rev* to the head of the branch containingw *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.
`-R`	Print only the name of the RCS file; convenient for translating a working file name into an RCS file name.
`-s`*states*	Print information about revisions whose state attributes match one of the states given in the comma-separated list *states*.
`-t`	Print the same as `-h`, plus the descriptive text.
`-w`[*logins*]	Prints information about revisions checked in by users whose login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

`rlog` prints the intersection of the revisions selected with the options `-d`, `-l`, `-s`, `-w`, and `-r`.

EXAMPLES

Print the names of all RCS files in the subdirectory named `RCS` that have locks:

```
rlog -L -R RCS/*,v
```

Print the headers of those files:

```
rlog -L -h RCS/*,v
```

Print the headers plus the log messages of the locked revisions:

```
rlog -L -l RCS/*,v
```

Print complete log information:

```
rlog RCS/*,v
```

DIAGNOSTICS

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 if unsuccessful.

AUTHOR

`rlog` was developed by Walter F. Tichy.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsdiff(1), rcsmerge(1), rcsfile(4), rcsintro(5).

NAME

rlogin - remote login

SYNOPSIS

`rlogin` *rhost* [`-e` *c*] [`-7`] [`-8`] [`-l` *username*]
rhost [`-e` *c*] [`-7`] [`-8`] [`-l` *username*]

DESCRIPTION

`rlogin` connects your terminal on the local host to the remote host, *rhost*; `rlogin` acts as a virtual terminal to the remote system. The hostname *rhost* can be either the official name or an alias as listed in `/etc/hosts`; see *hosts*(4).

In a manner similar to *remsh*(1), `rlogin` allows a user to log in on an equivalent remote host, *rhost*, bypassing the normal login/password sequence. For more information about equivalent hosts and how to specify them in the files `/etc/hosts.equiv` and `.rhosts`, see *hosts.equiv*(4). Note that the searching of the files `/etc/hosts.equiv` and `.rhosts` occurs on the remote host, and that the `.rhosts` file must be owned by the remote user account or by the super-user.

If the originating user account is not equivalent to the remote user account, the originating user is prompted for the password of the remote account. If this fails, a login and password are prompted for, as when `login` is used (see *login*(1)).

The terminal type specified by the current `TERM` environment variable is propagated across the network and used to set the initial value of your `TERM` environment variable on the remote host. Your terminal baud rate is also propagated to the remote host, and is required by some systems to set up the pseudo-terminal used by `rlogind`.

All echoing takes place at the remote site, so that (except for delays) the remote login is transparent. A line beginning with `~.` disconnects from the remote host. A line beginning with `~!` causes a shell escape on the local host, where `~` is the escape character (see the `-e` option below).

If `rlogin` is run from a shell that supports job control, see (*csh*(1), *ksh*(1)), and *sh-posix*(1)), escape sequences can be used to suspend `rlogin`. The following assumes that `^Z` and `^Y` are the user's `susp` and `dsusp` characters, respectively (see *stty*(1) and *termio*(7)). A line beginning with the escape sequence `~^Z` or `~^Y` suspends the `rlogin` session and returns the user to the shell that invoked `rlogin`. The `rlogin` job can be resumed with the `fg` command (see *csh*(1), *ksh*(1), and *sh-posix*(1). `~^Z` suspends both `rlogin` processes: the one transmitting user input to the remote login, and the one displaying output from the remote login. `~^Y` suspends only the input process; output from the remote login continues to be displayed.

The system administrator can arrange for more convenient access to a remote host *rhost* by linking `remsh` to `/usr/hosts/`*rhost*, allowing use of the remote host name (*rhost*) as a command; see *remsh*(1). For example, if `remotehost` is the name of a remote host and `/usr/hosts/remotehost` is linked to *remsh*, and if `/usr/hosts` is in your search path, the command

```
remotehost
```

is equivalent to

```
rlogin remotehost
```

If at any time `rlogin` is unable to read from or write to the socket connection to the remote host, the message `Connectionclosed.` is printed on *stderr* and `rlogin` exits.

Options

`rlogin` recognizes the following options:

`-e`*c*	Sets the escape character to *c*. There is no space separating this option and the argument character. To start a line with the escape character, two of the escape characters must be entered. The default escape character is `~`. Some characters may conflict with your terminal configuration, such as `^S`, `^Q`, or backspace. Using one of these as the escape character may not be possible or may cause problems communicating with the remote host (see *stty*(1) and *tty*(7)).
`-l` *username*	Sets the user login name on the remote host to *username*. The default name is the current account name of the user invoking `rlogin`.

`-7`	Causes `rlogin` to set the character size to seven bits. The eighth bit of each byte sent is set to zero.
`-8`	Causes `rlogin` to use an eight-bit data path. This is the default HP-UX behavior. To use eight-bit characters, the terminal must be configured to generate either eight-bit characters with no parity, or seven bit characters with null parity. The HP-UX implementation of `rlogind` interprets seven bit characters with even, odd, or mark parity as eight-bit non-USASCII characters. You may also need to reconfigure the remote host appropriately (see *stty*(1) and *tty*(7)). Some remote hosts may not provide the necessary support for eight-bit characters. In this case, or if it is not possible to disable parity generation by the local terminal, use the `-7` option.

RETURN VALUES

`rlogin` sends an error message to *stderr* and returns a non-zero value if an error occurs before the connection to the remote host is completed; otherwise it returns a zero.

DIAGNOSTICS

Diagnostics can occur from both the local and remote hosts. Those that occur on the local host before the connection is completely established are written to *stderr*. Once the connection is established, any error messages from the remote host are written on *stdout* like any other data.

`login/tcp: Unknown service`
: `rlogin` was unable to find the login service listed in the `/etc/services` database file.

`There is no entry for you (user ID <uid>) in /etc/passwd`
: `rlogin` was unable to find your user ID in the password file.

  Next Step: Contact your system administrator.

`< system call >: ...`
: An error occurred when `rlogin` attempted the indicated system call. See the appropriate manual entry for a description of the error.

EXAMPLES

Login as the same user on the remote host `remote`:

```
rlogin remote
```

Set the escape character to a `!`, use a seven-bit data connection, and attempt a login as user `guest` on host `rhost`:

```
rlogin rhost -e! -7 -l guest
```

Assuming that your system administrator has set up the links in `/usr/hosts`, the following is equivalent to the previous command:

```
rhost -e! -7 -l guest
```

WARNINGS

For security purposes, the `/etc/hosts.equiv` and `.rhosts` files should exist, even if they are empty, and should be readable and writable only by the owner. Note also that all information, including any passwords asked for, is passed unencrypted between the two hosts.

`rlogin` is unable to transmit the **Break** key as an interrupt signal to the remote system, regardless of whether the user has done `stty brkint` on the local system. The key assigned to `SIGINT` with `stty intr` *c* should be used instead (see *stty (1)).*

AUTHOR

`rlogin` was developed by the University of California, Berkeley.

FILES

`/etc/hosts.equiv`	list of equivalent hosts
`$HOME/.rhosts`	user's private equivalence list
`/usr/hosts/*`	for `rhost` version of the command

SEE ALSO

csh(1), ksh(1), remsh(1), stty(1), telnet(1), rlogind(1M), hosts(4), hosts.equiv(4), services(4), termio(7).

NAME

rm - remove files or directories

SYNOPSIS

`rm [-f|-i] [-Rr]` *file* ...

DESCRIPTION

`rm` removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write and search (execute) permission in its directory, but no permissions on the file itself; but if the sticky bit is set on the directory containing the file, only the owner of the file, the owner of the directory, or a user having appropriate privileges can remove the file.

If a user does not have write permission for a file to be removed and the standard input is a terminal, a prompt containing the file name and its permissions is printed requesting that the removal of the file be confirmed (see Access Control Lists below). A line is then read from the standard input. If that line begins with `y` the file is deleted; otherwise, the file remains. No questions are asked when the `-f` option is given or if the standard input is not a terminal.

If *file* is of type directory, and `-f` option is not specified, and either the permissions of *file* do not permit writing and the standard input is a terminal or the `-i` option is specified, `rm` writes a prompt to standard error and reads a line form the standard input. If the response is not affirmative, it does nothing more with the current file and goes on to any remaining files.

Options

`rm` recognizes the following options:

`-f` Force each file or directory to be removed without prompting for confirmation, regardless of the permissions of the entry. This option also suppresses diagnostic messages regarding non-existent operands.

This option does not suppress any diagnostic messages other than those regarding non-existent operands. To suppress all error message and interactive prompts, the `-f` option should be used while redirecting the standard error output to */dev/null*.

`-i` Write a prompt to the standard error output requesting confirmation before removing each entry. This option is ignored when used in conjunction with the `-f` option.

`-R` For each argument that is a directory, this option causes *rm* to recursively delete the entire contents of that directory before removing the directory itself. When used in conjunction with the `-i` option, *rm* asks whether to examine each directory before interactively removing files in that directory and again afterward to confirm removing the directory itself.

`-r` Equivalent to `-R`.

Access Control Lists (ACLs).

If a file has optional ACL entries, `rm` displays a plus sign (+) after the file's permissions. The permissions shown summarize the file's `st_mode` value returned by `stat()` (see *stat*(2)). See also *acl*(5).

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of filenames as single and/or multi-byte characters for the `rm` command.

`LANG` determines the language in which messages are displayed and determines the local language equivalent of `y` (for yes/no) queries.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `rm` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

Generally self-explanatory. Note that the `-f` option does not suppress all diagnostic messages.

It is forbidden to remove the file `..` in order to avoid the consequences of inadvertently using a command such as:

```
rm -r .*
```

If a designated file is a directory, an error comment is printed unless the `-R` or `-r` option is used.

EXAMPLES

Remove files with a prompt for verification:

`rm -i` *filenames*

Remove all the files in a directory:

`rm -i` *directoryname*`/*`

Note that this command removes files only, and does not remove any directories in *directoryname*.

Remove file in current directory whose name starts with `-` or some other character that is special to the shell:

`rm ./-`*filename*
`rm ./\*`*filename*
etc.

Remove file in current directory whose name starts with some strange (usually non-printing, invisible) character or perhaps has spaces at the beginning or end of the filename:

`rm ./*`*filename*`*`

`*`*filename*`*` must be unique in the directory or other files will also be removed.

A powerful and dangerous command to remove a directory is:

`rm -fR` *directoryname*

or

`rm -Rf` *directoryname*

which removes all files and directories from *directoryname* without any prompting for verification to remove the files or the directories. This command should only be used when absolutely certain that all the files and directories in *directoryname* as well as *directoryname* itself are to be removed.

DEPENDENCIES

NFS

`rm` does not display a plus sign (+) to indicate the existence of optional access control list entries when asking for confirmation before removing a networked file.

SEE ALSO

unlink(2), acl(5).

STANDARDS CONFORMANCE

`rm`: SVID2, XPG2, XPG3, POSIX.2

`rmdir`: SVID2, XPG2, XPG3, POSIX.2

NAME
rmdel - remove a delta from an SCCS file

SYNOPSIS
`rmdel -r` *SID file* ...

DESCRIPTION
`rmdel` removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get*(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*).

If a directory is named, **`rmdel`** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **`s.`**) and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

EXTERNAL INFLUENCES
Environment Variables
LANG determines the local language equivalent of the affirmative string ("yes"). LANG also determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, **`rmdel`** behaves as if all internationalization variables are set to "C". See *environ*(5).

DIAGNOSTICS
Use *help*(1) for explanations.

FILES
`x.file`	See *delta*(1).
`z.file`	See *delta*(1).

SEE ALSO
delta(1), get(1), help(1), prs(1), sccsfile(4).
SCCS User's Guide, tutorial in *Programming on HP-UX*.

STANDARDS CONFORMANCE
rmdel: SVID2, XPG2, XPG3

NAME

rmdir - remove directories

SYNOPSIS

`rmdir` [`-f` | `-i`] [`-p`] *dir* ...

DESCRIPTION

`rmdir` removes the directory entry for each *dir* operand that refers to an empty directory.

Directories are removed in the order specified. Consequently, if a directory and a subdirectory of that directory are both specified as arguments, the subdirectory must be specified before the parent directory so that the parent directory will be empty when `rmdir` tries to remove it. Removal of a directory requires write and search (execute) permission in its parent directory, but no permissions on the directory itself; but if the sticky bit is set on the parent directory, only the owner of the directory, the owner of the parent directory, or a user having appropriate privileges can remove the directory.

Options

`rmdir` recognizes the following options:

`-f` Force each directory to be removed without prompting for confirmation, regardless of the presence of the `-i` option. This option also suppresses diagnostic messages regarding non-existent operands.

This option does not suppress any diagnostic messages other than those regarding non-existent operands. To suppress all error message and interactive prompts, the `-f` option should be used while redirecting the standard error output to `/dev/null`.

`-i` Write a prompt to the standard error output requesting confirmation before removing each directory. This option is ignored when used in conjunction with the `-f` option.

`-p` Path removal. If, after removing a directory with more than one pathname component, the parent directory of that directory is now empty, `rmdir` removes the empty parent directory. This continues until `rmdir` encounters a non-empty parent directory, or until all components of the original pathname have been removed.

When used in conjunction with the `-i` option, `rmdir` asks whether to remove each directory component of a path.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of filenames as single and/or multi-byte characters for the *rmdir* command.

`LANG` determines the language in which messages are displayed and determines the local language equivalent of `y` (for yes/no) queries.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `rmdir` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

Generally self-explanatory. Note that the `-f` option does not suppress all diagnostic messages.

EXAMPLES

To remove directories with a prompt for verification:

`rmdir -i` *directories*

To remove as much as possible of a path, type:

`rmdir -p` *component1*`/`*component2*`/`*dir*

SEE ALSO

rm(1), rmdir(2), stat(2).

STANDARDS CONFORMANCE

rmdir: SVID2, XPG2, XPG3

NAME

rmnl - remove extra new-line characters from file

SYNOPSIS

`rmnl`

DESCRIPTION

`rmnl` removes all blank lines from a file (except at beginning of file as explained below), and is useful for removing excess white space from files for display on a CRT terminal. Groups of two or more successive \n (new-line) characters are reduced to a single \n character, effectively eliminating all blank lines in the file *except* that one or more blank lines at the beginning of a file remain as a single blank line.

To remove redundant blank lines rather than all blank lines, use *ssp*(1).

To remove all blank lines from a file including beginning of file, use `rmnl` piped to `ssp`, or `ssp` piped to `rmnl`.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

SEE ALSO

man(1), ssp(1).

NAME

rpcgen - an RPC protocol compiler

SYNOPSIS

```
rpcgen [-u] infile
rpcgen -h [-o outfile ] [ infile ]
rpcgen -c [-o outfile ] [ infile ]
rpcgen -s transport [-u] [-o outfile ] [ infile ]
rpcgen -l [-o outfile ] [ infile ]
rpcgen -m [-o outfile ] [ infile ]
```

DESCRIPTION

`rpcgen` is a tool that generates `C` code to implement an RPC protocol. The input to `rpcgen` is a language similar to `C` known as RPC (Remote Procedure Call) Language. Information about RPC Language syntax is available in the rpcgen chapter of the manual *Programming and Protocols for NFS Services*.

`rpcgen` is normally used in the first synopsis shown above where it takes an input file and generates four output files. If the *infile* is named `proto.x`, `rpcgen` generates a header file in `proto.h`, XDR routines in `proto_xdr.c`, server-side stubs in `proto_svc.c`, and client-side stubs in `proto_clnt.c`.

The other synopsis forms shown above are used when only a particular output file is needed. Their usage is described in the Options section below.

The input file is processed by `cpp` before being interpreted by `rpcgen` (see *cpp*(1)), meaning that all standard `cpp` directives are available for use in an `rpcgen` input file. Preprocessing by `cpp` is done for each output file created. For each type of output file, `rpcgen` defines a special `cpp` symbol for use by the `rpcgen` programmer:

`RPC_HDR`	defined when compiling into header files
`RPC_XDR`	defined when compiling into XDR routines
`RPC_SVC`	defined when compiling into server-side stubs
`RPC_CLNT`	defined when compiling into client-side stubs

In addition, `rpcgen` does preprocessing of its own. Any line beginning with `%` is passed directly into the output file, uninterpreted by `rpcgen`. By using the `cpp` symbols mentioned above, you can pass text into a specific output file.

To provide customized XDR routines for customized data types, you can declare data objects to be of a data type that is undefined to `rpcgen`. `rpcgen` passes them through to the `.h` file it creates. You are responsible for providing the definition of such data types in user-supplied files. For every data type that is undefined, `rpcgen` assumes that there exists a routine with the name `xdr_` prefixed to the name of the undefined type.

Options

`rpcgen` recognizes the following options and command-line arguments:

`-c`	Compile into XDR routines.
`-h`	Compile into `C` data-definitions (a header file)
`-l`	Compile into client-side stubs.
`-s` *transport*	Compile into server-side stubs, using the specified transport. The supported transports are `udp` and `tcp`. This option can be invoked more than once so as to compile a server that serves multiple transports. If `rpcgen` is called without options, the server-side code that is generated can serve both `udp` and `tcp` transports.
`-m`	Compile into server-side stubs, but do not produce a `main()` routine. This option is useful if you want to supply your own `main()` .
`-o` *outfile*	Specify the name of the output file. If none is specified, standard output is used (`-c`, `-h`, `-l`, `-m`, and `-s` modes only).
`-u`	When the server-side stub is produced, additional code to handle signals is generated. On reception of a signal, this signal handler code unmaps the server program from the port mapper before the server terminates. This code is added only if a `main()` routine is produced in the server-side stub. The `-u` option must not be specified with

the `-c`, `-h`, `-l`, or `-m` options.

The following signals are trapped: `SIGHUP`, `SIGINT`, `SIGQUIT`, and `SIGTERM`.

WARNINGS

Nesting of structures is not supported. As a work-around, structures can be declared at the top-level, and their names used inside other structures in order to achieve the same effect.

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

AUTHOR

rpcgen was developed by Sun Microsystems, Inc. and HP.

SEE ALSO

rpcgen chapter of *Programming and Protocols for NFS Services*

INTERNATIONAL SUPPORT

8- and 16-bit data only in strings, comments and filenames.

NAME

rtprio - execute process with real-time priority

SYNOPSIS

`rtprio` *priority command* [*arguments*]
`rtprio` *priority* *-pid*
`rtprio -t` *command* [*arguments*]
`rtprio -t` *-pid*

DESCRIPTION

`rtprio` executes *command* with a real-time priority, or changes the real-time priority of currently executing process *pid*. Real-time priorities range from zero (highest) to 127 (lowest). Real-time processes are not subject to priority degradation, and are all of greater (scheduling) importance than non-real-time processes. See *rtprio*(2) for more details.

If `-t` is specified instead of a real-time *priority*, `rtprio` executes *command* with a timeshare (non-real-time) priority, or changes the currently executing process *pid* from a possibly real-time priority to a timeshare priority. The former is useful to spawn a timeshare priority command from a real-time priority shell.

If `-t` is not specified, *command* is not scheduled, or *pid*'s real-time priority is not changed, if the user is not a member of a group having `PRIV_RTPRIO` access and is not the user with appropriate privileges. When changing the real-time priority of a currently executing process, the effective user ID of the calling process must be the user with appropriate privileges, or the real or effective user ID must match the real or saved user ID of the process to be modified.

RETURN VALUE

`rtprio` returns exit status 0 if *command* is successfully scheduled or if *pid*'s real-time priority is successfully changed, 1 if *command* is not executable or *pid* does not exist, and 2 if *command* (*pid*) lacks real-time capability, or the invoker's effective user ID is not a user who has appropriate privileges, or the real or effective user or the real or effective user ID does not match the real or saved user ID of the process being changed.

EXAMPLES

Execute file `a.out` at a real-time priority of 100:

```
rtprio 100 a.out
```

Set the currently running process pid 24217 to a real-time priority of 40:

```
rtprio 40 -24217
```

AUTHOR

`rtprio` was developed by HP.

SEE ALSO

setprivgrp(1M), getprivgrp(2), rtprio(2), privilege(5).

NAME

rup - show host status of local machines (RPC version)

SYNOPSIS

`rup [-h] [-l] [-t] [`*host* ...`]`

DESCRIPTION

`rup` gives a status similar to `uptime` for remote machines. It broadcasts on the local network and displays the responses it receives. Though the listing is normally in the order responses are received, the order can be changed by using command-line options. The broadcast process takes about two minutes.

When *host* arguments are given, instead of broadcasting, `rup` only queries the list of specified hosts.

A remote host only responds if it is running the `rstatd` daemon (see *rstatd*(1M)).

Options

`rup` recognizes the following command-line options:

`-h`	Sort the display alphabetically by host name.
`-l`	Sort the display by load average.
`-t`	Sort the display by up time.

WARNINGS

Broadcasting does not work through gateways; therefore, `rup` does not report information about machines that are reachable only through gateways.

DIAGNOSTICS

The exit code of `rup` is the number of errors (eg. bad host names) encountered; zero for success.

AUTHOR

`rup` was developed by Sun Microsystems, Inc.

FILES

`/etc/inetd.conf`

SEE ALSO

ruptime(1), inetd(1M), rstatd(1M), services(4).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages

NAME

ruptime - show status of local machines

SYNOPSIS

`ruptime [-a][-r][-l][-t][-u]`

DESCRIPTION

`ruptime` outputs a status line for each machine on the local network that is running the `rwho` daemon. `ruptime`'s status lines are formed from packets broadcast once every 3 minutes between `rwho` daemons (see *rwhod*(1M)) on each host on the network. Each status line has a field for the name of the machine, the status of the machine (up or down), how long the machine has been up or down, the number of users logged into the machine, and the 1-, 5- and 15-minute load averages for the machine when the packet was sent.

The status of the machine is reported as "up" unless no report has been received from the machine for 11 minutes or more.

The length of time that the machine has been up is shown as:

days +*hours* :*minutes*

Load averages are the average number of jobs in the run queue over the last 1-, 5- and 15-minute intervals when the packet was sent.

An example status line output by `ruptime` might be:

```
machine1  up  1+5:15,  7 users, load  1.47, 1.16, 0.80
```

The above status line would be interpreted as follows:

`machine1` is presently "up" and has been up for 1 day, 5 hours and 15 minutes. It currently has 7 users logged in. Over the last 1-minute interval, an average of 1.47 jobs were in the run queue. Over the last 5-minute interval, an average of 1.16 jobs were in the run queue. Over the last 15-minute interval, an average of 0.80 jobs were in the run queue.

If a user has not used the system for an hour or more, the user is considered idle. Idle users are not shown unless the `-a` option is specified.

Options

If not options are specified, the listing is sorted by host name. Options change sorting order as follows:

`-l`	Sort by load average.
`-t`	Sort by up time.
`-u`	Sort by the number of users.
`-r`	Reverse the sort order.

DIAGNOSTICS

`no hosts!?!`

No status report files in `/usr/spool/rwho`. *Next Step*: Ask the system administrator to check whether the `rwho` daemon is running.

AUTHOR

`ruptime` was developed by the University of California, Berkeley.

FILES

`/usr/spool/rwho/whod.*` data files

SEE ALSO

rwho(1), rwhod(1M).

NAME

rusers - determine who is logged in on machines on local network

SYNOPSIS

`rusers` [`-a`][`-h`][`-i`][`-l`][`-u`][*host* ...]

DESCRIPTION

`rusers` produces output similar to the "quick" option of *who*(1), but for remote machines. It broadcasts on the local network and prints the responses it receives. Though the listing is normally in the order that responses are received, the order can be changed by specifying a command-line option. The broadcast process takes about two minutes.

When *host* arguments are given, instead of broadcasting, `rusers` only queries the list of specified hosts.

For each machine, the default is to print a line listing the host name and all users on that host. When the `-l` option is given, `rusers` uses an output format similar to *rwho*(1). If a user has not typed on the system for a minute or more, the idle time is reported.

A remote host only responds if it is running the *rusersd*(1M) daemon.

Options

`rusers` recognizes the following command-line options:

`-a`	Give a report for a machine even if no users are logged in on it.
`-h`	Sort alphabetically by host name.
`-i`	Sort by idle time.
`-l`	Give a longer listing in the style of `who -R` (see *who*(1)).
`-u`	Sort by number of users.

RETURN VALUE

`rusers` returns exit code zero if no errors are encountered; otherwise it returns the number of errors found.

AUTHOR

`rusers` was developed by Sun Microsystems, Inc.

WARNINGS

Broadcasting does not work through gateways; therefore, `rusers` does not report information about machines that are reached only through gateways.

FILES

`/etc/inetd.conf`

SEE ALSO

rwho(1), who(1), inetd(1M), rusersd(1M).

INTERNATIONAL SUPPORT

8-bit data, 16-bit data, messages.

NAME

rwho - show who is logged in on local machines

SYNOPSIS

`rwho [-a]`

DESCRIPTION

rwho produces output similar to the output of the HP-UX **who** command for all machines on the local network that are running the **rwho** daemon (see *who*(1) and *rwhod*(1M)). If **rwhod** has not received a report from a machine for 11 minutes, **rwho** assumes the machine is down and **rwho** does not report users last known to be logged into that machine.

rwho's output line has fields for the name of the user, the name of the machine, the user's terminal line, the time the user logged in, and the amount of time the user has been idle. Idle time is shown as:

hours **:** *minutes*

If a user has not typed to the system for a minute or more, **rwho** reports this as idle time. If a user has not typed to the system for an hour or more, the user is omitted from **rwho**'s output unless the **-a** flag is given.

An example output line from **rwho** would look similar to:

```
joe_user   machine1:tty0p1  Sep 12  13:28   :11
```

This output line could be interpreted as **joe_user** is logged into **machine1** and his terminal line is **tty0p1**. **joe_user** has been logged on since September 12 at 13:28 (1:28 p.m.). **joe_user** has not typed anything into **machine1** for 11 minutes.

WARNINGS

rwho's output becomes unwieldy when the number of users for each machine on the local network running **rwhod** becomes large. One line of output occurs for each user on each machine on the local network that is running **rwhod**.

AUTHOR

rwho was developed by the University of California, Berkeley.

FILES

/usr/spool/rwho/whod.*
information about other machines

SEE ALSO

ruptime(1), rusers(1), rwhod(1M).

NAME

sact - print current SCCS file editing activity

SYNOPSIS

`sact` *file* ...

DESCRIPTION

`sact` informs the user of any impending deltas to a named SCCS file. This situation occurs when `get -e` has been previously executed without a subsequent execution of `delta` (see *delta*(1) and *get*(1)). If a directory is named on the command line, `sact` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1	SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	SID for the new delta to be created.
Field 3	Logname of the user making the delta (i.e., executed a `get` for editing).
Field 4	Date when `get -e` was executed.
Field 5	Time when `get -e` was executed.

DIAGNOSTICS

Use `help` for explanations (see *help*(1)).

SEE ALSO

delta(1), get(1), unget(1).

SCCS User's Guide, in *Programming on HP-UX*.

STANDARDS CONFORMANCE

sact: SVID2, XPG2, XPG3

NAME

sar - system activity reporter

SYNOPSIS

sar [**-ubdycwaqvmAM**] [**-o** *file*] *t* [*n*]

sar [**-ubdycwaqvmAM**] [**-s** *time*] [**-e** *time*] [**-i** *sec*] [**-f** *file*]

DESCRIPTION

In the first form above, **sar** samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the **-o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second form, with no sampling interval specified, extracts data from a previously recorded *file*, either the one specified by **-f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **-s** and **-e** *time* arguments of the form *hh* [*:mm* [*:ss*]]. The **-i** option selects records at *sec*-second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

-u Report CPU utilization (the default); portion of time running in one of several modes. On a multi-processor system, if the **-M** option is used together with the **-u** option, per-CPU utilization as well as the average CPU utilization of all the processors are reported. If the **-M** option is not used, only the average CPU utilization of all the processors is reported:

cpu	cpu number (only on a multi-processor system with the **-M** option);
%usr	user mode;
%sys	system mode;
%wio	idle with some process waiting for I/O (only block I/O, raw I/O, or VM pageins/swapins indicated);
%idle	otherwise idle.

-b
Report buffer activity:

bread/s	Number of physical reads per second from the disk (or other block devices) to the buffer cache;
bwrit/s	Number of physical writes per second from the buffer cache to the disk (or other block device);
lread/s	Number of reads per second from buffer cache;
lwrit/s	Number of writes per second to buffer cache;
%rcache	Buffer cache hit ratio for read requests e.g., 1 – bread/lread;
%wcache	Buffer cache hit ratio for write requests e.g., 1 – bwrit/lwrit;
pread/s	Number of reads per second from character device using the **physio()** (raw I/O) mechanism;
pwrit/s	Number of writes per second to character device using the **physio()** (i.e., raw I/O) mechanism; mechanism.

-d
Report activity for each block device, e.g., disk or tape drive:

device Logical name of the device and its corresponding LU. Devices are categorized into the following three device types:

disk1 – HP-IB disks (CS/80)
disk2 – CIO HP-FL disks (CS/80)
disk3 – SCSI disks and NIO FL disks;

%busy
Portion of time device was busy servicing a request;

`avque`
Average number of requests outstanding for the device;

`r+w/s`
Number of data transfers per second (read and writes) from and to the device;

`blks/s`
Number of bytes transferred (in 512-byte units) from and to the device;

`avwait`
Average time (in milliseconds) that transfer requests waited idly on queue for the device;

`avserv`
Average time (in milliseconds) to service each transfer request (includes seek, rotational latency, and data transfer times) for the device.

`-y`
Report tty device activity:

`rawch/s`	Raw input characters per second;
`canch/s`	Input characters per second processed by `canon()`;
`outch/s`	Output characters per second;
`rcvin/s`	Receive incoming character interrupts per second;
`xmtin/s`	Transmit outgoing character interrupts per second;
`mdmin/s`	Modem interrupt rate (not supported; always 0).

`-c`
Report system calls:

`scall/s`	Number of system calls of all types per second;
`sread/s`	Number of `read()` and/or `readv()` system calls per second;
`swrit/s`	Number of `write()` and/or `writev()` system calls per second;
`fork/s`	Number of `fork()` and/or `vfork()` system calls per second;
`exec/s`	Number of `exec()` system calls per second;
`rchar/s`	Number of characters transferred by read system calls block devices only) per second;
`wchar/s`	Number of characters transferred by write system calls (block devices only) per second.

`-w`
Report system swapping and switching activity:

`swpin/s`	Number of process swapins per second;
`swpot/s`	Number of process swapouts per second;
`bswin/s`	Number of 512-byte units transferred for swapins per second;
`bswot/s`	Number of 512-byte units transferred for swapouts per second;
`pswch/s`	Number of process context switches per second.

`-a`
Report use of file access system routines:

`iget/s`	Number of file system `iget()` calls per second;
`namei/s`	Number of file system `lookuppn()` (pathname translation) calls per second;
`dirblk/s`	Number of file system blocks read per second doing directory lookup.

`-q`
Report average queue length while occupied, and percent of time occupied. On a multi-processor machine, if the `-M` option is used together with the `-q` option, the per-CPU run queue as well as the average run

queue of all the processors are reported. If the **-M** option is not used, only the average run queue information of all the processors is reported:

`cpu`	cpu number (only on a multi-processor system and used with the **-M** option)
`runq-sz`	Average length of the run queue(s) of processes (in memory and runnable);
`%runocc`	The percentage of time the run queue(s) were occupied by processes (in memory and runnable);
`swpq-sz`	Average length of the swap queue of runnable processes (processes swapped out but ready to run);
`%swpocc`	The percentage of time the swap queue of runnable processes (processes swapped out but ready to run) was occupied.

-v
Report status of text, process, inode and file tables:

`text-sz`	(Not Applicable);
`proc-sz`	The current-size and maximum-size of the process table;
`inod-sz`	The current-size and maximum-size of the inode table (inode cache);
`file-sz`	The current-size and maximum-size of the system file table;
`text-ov`	(Not Applicable);
`proc-ov`	The number of times the process table overflowed (number of times the kernel could not find any available process table entries) between sample points;
`inod-ov`	The number of times the inode table (inode cache) overflowed (number of times the kernel could not find any available inode table entries) between sample points;
`file-ov`	The number of times the system file table overflowed (number of times the kernel could not find any available file table entries) between sample points.

-m
Report message and semaphore activities:

`msg/s`	Number of System V `msgrcv()` calls per second;
`sema/s`	Number of System V `semop()` calls per second.

-A
Report all data. Equivalent to **-udqbwcayvm**.

-M
Report the per-processor data on a multi-processor system when used with **-q** and/or **-u** options. If the **-M** option is not used on a multi-processor system, the output format of the **-u** and **-q** options is the same as the uni-processor output format and the data reported is the average value of all the processors.

EXAMPLES

Watch CPU activity evolve for 5 seconds:

```
sar 1 5
```

Watch CPU activity evolve for 10 minutes and save data:

```
sar -o temp 60 10
```

Review disk and tape activity from that period later:

```
sar -d -f temp
```

Review cpu utilization on a multi-processor system later:

```
sar -u -M  -f temp
```

FILES

`/usr/adm/sa/sa`*dd*	daily data file, where *dd* is two digits representing the day of the month.

SEE ALSO

sa1(1M).

STANDARDS CONFORMANCE

`sar`: SVID2

NAME

sccsdiff - compare two versions of an SCCS file

SYNOPSIS

`sccsdiff -r`*SID1* `-r`*SID2* [`-p`] [`-s`*n*] *file* ...

DESCRIPTION

`sccsdiff` compares two versions of an SCCS file, and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

`-r`*SID?*	*SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to `bdiff` in the order given (see *bdiff*(1)). The SIDs accepted, and the corresponding version retrieved for the comparison are the same as for `get` (see *get*(1)).
`-p`	pipe output for each file through `pr` (see *pr*(1)).
`-s`*n*	*n* is the file segment size that `bdiff` passes to `diff` (see `diff (1))`. This is useful when `diff` fails due to a high system load.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

DIAGNOSTICS

file`: No differences`
The two versions are identical.

Use `help` for explanations.

FILES

`/tmp/get?????`	Temporary files

SEE ALSO

bdiff(1), diff(1), get(1), help(1), pr(1).

SCCS User's Guide, in *Programming on HP-UX*.

NAME

script - make typescript of terminal session

SYNOPSIS

`script` [`-a`] [*file*]

DESCRIPTION

`script` makes a typescript of everything printed on your terminal. It starts a shell named by the **SHELL** environment variable, or by default **/bin/sh**, and silently records a copy of output to your terminal from that shell or its descendents, using a pseudo-terminal device (see *pty*(7)).

All output is written to *file*, or appended to *file* if the **-a** option is given. If no file name is given, the output is saved in a file named **typescript**. The recording can be sent to a line printer later with *lp*(1), or reviewed safely with the **-v** option of *cat*(1).

The recording ends when the forked shell exits (or the user ends the session by typing "exit") or the shell and all its descendents close the pseudo-terminal device.

This program is useful when operating a CRT display and a hard-copy record of the dialog is desired. It can also be used for a simple form of session auditing.

`script` respects the convention for login shells as described in *su*(1), *sh*(1), and *ksh*(1). Thus, if it is invoked with a command name beginning with a hyphen (-) (that is, **-script**), **script** passes a basename to the shell that is also preceded by a hyphen.

EXAMPLES

Save everything printed on the user's screen into file `scott`:

```
script scott
```

Append a copy of everything printed to the user's screen to file `temp`:

```
script -a temp
```

WARNINGS

A command such as `cat scott`, which displays the contents of the destination file, should not be issued while executing `script` because it would cause `script` to log the output of the `cat` command to itself until all available disk space is filled. Other commands, such as *more*(1), can cause the same problem but to a lesser degree.

`script` records all received output in the *file*, including typing errors, backspaces, and cursor motions. Note that it does not record typed characters; only echoed characters. Thus passwords are not recorded in the *file*. Responses other than simple echoes (such as output from screen-oriented editors and **ksh** command editing) are recorded as they appeared in the original session.

AUTHOR

`script` was developed by the University of California, Berkeley and HP.

NAME

sdfchmod - change mode of an SDF file

SYNOPSIS

`sdfchmod` *mode device* `:` *file...*

DESCRIPTION

`sdfchmod` is intended to mimic `chmod` (see *chmod*(1)).

An SDF file name is recognized by the embedded colon (`:`) delimiter (see *sdf*(4) for SDF file naming conventions).

The permissions of each named file are changed according to *mode*, which can be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see *chmod*(2)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others.

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

who is a combination of the letters `u` (for user's permissions), `g` (group), and `o` (other). The letter `a` (all) is a synomym for `ugo`, which is the default if *who* is omitted.

op can be `+` to add *permission* to the file's mode, `-` to take away *permission*, or `=` to assign *permission* absolutely (all other bits are reset).

permission is any combination of the letters `r` (read), `w` (write), `x` (execute), `s` (set owner or group ID) and `t` (save text – sticky); `u`, `g`, or `o` indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with `=` to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter `s` is only useful with `u` or `g`; `t` works only with `u`.

EXAMPLES

The following examples assume that an SDF directory structure exists on the HP-UX device file `/dev/rdsk/c1d0s3`.

Deny write permission to others for the SDF directory `/bin`:

```
sdfchmod o-w /dev/rdsk/c1d0s3:/bin
```

Make the SDF file `/users/fred/a.out` executable and readable by everyone:

```
sdfchmod a=rx /dev/rdsk/c1d0s3:/users/fred/a.out
```

Add read permission for the group associated with the SDF file `/last.boot.rev`:

```
sdfchmod g+r /dev/rdsk/c1d0s3:/last.boot.rev
```

Assign read and execute permission to everybody, and set the set-user-id bit for the SDF file `/usr/local/hoo`:

```
sdfchmod 4555 /dev/rdsk/c1d0s3:/usr/local/hoo
```

These two commands perform the same function: give read, write, and execute permission to the owner; and read and execute permissions to everybody else for the SDF file `/users/debbie/script`:

```
sdfchmod a=rx,u+w /dev/rdsk/c1d0s3:/users/debbie/script
sdfchmod 755 /dev/rdsk/c1d0s3:/users/debbie/script
```

AUTHOR

`sdfchmod` was developed by HP.

SEE ALSO

sdf(4), chmod(1), chmod(2).

NAME

sdfchown, sdfchgrp - change owner or group of an SDF file

SYNOPSIS

`sdfchown` *owner device*`:`*file* ...

`sdfchgrp` *group device*`:`*file* ...

DESCRIPTION

`sdfchown` and `sdfchgrp` are intended to mimic *chown*(1) and *chgrp*(1).

An SDF file name is recognized by the embedded colon (`:`) delimiter (see *sdf*(4) for SDF file naming conventions).

`sdfchown` changes the owner of the *files* to *owner*. The owner can be either a decimal user ID or a login name found in the password file.

`sdfchgrp` changes the group ID of the *files* to *group*. The group can be either a decimal group ID or a group name found in the group file.

EXAMPLES

The following examples assume that an SDF directory structure exists on the HP-UX device file `/dev/rdsk/c9d1d5`.

Set the owner of the SDF file `/users/abc/phone.num` to `adm`:

```
sdfchown adm /dev/rdsk/c9d1d5:/users/abc/phone.num
```

Set the group ID of the SDF file `/tmp/b.date` to the decimal number `105`:

```
sdfchgrp 105 /dev/rdsk/c9d1d5:/tmp/b.date
```

AUTHOR

`sdfchown` was developed by HP.

FILES

`/etc/passwd`
`/etc/group`

SEE ALSO

chown(1), chgrp(1), group(4), passwd(4), sdf(4).

NAME

sdfcp, sdfln, sdfmv - copy, link, or move files to/from an SDF volume

SYNOPSIS

`sdfcp` *file1* [*file2* ...] *target*
`sdfln` *file1* [*file2* ...] *target*
`sdfmv` *file1* [*file2* ...] *target*

DESCRIPTION

sdfcp, **sdfln**, and **sdfmv** are intended to mimic **cp**, **mv**, and **ln**, respectively (see *cp*(1), *mv*(1), and *ln*(1)).

An SDF file name is recognized by the embedded colon (**:**) delimiter (see *sdf*(4) for SDF file naming conventions).

sdfcp copies an HP-UX file to an SDF file, or an SDF file to either an SDF or HP-UX file. It also copies a list of HP-UX files to an SDF directory, or copies a list of SDF files to either an SDF or HP-UX directory.

sdfln creates links to *target* if, and only if, all *files* referenced on the command line are on the same SDF volume.

sdfmv behaves the same way as **sdfcp**, except that it moves files instead of copying them.

The last name on the argument list is the target file or directory. If two or more files are specified in the command line, not counting *target*, then *target* must be a directory. Under no circumstances can any argument other than *target* be a directory.

The file name **-** (dash) is interpreted to mean standard input or standard output, depending on the position in the argument list. The use of the file name **-** is meaningless when using **sdfln** or **sdfmv**.

EXAMPLES

The following examples assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c2d0s2**.

Copy the HP-UX file **mydata** to the SDF file **/users/old/mike/olddata**:

```
sdfcp mydata /dev/rdsk/c2d0s2:/users/old/mike/olddata
```

Copy SDF file **/users/gary/.cshrc** to SDF directory **/tmp** (on the same SDF volume):

```
sdfcp /dev/rdsk/c2d0s2:/users/gary/.cshrc /dev/rdsk/c2d0s2:/tmp
```

Copy SDF files **/a/b** and **/a/c** to HP-UX directory **/users/dave**:

```
sdfcp /dev/rdsk/c2d0s2:/a/b /dev/rdsk/c2d0s2:/a/c /users/dave
```

Copy standard input to SDF file **/users/craig/memo**:

```
sdfcp - /dev/rdsk/c2d0s2:/users/craig/memo
```

Copy SDF file **/etc/rc** to SDF file **/etc/rc.old** on another SDF volume residing in the HP-UX device file **/dev/rdsk/c2d1s0**:

```
sdfcp /dev/rdsk/c2d0s2:/etc/rc /dev/rdsk/c2d1s0:/etc/rc.old
```

Implement a *cat*(1) program for concatenating SDF files using **sdfcp** in a shell script:

```
if [ $# -lt 1 ]
then
    echo "Usage: sdfcat file ..."
    exit 1
fi
for i in $*
do
    sdfcp $i -
done
```

Link SDF file **/tmp/x** to **/users/gary/x1**:

```
sdfln /dev/rdsk/c2d0s2:/tmp/x /dev/rdsk/c2d0s2:/users/gary/x1
```

Move HP-UX file `/etc/rc.backup` to SDF file `/etc/rc`:

```
sdfmv /etc/rc.backup /dev/rdsk/c2d0s2:/etc/rc
```

Assuming that the current HP-UX directory contains only regular files, move all files in an HP-UX directory to SDF directory `/savestuff`:

```
sdfmv * /dev/rdsk/c2d0s2:/savestuff
```

AUTHOR

`sdfcp` was developed by HP.

SEE ALSO

sdf(4), cp(1).

NAME

sdffind - find files in an SDF system

SYNOPSIS

`sdffind` *path-name-list expression*

DESCRIPTION

`sdffind` is intended to mimic *find*(1).

An SDF file name is recognized by the embedded colon (`:`) delimiter (see *sdf*(4) for SDF file naming conventions).

`sdffind` recursively descends the directory hierarchy for each path name in *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below.

`-name` *pattern* True if *pattern* matches the current file name.

`-perm` *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

(flags&onum)==onum

`-type` *c*
True if the type of the file is *c*, where *c* is `b`, `c`, `d`, `p`, or `f` for block special file, character special file, directory, fifo (also called a named pipe), or plain file.

`-type` *n*
True if the current file being examined by `sdffind` is a network special file.

`-links` *n*
True if the file has *n* links.

`-user` *uname*
True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the `/etc/passwd` file (on the local system, not the SDF file system), it is taken as a user ID.

`-group` *gname*
True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the `/etc/group` file (on the local system, not the SDF file system), it is taken as a group ID.

`-size` *n*
True if the file is *n* blocks long.

`-exec` *cmd*
True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument `{ }` is replaced by the current path name.

`-ok` *cmd*
Like `-exec` except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing `y`.

`-print`
Always true; causes the current path name to be printed. This option must be included on the `sdffind` command line anytime you want `sdffind` to print the path names it has found on the standard output. If `-print` is not specified, `sdffind` locates the files, but fails to tell you about them!

When `-print` is specified as the only *expression*, `sdffind` prints the absolute path names of all files it finds, beginning at each directory in the *path-name-list*. If `-print` is included as the last component of an *expression*, `sdffind` prints the absolute path names of only those files that satisfy the other primaries in *expression*.

`-inum` *n*
True if the file has inode number *n*.

EXAMPLES

The following examples assume that an SDF directory structure exists on HP-UX device file `/dev/rdsk/c3d0s0`.

Print the names of all files on SDF volume `/dev/rdsk/c3d0s0`:

```
sdffind /dev/rdsk/c3d0s0: -print
```

Print the name of all the subdirectories under `/usr/lib` on the SDF file system:

```
sdffind /dev/rdsk/c3d0s0:/usr/lib -type d -print
```

Give a long listing of every ordinary file under `/users` on the SDF file system:

```
sdffind /dev/rdsk/c3d0s0:/users -type f -exec sdfls -l {} ';'
```

Find all files named `core` on the SDF volume and ask whether they should be removed:

```
sdffind /dev/rdsk/c3d0s0: -name core -ok sdfrm {} ';'
```

AUTHOR

`sdffind` was developed by HP.

FILES

`/etc/passwd`
`/etc/group`

SEE ALSO

chmod(1), find(1), stat(2), sdf(4).

NAME

sdfls, sdfll - list contents of SDF directories

SYNOPSIS

`sdfls [-AadlpFi] [`*names*`]`
`sdfll [-AadlpFi] [`*names*`]`

DESCRIPTION

`sdfls` is intended to mimic *ls*(1). `sdfll` is equivalent to `sdfls -l`.

An SDF file name is recognized by the embedded colon (`:`) delimiter (see *sdf*(4) for SDF file naming conventions).

For each SDF directory named, `sdfls` lists the contents of that SDF directory; for each SDF file named, `sdfls` repeats its name and the information requested.

For users with appropriate privileges, `sdfls` defaults to listing all files except `.` (current directory) and `..` (parent directory).

Options

`sdfls` recognizes the following options:

- `-a` List all entries. In the absence of this option, entries whose names begin with a period (`.`) are *not* listed.
- `-A` Same as `-a`, except that the current directory `.` and parent directory `..` are not listed. For users with appropriate privileges, this flag defaults to ON, and is turned off by `-A`. Due to the internal data representation of the SDF directory format, the `-A` and `-a` options perform the same function.
- `-d` If argument is a directory, list only its name; often used with `-l` to get the status of a directory.
- `-l` List in long format giving mode, number of links, owner, group, size in bytes, and time of last modification for each file.
- `-p` Do not use `/etc/passwd` and `/etc/group` to interpret user and group ownership, but rather print out the numeric form.
- `-F` If the entry is a directory or SRM special file, print a `/` character after the entry, or if the entry is executable, print a `*` character after the entry.
- `-i` Print the inode number of each entry before listing the entry names.

EXAMPLES

The following examples assume that an SDF directory structure exists on the HP-UX device file `/dev/rdsk/c7s0s1`.

List all files in the root directory of the SDF directory structure:

```
sdfls -a /dev/rdsk/c7s0s1:
```

List (in long format) all the information about the SDF directory `/users/root` itself (but not the files in the directory):

```
sdfls -ld /dev/rdsk/c7s0s1:/users/root
```

Print (in long form) all the information about every file in the SDF directory `etc`, printing numbers instead of names for user and group IDs. This is useful if the SDF directory structure was not created on the local system, but was brought in from an external Series 500 system.

```
sdfls -ap /dev/rdsk/c7s0s1:/etc
```

DEPENDENCIES

Series 500 systems support network special files. `sdfls` with the `-F` option prints a `/` character after any network special file entries.

AUTHOR

`sdfls` was developed by HP.

FILES

`/etc/passwd`
: user id file

`/etc/group`
: group id file

SEE ALSO

ls(1), sdf(4).

NAME

sdfmkdir - make an SDF directory

SYNOPSIS

sdfmkdir *device* **:** *dirname...*

DESCRIPTION

sdfmkdir is intended to mimic *mkdir*(1).

An SDF file name is recognized by the embedded colon (**:**) delimiter (see *sdf*(4) for SDF file naming conventions).

sdfmkdir creates specified directories in mode 777, masked with the current value of **umask**.

RETURN VALUE

sdfmkdir returns zero if all directories were successfully created; otherwise, it prints a diagnostic and returns non-zero.

EXAMPLES

The following example assumes that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c0d1s5**.

Create an empty subdirectory named **sysmods** under directory **/usr/lib**:

```
sdfmkdir /dev/rdsk/c0d1s5:/usr/lib/sysmods
```

AUTHOR

sdfmkdir was developed by HP.

SEE ALSO

sdf(4), mkdir(1).

NAME

sdfrm, sdfrmdir - remove SDF files or directories

SYNOPSIS

sdfrm [**-fri**] *device*:*file* ...

sdfrmdir *device*:*file* ...

DESCRIPTION

sdfrm and **sdfrmdir** are intended to mimic *rm*(1) and *rmdir*(1).

An SDF file name is recognized by the embedded colon (**:**) delimiter (see *sdf*(4) for SDF file naming conventions).

sdfrm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed.

If a designated file is a directory, an error comment is printed (unless the optional argument **-r** has been used; see below).

Options

Recognized options are:

- **-f** Remove a file with no questions asked, even if the file has no write permission.
- **-r** Cause **sdfrm** to recursively delete the entire contents of a directory, followed by the directory itself. **sdfrm** can recursively delete up to 17 levels of directories.
- **-i** Cause **sdfrm** to ask whether or not to delete each file. If **-r** is also specified, **sdfrm** asks whether to examine each directory encountered.

sdfrmdir removes entries for the named directories, which must be empty.

EXAMPLES

The following examples assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c6d0s1**.

Recursively comb through SDF directory **/tmp** and ask if each SDF file should be removed (forced, with no file mode checks):

```
sdfrm -irf /dev/rdsk/c6d0s1:/tmp
```

Remove SDF directory **/users/doug**:

```
sdfrmdir /dev/rdsk/c6d0s1:/users/doug
```

AUTHOR

sdfrm was developed by HP.

SEE ALSO

rm(1), rmdir(1), sdf(4).

NAME

sdiff - side-by-side difference program

SYNOPSIS

`sdiff` [*options* ...] *file1 file2*

DESCRIPTION

`sdiff` uses the output of *diff*(1) to produce a side-by-side listing of two files, indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
abc  |  xyz
abc     abc
bca  <
cba  <
dcb     dcb
     >  cde
```

Options

`sdiff` recognizes the following options:

`-w` *n*	Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
`-l`	Only print on the left side when lines are identical.
`-s`	Do not print identical lines.
`-o` *output*	Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, `sdiff` prompts the user with a `%` and waits for one of the following user-typed commands:

`l`	append the left column to the output file
`r`	append the right column to the output file
`s`	turn on silent mode; do not print identical lines
`v`	turn off silent mode
`e l`	call the editor with the left column
`e r`	call the editor with the right column
`e b`	call the editor with the concatenation of left and right
`e`	call the editor with a zero length file
`q`	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed - stream text editor

SYNOPSIS

`sed` [`-n`] *script* [*file* ...]

`sed` [`-n`] [`-e` *script*] ... [`-f` *script_file*] ... [*file* ...]

DESCRIPTION

`sed` copies the named text *files* (standard input default) to the standard output, edited according to a script containing up to 100 commands. Only complete input lines are processed. Any input text at the end of a file that is not terminated by a new-line character is ignored.

Options

`sed` recognizes the following options:

- `-f` *script_file* — Take script from file *script_file*.
- `-e` *script* — Edit according to *script*. If there is just one `-e` option and no `-f` options, the flag `-e` can be omitted.
- `-n` — Suppress the default output.

`sed` interprets all `-e`*script* and `-f`*script_file* arguments in the order given. Use caution, if mixing `-e` and `-f` options, to avoid unpredictable or incorrect results.

Command Scripts

A script consists of editor commands, one per line, of the following form:

[*address* [`,` *address*]] *function* [*arguments*]

In normal operation, `sed` cyclically copies a line of input into a *pattern space* (unless there is something left after a `D` command), applies in sequence all commands whose *addresses* select that pattern space, and, at the end of the script, copies the pattern space to the standard output (except under `-n`) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

Command Addresses

An *address* is either a decimal number that counts input lines cumulatively across files, a `$` which addresses the last line of input, or a context address; that is, a `/`*regular expression*`/` in the style of *ed*(1) modified thus:

- In a context address, the construction `\`*?regular expression?*, where *?* is any character, is identical to `/`*regular expression*`/`. Note that in the context address **`\xabc\xdefx`**, the second **`x`** stands for itself, so that the regular expression is **`abcxdef`**.
- The escape sequence `\n` matches a new-line character embedded in the pattern space.
- A period (`.`) matches any character except the terminal new-line of the pattern space.
- A command line with no addresses selects every pattern space.
- A command line with one address selects each pattern space that matches the address.
- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second (if the second address is a number less than or equal to the line number first selected, only one line is selected). Thereafter the process is repeated, looking again for the first address.

`sed` supports Basic Regular Expression syntax (see *regexp*(5)).

Editing commands can also be applied to only non-selected pattern spaces by use of the negation function `!` (described below).

Command Functions

In the following list of functions, the maximum number of permissible addresses for each function is indicated in parentheses. Other funtion elements are interpreted as follows:

text One or more lines, all but the last of which end with `\` to hide the new-line. Backslashes in *text* are treated like backslashes in the replacement string of an `s` command, and can be used to protect initial blanks and tabs against the stripping that is done on every script line.

rfile Must terminate the command line, and must be preceded by exactly one blank.

wfile Must terminate the command line, and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

`sed` recognizes the following functions:

(1) `a\`
text Append. Place *text* on the output before reading next input line.

(2) `b` *label* Branch to the `:` command bearing *label*. If no *label* is specified, branch to the end of the script.

(2) `c\`
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) `d` Delete pattern space and start the next cycle.

(2) `D` Delete initial segment of pattern space through first new-line and start the next cycle.

(2) `g` Replace contents of the pattern space with contents of the hold space.

(2) `G` Append contents of hold space to the pattern space.

(2) `h` Replace contents of the hold space with contents of the pattern space.

(2) `H` Append the contents of the pattern space to the hold space.

(1) `i\`
text Insert. Place *text* on the standard output.

(2) `l` List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in three-digit octal number format (with a preceding backslash), and long lines are folded.

(2) `n` Copy the pattern space to the standard output if the default output has not been suppressed (by the `-n` option on the command line or the `#n` command in the *script* file). Replace the pattern space with the next line of input.

(2) `N` Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) `p` Print. Copy the pattern space to the standard output.

(2) `P` Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) `q` Quit. Branch to the end of the script. Do not start a new cycle.

(1) `r` *rfile* Read contents of *rfile* and place on output before reading the next input line.

(2) `s` */regular expression/replacement/flags*
Substitute *replacement* string for instances of *regular expression* in the pattern space. Any character can be used instead of `/`. For a fuller description see *ed*(1). *flags* is zero or more of:

n *n* = 1-2048 (`LINE_MAX`). Substitute for just the *n*th occurrence of *regular expression* in the pattern space.

`g` Global. Substitute for all non-overlapping instances of *regular expression* rather than just the first one.

`p` Print the pattern space if a replacement was made and the default output has been suppressed (by the `-n` option on the command line or the `#n` command in the *script* file).

`w` *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) `t` *label* Test. Branch to the `:` command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a `t`. If *label* is empty, branch to the end of the script.

(2) `w` *wfile* Write. Append the pattern space to *wfile*.

(2) `x` Exchange the contents of the pattern and hold spaces.

(2) `y/`*string1*`/`*string2*`/`
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2) `!` *function*
Don't. Apply the *function* (or group, if *function* is `{}`) only to lines *not* selected by the address or addresses.

(0) `:` *label* This command does nothing; it bears a *label* for `b` and `t` commands to branch to.

(1) `=` Place the current line number on the standard output as a line.

(2) `{` Execute the following commands through a matching `}` only when the pattern space is selected. The syntax is:

```
{ cmd1
cmd2
cmd3
  .
  .
  .
}
```

(0) An empty command is ignored.

(0) `#` If a `#` appears as the first character on the first line of a script file, that entire line is treated as a comment with one exception: If the character after the `#` is an `n`, the default output is suppressed. The rest of the line after `#n` is also ignored. A script file must contain at least one non-comment line.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating regular expressions.

`LC_CTYPE` determines the interpretation of text as single- and/or multi-byte characters, and the characters matched by character class expressions in regular expressions.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `sed` behaves as if all internationalization variables are set to "C" (see *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Make a simple substitution in a file from the command line or from a shell script, changing `abc` to `xyz`:

```
sed 's/abc/xyz/' file1 >file1.out
```

Same as above but use shell or environment variables `var1` and `var2` in search and replacement strings:

```
sed "s/$var1/$var2/" file1 >file1.out
```

or

```
    sed 's/'$var1'/'$var2'/' file1 >file1.out
```

Multiple substitutions in a single command:

```
    sed -e 's/abc/xyz/' -e 's/lmn/rst/' file1
```

or

```
    sed -e 's/abc/xyz/' \
    -e 's/lmn/rst/' \
    file1 >file1.out
```

SEE ALSO

awk(1), ed(1), grep(1), environ(5), lang(5), regexp(5).

`sed`: *A Non-Interactive Streaming Editor* tutorial in the *Text Processing Users Guide*.

WARNINGS

`sed` limits command scripts to a total of not more than 100 commands.

The hold space is limited to 8192 characters.

`sed` processes only text files. See the glossary for a definition of text files and their limitations.

STANDARDS CONFORMANCE

`sed`: SVID2, XPG2, XPG3, POSIX.2

NAME

sh, rsh - shell, the standard/restricted command programming language

SYNOPSIS

sh [**- -acefhiknrstuvx** ...] [*arg* ...]
rsh [**- -acefhiknrstuvx** ...] [*arg* ...]

DESCRIPTION

`sh` is a command programming language that executes commands read from a terminal or a file. `rsh` is a restricted version of the standard command interpreter `sh`; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See "Invocation" and "Special Commands" sections later in this entry for details about command line options and arguments, particularly the `set` command.

Definitions

A **blank** is a tab or a space character. A **name** is a sequence of letters, digits, or underscores beginning with a letter or underscore. A **parameter** is a name, a digit, or any of the characters *****, **@**, **#**, **?**, **-**, **$**, and **!**.

Commands

A **simple-command** is a sequence of non-blank **words** separated by **blanks**. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The **value** of a simple-command is its exit status if it terminates normally, or (octal) 200+**status** if it terminates abnormally (see *signal*(5) for a list of status values).

A **pipeline** is a sequence of one or more **commands** separated by `|` (or, for historical compatibility, by `^`). The standard output of each command except the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process and the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A **list** is a sequence of one or more pipelines separated by `;`, `&`, `&&`, or `||`, and optionally terminated by `;` or `&`. Of these four symbols, `;` and `&` have equal precedence, which is lower than that of `&&` and `||`. The symbols `&&` and `||` also have equal precedence. A semicolon (`;`) causes sequential execution of the preceding pipeline; an ampersand (`&`) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol `&&` (`||`) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-line characters instead of semicolons can appear in a **list** to delimit commands.

A **command** is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

Each time a `for` command is executed, *name* is set to the next *word* taken from the `in` *word* list. If `in` *word* ... is omitted, the `for` command executes the `do` *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

`case` *word* `in` [*pattern* [| *pattern*] ...) *list* `;;`] ... `esac`
: A `case` command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the Pattern Matching Notation as qualified for the `case` command (see *regexp*(5)).

`if` *list* `then` *list* [`elif` *list* `then` *list*] ... [`else` *list*] `fi`
: The *list* following `if` is executed and, if it returns a zero exit status, the *list* following the first `then` is executed. Otherwise, the *list* following `elif` is executed and, if its value is zero, the *list* following the next `then` is executed. Failing that, the `else` *list* is executed. If no `else` *list* or `then` *list* is executed, the `if` command returns a zero exit status.

`while` *list* `do` *list* `done`
: A `while` command repeatedly executes the `while` *list* and, if the exit status of the last command in the list is zero, executes the `do` *list*; otherwise the loop terminates. If no commands in the `do` *list* are executed, the `while` command returns a zero exit status; `until` can be used in place of `while` to negate the loop termination test.

(*list*) Execute *list* in a sub-shell.

{*list* ;} The purpose of using braces is to allow the aggregate output from *list* to be redirected elsewhere. If redirection is used as in:

{*list* ; } >*file*

a subshell is created to execute *list*. This implies that any shell variables set, created, or modified within the { } do not retain their values outside the { }. If no redirection is used, a subshell is not created, and shell modifications made within the { } are preserved.

name ( ) {*list* ; }
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see "Execution").

The following words are recognized only as the first word of a command, and when not quoted:

```
if then else elif fi case esac for while until do done { }
```

Comments

A word beginning with # causes that word and all the following characters up to a new-line character to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (` `) can be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution

The $ character is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters can be assigned values by `set`. Keyword parameters (also known as variables) can be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. Having a function and a variable with the same *name* is not allowed.

${*parameter*}
The value, if any, of the parameter is substituted. Braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all positional parameters, starting with $1, are substituted (separated by spaces). Parameter $0 is set from argument zero when the shell is invoked.

${*parameter*:-*word*}
If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null, set it to *word*; the value of the parameter is then substituted. Positional parameters cannot be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}
If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string. Thus, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the `set` command.

`?`	The decimal value returned by the last synchronously executed command.
`$`	The process id of the last separately-invoked shell (i.e., a shell arising from direct invocation or `#!`). This parameter is not updated for subshells arising from `( )`, command substitution, etc.
`!`	The process number of the last background command invoked.
`LINES` `COLUMNS`	The number of lines and columns in the current display area. These parameters are set only under specific circumstances. See *Signals*.

The following parameters are used by the shell:

`HOME`	The default argument (home directory) for the `cd` command.
`PATH`	The search path for commands (see "Execution" below). Users executing under **rsh** cannot change **PATH**.
`CDPATH`	The search path for the `cd` command.
`MAIL`	If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.
`MAILCHECK`	This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell checks before each prompt.
`MAILPATH`	A colon (**:**)-separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by `%` and a message to be printed when the modification time changes. The default message is `you have mail`.
`PS1`	Primary prompt string, by default "**$** ".
`PS2`	Secondary prompt string, by default "**>** ".
`IFS`	Internal field separators, normally **space**, **tab**, and **new-line**.
`SHACCT`	If this parameter is set to the name of a file writable by the user, the shell writes an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1M) and *acctcms*(1M) can be used to analyze the data collected.
`SHELL`	When the shell is invoked, it scans the environment (see "Environment" below) for this name. If it is found and there is an **r** in the file name part of its value, the shell becomes a restricted shell. **SHELL** is also used by some processors to determine which command interpreter to run.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (`""` or `´´`) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is processed as a pattern for file name expansion. The form of the patterns is the Pattern Matching Notation defined by *regexp*(5).

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

;, **&**, **(**, **)**, **|**, **^**, **<**, **>**, *new-line*, *space*, *tab*, **#** (comment)

A character can be **quoted** (i.e., made to stand for itself) by preceding it with a \. The pair **\***new-line* is ignored. All characters enclosed between a pair of single quote marks (`´´`), except a single quote, are quoted. Inside double quote marks (`""`), parameter and command substitution occurs and \ quotes the characters `\`, `` ` ``, `"`, and `$`. `"$*"` is equivalent to `"$1 $2 ..."`, whereas `"$@"` is equivalent to `"$1""$2"` ....

Prompting

When used interactively, the shell prompts with the value of `PS1` before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of `PS2`) is issued.

Input/Output

Before a command is executed, its input and output can be redirected using a special notation interpreted by the shell. The following can appear anywhere in a simple-command or can precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

`<`*word*	Use file *word* as standard input (file descriptor 0).
`>`*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
`>>`*word*	Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
`<<[-]`*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) `\`*new-line* is ignored, and `\` must be used to quote the characters `\`, `$`, `` ` ``, and the first character of *word*. If `-` is appended to `<<`, all leading tabs are stripped from *word* and from the document.
`<&`*digit*	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using `>&`*digit* (see *dup*(2)). Note that *digit* must be in the range 0 through 9.
`<&-`	The standard input is closed. Similarly for the standard output using `>&-`.

If any of the above is preceded by a digit in the range 0 through 9, the file descriptor that becomes associated with the file is that specified by the digit (rather than the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1. Note that this type of I/O redirection is necessary when *synchronously* collecting standard output and standard error output in the same file. Redirecting standard output and standard error separately causes asynchronous collection of data at the destination (information written to standard output can be subsequently over-written by information written to standard error and vice-versa).

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *xxx*). If the order of redirections is reversed, file descriptor 2 is associated with the terminal (assuming file descriptor 1 was originally) and file descriptor 1 is associated with file *xxx*.

If a command is followed by `&`, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for executing a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The **environment** (see *environ*(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the `export` command is used to bind the shell's parameter to the environment (see also `set -a`). To remove a parameter from the environment, use the `unset` command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* can be augmented by prefixing it with one or more assignments to parameters. Thus:

```
     TERM=450 cmd
```

and

```
     (export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **-k** option is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
     echo a=b c
     set -k
     echo a=b c
```

Signals

The **INTERRUPT** and **QUIT** signals for an invoked command are ignored if the command is followed by **&** and the command does not override the **SIGINT** and **SIGQUIT** signal settings it inherited from **/bin/sh**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

If a SIGWINCH signal is received, **sh** determines the new size of the display area and resets the values of the **LINES** and **COLUMNS** variables appropriately. Note that the value of either variable is modified only if that variable exists at the time SIGWINCH is received. **sh** does not create these variables. Any traps set on SIGWINCH are executed immediately after **LINES** and **COLUMNS** are reset.

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a **/** the search path is not used (such commands are not executed by the restricted shell). Otherwise, each directory in the path is searched for an executable file. If the file has execute permissions but is not a directory or an executable object code file, it is assumed to be a script file which is a file of data for an interpreter. If the first two characters of the script file are **#!**, **exec** (see *exec*(2)) expects an interpreter path name to follow. **exec** then attempts to execute the specified interpreter as a separate process to read the entire script file. If a call to **exec** fails, **/bin/sh** is spawned to interpret the script file.

A parenthesized command is also executed in a sub-shell. The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary **exec**s later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

The following commands are executed in the shell process. Input/output redirection is permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* — Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*. Note that this command does not spawn another shell to execute *file*, and thus differs in behavior and output from executing *file* as a shell script. It is not necessary that the execute permission bit be set for *file*.

`break` [*n*] — Exit from the enclosing `for` or `while` loop, if any. If *n* is specified, break *n* levels.

`continue` [*n*] — Resume the next iteration of the enclosing `for` or `while` loop. If *n* is specified, resume at the *n*-th enclosing loop.

`cd` [*arg*] — Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is null (meaning the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a **/** the search path is not used. Otherwise, each directory in the path is searched for *arg*. The `cd` command cannot be executed by `rsh`.

`echo` [*arg* ...] — Echo arguments. See *echo*(1) for usage and description.

`eval` [*arg* ...] — The arguments are read as input to the shell and the resulting command(s) executed.

`exec` [*arg* ...] — The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments can appear and, if no other arguments are given, cause the shell input/output to be modified.

`exit` [*n*] — Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file also causes the shell to exit.)

`export` [*name* ...] — The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of names currently included in the environment are printed. Function names *cannot* be exported.

`hash` [`-r`] [*name* ...]
: For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The `-r` option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *hits* is the number of times a command has been invoked by the shell process. *cost* is a measure of the work required to locate a command in the search path. Certain situations require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *cost* is incremented when the recalculation is done.

`newgrp` [*arg* ...] — Equivalent to **`exec newgrp`** *arg* .... See *newgrp*(1) for usage and description.

`pwd` — Print the current working directory. A **-H** option to *pwd* is supported. It makes hidden directories (see *cdf*(4)) in the path visible. See *pwd*(1) for usage and description.

`read` *name* ... — One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

Note that although the read command is a built-in command and is generally executed directly by the shell, this is not the case when it is used in a pipeline. In a pipeline, a new shell is forked to execute the read command with the result that any shell variables set are not available to the parent shell when the pipeline is finished. This has the effect of making the read command useless in a pipeline.

`readonly` [*name* ...]
: The given *names* are marked *readonly* and the values of the these *names* cannot be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

`return` [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

`set` [`--aefhkntuvx` [*arg* ...]]
`set` sets or unsets options, and resets the values of the positional parameters to the args given, if any. The option list is terminated by the first argument that does not begin with `-` or `+`, or upon encountering an argument consisting entirely of `--`. Recognized options are:

- `-a` Mark variables which are modified or created for export.
- `-e` Exit immediately if a command exits with a non-zero exit status.
- `-f` Disable file name generation
- `-h` Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- `-k` All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- `-n` Read commands but do not execute them.
- `-t` Exit after reading and executing one command.
- `-u` Treat unset variables as an error when substituting.
- `-v` Print shell input lines as they are read.
- `-x` Print commands and their arguments as they are executed.
- `--` Do not change any of the options; useful when `$1` is to be set to a string beginning with `-` or `+`.

Using `+` rather than `-` causes these options to be unset. These options can also be used upon invocation of the shell. The current set of options can be found in `$-`. The remaining arguments are positional parameters and are assigned, in order, to `$1`, `$2`, ... If no arguments are given, the values of all names are printed.

`shift` [*n*]
The positional parameters from `$n+1` ... are renamed `$1` .... If *n* is not given, it is assumed to be 1.

`test`
Evaluate conditional expressions. See *test*(1) for usage and description. Note that `[ ... ]` in an `if` *list* is interpreted the same as `test` .... There must be blanks around the brackets.

`times`
Print the accumulated user and system times for processes run from the shell.

`trap` [*arg*] [*n*] ...
The command *arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) or signal 18 (death of child) produces an error. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

`type` [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

`ulimit` [`-f` [*n*]]
If the `-f` *n* option is used, a size limit of *n* blocks is imposed on files written by child processes (files of any size can be read). If *n* is not specified, the current limit is printed. If no option is specified, `-f` is assumed.

`umask` [*nnn*]
The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the

mask is printed.

`unset` [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK**, and **IFS** cannot be unset.

`wait` [*n*]
Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

Invocation

Options can be specified in a single argument or in multiple arguments, but in all cases each option argument must begin with `-`. (All options except `c`, `s`, `i`, and `r` can also be prefaced with a `+`, which turns off the associated option or options, but this is redundant when invoking a new shell because all options are turned off by default).

If the first character of argument zero is `-`, commands are initially read from `/etc/profile`, then from `$HOME/.profile`, if the files exist. Thereafter, commands are read as described below.

The options below are interpreted by the shell at invocation (thus they cannot be used with the `set` command). Unless the `-c` or `-s` option is specified, the first non-option argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file.

`-c` *string*	If the `-c` option is present then commands are read from *string*.
`-s`	If the `-s` option is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for "Special Commands") is written to file descriptor 2.
`-i`	If the `-i` option is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that `kill 0` does not kill an interactive shell) and INTERRUPT is caught and ignored (so that `wait` is interruptible). In all cases, QUIT is ignored by the shell.
`-r`	If the `-r` option is present the shell is a restricted shell.

The remaining options and arguments are described under the `set` command above.

`rsh` Only

`rsh` is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of `rsh` are identical to those of `sh`, except that the following are disallowed:

- Changing directory (see *cd*(1)),
- Setting the value of `$PATH`,
- Specifying path or command names containing `/`,
- Redirecting output (> and >>).

The restrictions above are enforced after `.profile` is interpreted.

When a command to be executed is found to be a shell procedure, `rsh` invokes `sh` to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the `.profile` has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (such as `/usr/rbin`) that can be safely invoked by `rsh`. Commands such as `vi`, `sh`, `ksh`, `csh`, and such that can break `rsh` restrictions should not be included in that directory. Some systems also provide a restricted editor `red`.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as letters, and the characters matched by character class expressions in pattern matching notation.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, `sh` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

The error codes returned by the shell are:

0	Success.
1	Built-in command failure (see *Special* Commands).
2	Syntax error.
3	Signal received that is not trapped.

If the shell is non-interactive, it terminates and passes one of the above as its exit status. If it is interactive, it does not terminate, but **`$?`** is set to one of the above values.

Whenever a child process of the shell dies due to a signal, the shell returns an exit status of 80 hexadecimal plus the number of the signal.

WARNINGS

If a command is executed and a command having the same name is installed in a directory in the search path before the directory where the original command was found, the shell continues to **`exec`** the original command. Use the **`hash`** command to correct this situation.

When the shell encounters **>>**, it does not open the file in append mode. Instead, it opens the file for writing and seeks to the end.

If you move the current directory or one above it, **`pwd`** cannot give the correct response. Use the **`cd`** command with a full path name to correct this situation.

The command **`readonly`** (without arguments) produces the same output as the command **`export`**.

Failure (non-zero exit status) of a special command preceding a `||` symbol prevents the *list* following `||` from executing.

In an international environment, character ordering is determined by the setting of LC_COLLATE, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particulary when using range expressions in file-name-generation patterns. For example, the command,

```
rm [a-z]*
```

might be expected to match all file names beginning with a lowercase alphabetic character. However, if dictionary ordering is specified by LC_COLLATE, it also matches file names beginning with an uppercase character (as well as those beginning with accented letters). Conversely, it fails to match letters collated after **`z`** in languages such as Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form:

```
rm [[:lower:]]*
```

This uses LC_CTYPE to determine character classes and works predictably for all supported languages and codesets. For shell scripts produced on non-internationalized systems (or without consideration for the above dangers), it is recommended that they be executed in a non-NLS environment. This requires that LANG, LC_COLLATE, etc., be set to "C" or not set at all.

`sh` implements command substitution by creating a pipe between itself and the command. If the root file system is full, the substituted command cannot write to the pipe. As a result, the shell receives no input from the command, and the result of the substitution is null. In particular, using command substitution for variable assignment under such circumstances results in the variable being silently assigned a NULL value.

The signal SIGSEGV should not be blocked when executing `sh`.

`sh` reserves file descriptor 59 for internal use. Reducing the number of available file descriptors below 60 causes `sh` to fail.

Each time a function is executed in a shell script, any arguments given to the function overwrite the values of the positional parameters for the entire script. If the values of the positional parameters must be preserved, they must be explicitly saved before each function call.

AUTHOR

`sh` was developed by AT&T and HP.

FILES

```
$HOME/.profile
/dev/null
/etc/profile
/tmp/sh*
```

SEE ALSO

cd(1), echo(1), env(1), login(1), newgrp(1), pwd(1), test(1), umask(1), acctcms(1M), acctcom(1M), dup(2), exec(2), fork(2), pipe(2), ulimit(2), umask(2), wait(2), a.out(4), cdf(4), profile(4), environ(5), lang(5), regexp(5), signal(5).

Bourne Shell tutorial in *Shells Users Guide*.

STANDARDS CONFORMANCE

`sh`: SVID2, XPG2, XPG3

`rsh`: SVID2

`.`: SVID2, XPG2, XPG3

`:`: SVID2, XPG2, XPG3

`break`: SVID2, XPG2, XPG3

`continue`: SVID2, XPG2, XPG3

`eval`: SVID2, XPG2, XPG3

`exec`: SVID2, XPG2, XPG3

`exit`: SVID2, XPG2, XPG3

`export`: SVID2, XPG2, XPG3

`read`: SVID2, XPG2, XPG3

`set`: SVID2, XPG2, XPG3

`shift`: SVID2, XPG2, XPG3

`time`: SVID2, XPG2, XPG3

`trap`: SVID2, XPG2, XPG3

`unset`: SVID2, XPG2, XPG3

NAME

sh - shell partially based on preliminary POSIX draft

SYNOPSIS

sh [±**aefhikmnoprstuvx**] [±**o** *option*] ... [**-c** *string*] [*arg* ...]

NOTE

This shell is intended to track the shell specification of the evolving POSIX *Shell and Utility Application Interface* and related FIPS standards. Check any standards conformance documents shipped with your system for information on the conformance of this shell to any standards.

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. See *Invocation* below for the meaning of arguments to the shell.

Definitions

A **metacharacter** is one of the following characters:

; **&** **(** **)** **|** **<** **>** new-line space tab

A **blank** is a tab or a space. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *functions* and *named parameters*. A **word** is a sequence of **characters** separated by one or more non-quoted **metacharacters**. A **command** is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A special command is a command that is carried out by the shell without creating a separate process. Except for documented side effects, most special commands can be implemented as separate utilities.

In addition, the **#** character is used as a comment delimiter and must be treated appropriately when used in arguments. See **Quoting** below.

Commands

A *simple-command* is a sequence of blank-separated words that may be preceded by a parameter assignment list. (See *Environment* below). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(5) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|** and optionally preceded with a **!**. The standard output of each command but the last is connected by a pipe (see *pipe*(2)) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. If **!** does not precede the pipeline, the exit status of the pipeline is the exit status of the last command in the pipeline. Otherwise, the exit status of the pipeline is the logical negation of the exit status of the last command in the pipeline.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;**, **&**, or **|&**. Of these five symbols, **;**, **&**, and **|&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (that is, the shell does not wait for that pipeline to finish). The symbol **|&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent shell using the **-p** option of the special commands **read** and **print** described later. The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines can appear in a *list,* instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *identifier* [**in** *word* ...] **do** *list* **done**

Each time **for** is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, **for** executes the **do** *list* once for each positional parameter set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

`select` *identifier* [`in` *word* ...] `do` *list* `done`
: A `select` command prints on standard error (file descriptor 2), the set of *word*s, each preceded by a number. If `in` *word* ... is omitted, the positional parameters are used instead (see *Parameter Substitution* below). The `PS3` prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed *word*s, the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty, the selection list is printed again. Otherwise the value of the parameter *identifier* is set to NULL . The contents of the line read from standard input is saved in the parameter `REPLY`. The *list* is executed for each selection until a **break** or *end-of-file* is encountered.

`case` *word* `in` [[`(`]*pattern* [| *pattern*] ...) *list* `;;`] ... `esac`
: A `case` command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is identical to that used for file name generation (see *File Name Generation* below).

`if` *list* `then` *list* [`elif` *list* `then` *list*] ... [`else` *list*] `fi`
: The *list* following `if` is executed and, if it returns a zero exit status, the *list* following the first `then` is executed. Otherwise, the *list* following `elif` is executed and, if its value is zero, the *list* following the next `then` is executed. Failing that, the `else` *list* is executed. If no `else` *list* or `then` *list* is executed, `if` returns a zero exit status.

`while` *list* `do` *list* `done`
`until` *list* `do` *list* `done`
: A `while` command repeatedly executes the `while` *list*, and if the exit status of the last command in the list is zero, executes the `do` *list*; otherwise the loop terminates. If no commands in the `do` *list* are executed, `while` returns a zero exit status; `until` may be used in place of `while` to negate the loop termination test.

(*list*)
: Execute *list* in a separate environment. If two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{ *list* ; }
: Execute *list*, but not in a separate environment. Note that `{` is a keyword and requires a trailing blank to be recognized.

[[*expression*]]
: Evaluates *expression* and returns a zero exit status when *expression* is true. See *Conditional Expressions* below, for a description of *expression*. Note that `[[` and `]]` are keywords and require blanks between them and *expression*.

`function` *identifier* { *list* ; }
`identifier` *() { list ;}*
: Define a function referred to by *identifier*. The body of the function is the *list* of commands between `{` and `}` (see *Functions* below).

`time` *pipeline*
: The *pipeline* is executed and the elapsed time, the user time, and the system time are printed on standard error.

The following keywords are recognized only as the first word of a command and when not quoted: `! if then else elif fi case esac for while until do done { } function select time [[ ]]`

Comments

A word beginning with `#` causes that word and all the following characters up to a new-line to be ignored.

Aliasing

The first word of each command is replaced by the text of an `alias`, if an `alias` for this word has been defined. An `alias` name consists of any number of characters excluding metacharacters, quoting characters, file expansion characters, parameter and command substitution characters, and =. The replacement string can contain any valid shell script, including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for additional aliases. If the last character of the alias value is a *blank*, the word following the alias is also checked for alias substitution. Aliases can be used to redefine special built-in commands, but cannot be used to redefine the keywords listed above. Aliases can be created, listed, and exported with the `alias` command and can be removed with the `unalias` command. Exported aliases remain in effect for subshells but must be reinitialized for separate invocations of the shell (see *Invocation* below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for it to take effect, `alias` must be executed before the command referring to the alias is read.

Aliases are frequently used as a shorthand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full path name of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the identifier is read and becomes undefined each time the `PATH` variable is reset. These aliases remain *tracked* so that the next reference will redefine the value. Several tracked aliases are compiled into the shell. The `-h` option of the `set` command converts each command name that is an *identifier* into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t -'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
stop='kill -STOP'
suspend='kill -STOP $$'
true=':'
type='whence -v'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, the word up to a `/` is checked to see if it matches a user name in the `/etc/passwd` file. If a match is found, the ~ and the matched login name are replaced by the login directory of the matched user. This is called a tilde substitution. If no match is found, the original text is left unchanged. A ~, alone or before a /, is replaced by the value of the `HOME` parameter. A ~ followed by a + or - is replaced by the value of the parameter `PWD` and `OLDPWD`, respectively. In addition, tilde substitution is attempted when the value of a parameter assignment begins with a ~.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign (`$()`) or a pair of grave accents (` `` `) can be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed (see *Quoting* below). The command substitution `$(cat file)` can be replaced by the equivalent but faster `$(<file)`. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

An arithmetic expression enclosed in double parenthesis preceded by a dollar sign (`$(())`) is replaced by the value of the arithmetic expression within the double parenthesis (See *Arithmetic Evaluation* below for a description of arithmetic expressions).

Parameter Substitution

A *parameter* is an *identifier*, one or more digits, or any of the characters `*`,
`@`, `#`, `?`, `-`, `$`, and `!` . A *named parameter* (a parameter denoted by an identifier) has a value and zero or more attributes. Named parameters can be assigned values and attributes by using the `typeset` special command. Attributes supported by `sh` are described later with the `typeset` special command. Exported parameters pass values and attributes to the environment.

The shell supports a limited one-dimensional array facility. An element of an array parameter is referenced by a subscript. A subscript is denoted by a `[`, followed by an arithmetic expression (see *Arithmetic Evaluation* below) followed by a `]`. To assign values to an array, use `set -A` *name value ...*. The value of all subscripts must be in the range of `0` through `1023`. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array is created if necessary. Referencing an array without a subscript is equivalent to referencing the first element.

The value of a named parameter may also be assigned by writing:

name=*value* [*name*=*value*] ...

If the `-i` integer attribute is set for *name*, the *value* is subject to arithmetic evaluation as described below.

Positional parameters, parameters denoted by a number, can be assigned values with the `set` special command. Parameter `$0` is set from argument zero when the shell is invoked.

The character `$` is used to introduce substitutable *parameters*.

`${` *parameter* `}`
: Substitute the value of the parameter, if any. Braces are required when *parameter* is followed by a letter, digit, or underscore that should not be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits, it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is `*` or `@`, all the positional parameters, starting with `$1`, are substituted (separated by a field separator character). If an array *identifier* with subscript `*` or `@` is used, the value for each element is substituted (separated by a field separator character). The shell reads all the characters from `${` to the matching `}` as part of the same word even if it contains braces or metacharacters.

`${#`*parameter* `}`
: If *parameter* is `*` or `@`, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

`${#`*identifier* `[*]}`
: Substitute the number of elements in the array *identifier*.

`${` *parameter* `:-` *word* `}`
: If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

`${` *parameter* `:=` *word* `}`
: If *parameter* is not set or is null, set it to *word*; then substitute the value of the parameter. Positional parameters may not be assigned in this way.

`${` *parameter* `:?` *word* `}`
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

`${` *parameter* `:+` *word* `}`
: If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

`${` *parameter* `#`*pattern* `}`
`${` *parameter* `##`*pattern* `}`
: If the shell *pattern* matches the beginning of the value of *parameter*, the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise the value of this *parameter* is substituted. In the former case, the smallest matching pattern is deleted; in the latter case, the largest matching pattern is deleted.

`${` *parameter* `%`*pattern* `}`
`${` *parameter* `%%`*pattern* `}`
: If the shell *pattern* matches the end of the value of *parameter*, the value of *parameter* with the matched part is deleted; otherwise substitute the value of *parameter*. In the former, the smallest matching pattern is deleted; in the latter, the largest matching pattern is deleted.

In the above, *word* is not evaluated unless it is used as the substituted string. Thus, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo  ${d:-$(pwd)}
```

If the colon (`:`) is omitted from the above expressions, the shell only checks to determine whether or not *parameter* is set.

The following parameters are set automatically by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the `set` command.
?	The decimal value returned by the last executed command.
$	The process number of this shell.
_	Initially, the value of _ is an absolute pathname of the shell or script being executed as passed in the *environment*. Subsequently it is assigned the last argument of the previous

	command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail.
!	The process number of the last background command invoked.
ERRNO	The value of `errno` as set by the most recently failed system call. This value is system-dependent and is intended for debugging purposes.
LINENO	The line number of the current line within the script or function being executed.
OLDPWD	The previous working directory set by the `cd` command.
OPTARG	The value of the last option argument processed by the `getopts` special command.
OPTERR	If set to 0 **OPTERR** will suppress error messages from the `getopts` special command. **OPTERR** is initially set to 1.
OPTIND	The index of the last option argument processed by the `getopts` special command.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the `cd` command.
RANDOM	Each time this parameter is evaluated, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.
REPLY	This parameter is set by the `select` statement and by the `read` special command when no arguments are supplied.
SECONDS	Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, the value returned upon reference is the value that was assigned plus the number of seconds since the assignment.

The following parameters are used by the shell:

CDPATH	The search path for the `cd` command.
COLUMNS	If this variable is set, its value is used to define the width of the edit window for the shell edit modes and for printing `select` lists.
EDITOR	If the value of this variable ends in **emacs**, **gmacs**, or **vi** and the **VISUAL** variable is not set, the corresponding option is turned on (see `set` in *Special Commands* below).
ENV	If this parameter is set, parameter substitution is performed on the value to generate the path name of the script to be executed when the shell is invoked (see *Invocation* below). This file is typically used for *alias* and *function* definitions.
FCEDIT	The default editor name for the `fc` command.
FPATH	The search path for function definitions. This path is searched when a function with the `-u` attribute is referenced and when a command is not found. If an executable file is found, then it is read and executed in the current environment.
IFS	Internal field separators, normally space, tab, and new-line that are used to separate command words resulting from command or parameter substitution and for separating words with the special command **read**. The first character of the **IFS** parameter is used to separate arguments for the `"$*"` substitution (see *Quoting* below). If the value of **IFS** is space, tab, and new-line, or if **IFS** is unset and it is being used to separate the results of command or parameter substitution, any sequence of **IFS** characters serves to delimit words; otherwise, each occurrence of a character in **IFS** serves to delimit a word. If the value of **IFS** is null, no word splitting is done.
HISTFILE	If this parameter is set when the shell is invoked, its value is the path name of the file that is used to store the command history. The default value is **$HOME/.sh_history**. If the user is super-user and no **HISTFILE** is given, then no history file is used. (See *Command Re-entry* below.)
HISTSIZE	If this parameter is set when the shell is invoked, the number of previously entered commands accessible to this shell will be greater than or equal to this number. The default is `128`.
HOME	The default argument (home directory) for the `cd` command.
LINES	If this variable is set, the value is used to determine the column length for printing `select` lists. `Select` lists print vertically until about two-thirds of **LINES** lines are filled.
MAIL	If this parameter is set to the name of a mail file and the **MAILPATH** parameter is not set, the shell informs the user of arrival of mail in the specified file.
MAILCHECK	This variable specifies how often (in seconds) the shell checks for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is `600` seconds. When the time has elapsed the shell checks before

	issuing the next prompt.
`MAILPATH`	A list of file names separated by colons (:). If this parameter is set, the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a `?` and a message to be printed, in which case the message will undergo parameter and command substitution with the parameter **$_** defined as the name of the changed file. The default message is *you have mail in $_.*
`PATH`	The search path for commands (see *Execution* below).
`PS1`	The value of this parameter is expanded for parameter substitution, to define the primary prompt string which, by default, is "**$** ". The character **!** in the primary prompt string is replaced by the command number (see *Command Re-entry* below).
`PS2`	Secondary prompt string, by default "**>** ".
`PS3`	Selection prompt string used within a `select` loop, by default "**#?** ".
`PS4`	The value of this variable is expanded for parameter substitution and precedes each line of an execution trace. If **PS4** is unset, the execution trace prompt is "**+** ".
`SHELL`	The path name of the shell is kept in the environment. When invoked, the shell is restricted if the value of this variable contains an `r` in the basename.
`TMOUT`	If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the `PS1` prompt. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded).
`VISUAL`	Invokes the corresponding option when the value of this variable ends in **emacs**, **gmacs**, or `vi`. (See `set` in *Special Commands* below.)

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, **TMOUT**, and **IFS**. On the other hand, **HOME**, **SHELL**, **ENV**, and **MAIL** are never set automatically by the shell (although **HOME**, **SHELL**, and **MAIL** are set by `login` – see *login*(1)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for field separator characters (found in `IFS`), and split into distinct arguments where such characters are found. `sh` retains explicit null arguments ("" or '') but removes implicit null arguments (those resulting from *parameters* that have no values).

File Name Generation

Following substitution, each command *word* is processed as a pattern for file name expansion unless the `-f` option has been `set`. The form of the patterns is the Pattern Matching Notation defined by *regexp*(5). The word is replaced with sorted file names matching the pattern. If no file name is found that matches the pattern, the word is left unchanged.

In addition to the notation described in *regexp*(5), `sh` recognizes composite patterns made up of one or more pattern lists separated from each other with a **|**. Composite patterns can be formed with one or more of the following:

? (*pattern-list*)	Optionally matches any one of the given patterns.
* (*pattern-list*)	Matches zero or more occurrences of the given patterns.
+ (*pattern-list*)	Matches one or more occurrences of the given patterns.
attern-list)	
	Matches exactly one of the given patterns.
! (*pattern-list*)	Matches anything, except one of the given patterns.

Quoting

Each of the *metacharacters* listed above (See *Definitions* above) has a special meaning to the shell and causes termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \\new-line is ignored. All characters enclosed between a pair of single quote marks (' ') are quoted. A single quote cannot appear within single quotes. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, **'**, **"**, and **$**. The meaning of **$*** and **$@** is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, `"$*"` is equivalent to `"$1`*d*`$2`*d*`..."`, (where *d* is the first character of the `IFS` parameter), whereas **$@** is equivalent to `"$1" "$2"` .... Inside grave quote marks (' ') **** quotes the characters ****, **'**, and **$**. If the grave quotes occur within double quotes, \ also quotes the character **"**.

The special meaning of keywords or aliases can be removed by quoting any character of the keyword. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

The ability to perform integer arithmetic is provided with the special command `let`. Evaluations are performed using long arithmetic. Constants take the form [*base*#]*n*, where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted, base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression of the C language. All the integral operators, other than ++, - -, `?:`, and `,` are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a *variable* can be specified with the `-i` option of the `typeset` special command. Arithmetic evaluation is performed on the value of each assignment to a variable with the `-i` attribute. If you do not specify an arithmetic base, the first assignment to the variable determines the arithmetic base. This base is used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the `let` command is provided. For any command beginning with `((`, all characters until the matching `))` are treated as a quoted expression. More precisely, `((... ))` is equivalent to `let "..."`.

Prompting

When used interactively, the shell prompts with the value of `PS1` before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (the value of `PS2`) is issued.

Conditional Expressions.

A *conditional expression* is used with the `[[` compound command to test attributes of files and to compare strings. Word splitting and file name generation are not performed on the words between `[[` and `]]`. Each expression can be constructed from one or more of the following unary or binary expressions:

`-a` *file*	True, if *file* exists.
`-b` *file*	True, if *file* exists and is a block special file.
`-c` *file*	True, if *file* exists and is a character special file.
`-d` *file*	True, if *file* exists and is a directory.
`-e` *file*	True, if *file* exists.
`-f` *file*	True, if *file* exists and is an ordinary file.
`-g` *file*	True, if *file* exists and is has its setgid bit set.
`-h` *file*	True, if *file* exists and is a a symbolic link.
`-k` *file*	True, if *file* exists and is has its sticky bit set.
`-n` *string*	True, if length of *string* is non-zero.
`-o` *option*	True, if option named *option* is on.
`-p` *file*	True, if *file* exists and is a fifo special file or a pipe.
`-r` *file*	True, if *file* exists and is readable by current process.
`-s` *file*	True, if *file* exists and has size greater than zero.
`-t` *fildes*	True, if file descriptor number *fildes* is open and associated with a terminal device.
`-u` *file*	True, if *file* exists and is has its setuid bit set.
`-w` *file*	True, if *file* exists and is writable by current process.
`-x` *file*	True, if *file* exists and is executable by current process. If *file* exists and is a directory, then the current process has permission to search in the directory.
`-z` *string*	True, if length of *string* is zero.
`-H` *file*	True, if *file* exists and is a hidden directory (see *cdf*(4)).
`-L` *file*	True, if *file* exists and is a symbolic link.
`-O` *file*	True, if *file* exists and is owned by the effective user id of this process.
`-G` *file*	True, if *file* exists and its group matches the effective group id of this process.
`-S` *file*	True, if *file* exists and is a socket.
file1 `-nt` *file2*	True, if *file1* exists and is newer than *file2*.
file1 `-ot` *file2*	True, if *file1* exists and is older than *file2*.
file1 `-ef` *file2*	True, if *file1* and *file2* exist and refer to the same file.

string = *pattern*	True, if *string* matches *pattern*.
string != *pattern*	True, if *string* does not match *pattern*.
string < *string2*	True, if *string1* comes before *string2* based on ASCII value of their characters.
string > *string2*	True, if *string1* comes after *string2* based on ASCII value of their characters.
exp1 `-eq` *exp2*	True, if *exp1* is equal to *exp2*.
exp1 `-ne` *exp2*	True, if *exp1* is not equal to *exp2*.
exp1 `-lt` *exp2*	True, if *exp1* is less than *exp2*.
exp1 `-gt` *exp2*	True, if *exp1* is greater than *exp2*.
exp1 `-le` *exp2*	True, if *exp1* is less than or equal to *exp2*.
exp1 `-ge` *exp2*	True, if *exp1* is greater than or equal to *exp2*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

(*expression*)	True, if *expression* is true. Used to group expressions.
!*expression*	True if *expression* is false.
expression1 && *expression2*	True, if *expression1* and *expression2* are both true.
expression1 || *expression2*	True, if either *expression1* or *expression2* is true.

Input/Output

Before a command is executed, its input and output can be redirected using a special notation interpreted by the shell. The following can appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used, except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<*word*	Use file *word* as standard input (file descriptor `0`).
>*word*	Use file *word* as standard output (file descriptor `1`). If the file does not exist, it is created. If the file exists, and the `noclobber` option is on, an error occurs; otherwise, the file is truncated to zero length.
>|*word*	Sames as >, except that it overrides the `noclobber` option.
>>*word*	Use file *word* as standard output. If the file exists, output is appended to it (by first searching for the end-of-file); otherwise, the file is created.
<>*word*	Open file *word* for reading and writing as standard input.
<<[-]*word*	The shell input is read up to a line that matches *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document. Otherwise, parameter and command substitution occurs, \new-line is ignored, and \ must be used to quote the characters \, $, ', and the first character of *word*. If - is appended to <<, all leading tabs are stripped from *word* and from the document.
<&*digit*	The standard input is duplicated from file descriptor *digit* (see *dup*(2)).
>&*digit*	The standard output is duplicated to file descriptor *digit* (see *dup*(2)).
<&-	The standard input is closed.
>&-	The standard output is closed.
<&p	The input from the co-process is moved to standard input.
>&p	The output to the co-process is moved to standard output.

If one of the above is preceded by a digit, the file descriptor number cited is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

Redirection order is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) assignment at the time of evaluation. For example:

... `1>` *fname* `2>&1`

first assigns file descriptor 1 to file *fname*. It then assigns file descriptor 2 to the file assigned to file descriptor 1 (that is, *fname*). If the order of redirection were reversed, file descriptor 2 would be assigned to the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be assigned to file *fname*.

The input and output of a *co-process* may be moved to a numbered file descriptor allowing other commands to write to them and read from them using the above redirection operators. If the input of the current *co-process* is moved to a numbered file descriptor, another *co-process* may be started.

If a command is followed by `&` and job control is inactive, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see *environ*(5)) is a list of name-value pairs passed to an executed program much like a normal argument list. The names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. When invoked, the shell scans the environment and creates a parameter for each name found, gives it the corresponding value and marks it *export*. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones by using the `export` or `typeset -x` commands, the values become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in `export` or `typeset -x` commands.

The environment for any *simple-command* or function can be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument takes the form *identifier=value*. For example,

`TERM=450` *cmd args* and
`(export TERM; TERM=450;` *cmd args* `)`

are equivalent (as far as the above execution of *cmd* is concerned except for special commands listed below that are preceded with a dagger).

If the `-k` option is set, all parameter assignment arguments are placed in the environment, even if they occur after the command name. The following echo statement prints `a=b c`. After the `-k` option is set, the second echo statement prints only `c`:

```
echo a=b c
set  -k
echo a=b c
```

This feature is intended for use with scripts written for early versions of the shell and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions

The `function` keyword (described in the *Commands* section above) is used to define shell functions. Shell functions are read and stored internally. Alias names are resolved when the function is read. Functions are executed like commands, with the arguments passed as positional parameters. (See *Execution* below.)

Functions execute in the same process as the caller and share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on `EXIT` set inside a function is executed after the function completes in the environment of the caller. Ordinarily, variables are shared between the calling program and the function. However, the `typeset` special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the `+f` option of the `typeset` special command. Function identifiers and the associated text of the functions can be listed with the `-f` option. Functions can be undefined with the `-f` option of the `unset` special command.

Ordinarily, functions are unset when the shell executes a shell script. The `-xf` option of the `typeset` command allows a function to be exported to scripts that are executed without reinvoking the shell. Functions that must be defined across separate invocations of the shell should be placed in the **ENV** file.

Jobs

If the `monitor` option of the `set` command is turned on, an interactive shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the `jobs` command, and assigns them small integer numbers. When a job is started asynchronously with `&`, the shell prints a line that looks like:

```
[1] 1234
```

indicating job number 1 was started asynchronously and had one (top-level) process whose process ID was 1234.

If you are running a job and wish to do something else, you can type the suspend character (usually `^Z` (Ctrl-Z)) to send a **STOP** signal to the current job. The shell then indicates that the job has been 'Stopped', and prints another prompt. The user can then manipulate the state of this job by putting it in the background with the `bg` command, running other commands, and eventually returning the job to the foreground with the `fg` command. A `^Z` takes effect immediately and resembles an interrupt, since pending output and unread input are discarded when `^Z` is typed.

A job run in the background stops if it tries to read from the terminal. Background jobs normally are allowed to produce output, but can be disabled by giving the `stty tostop` command. If the user sets this tty option, background jobs stop when trying to produce output.

There are several ways to refer to jobs in the shell. A job can be referred to by the process ID of any process in the job or by one of the following:

%*number*	The job with the given number.
%*string*	Any job whose command line begins with *string*.
%?*string*	Any job whose command line contains *string*.
%%	Current job.
%+	Equivalent to %%.
%-	Previous job.

The shell learns immediately when a process changes state. It informs the user when a job is blocked and prevented from further progress, but only just before it prints a prompt.

When the monitor mode is on, each background job that completes triggers any trap set for **CHLD**.

If you try to leave the shell while jobs are running or stopped, you are warned, "You have stopped (running) jobs." You can use the `jobs` command to identify them. If you immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

Signals

The **INT** and **QUIT** signals for an invoked command are ignored if the command is followed by `&` and the `monitor` option is off. Otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the `trap` command below).

Execution

Substitutions are made each time a command is executed. `sh` checks the command name to determine whether it matches one of the user-defined functions. If it does, `sh` saves the positional parameters, then sets them to the arguments of the *function* call. When the *function* completes or issues a `return`, `sh` restores the positional parameter list and executes any trap set on **EXIT** within the function. The value of a *function* is the value of the last command executed. A function is executed in the current shell process. Next, `sh` checks the command name to determine whether it matches one of the *Special Commands* listed below. If it does, it is executed within the current shell process. If a command name is not a user-defined *function* or a *special command*, `sh` creates a process and attempts to execute the command using `exec()` (see *exec*(2)).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (`:`). The default path is `/bin:/usr/bin:` (specifying `/bin`,

`/usr/bin`, and the current directory in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between colon delimiters, or at the end of the path list. The search path is not used if the command name contains a `/`. Otherwise, each directory in the path is searched for an executable file. If the file has execute permissions but is not a directory or an executable object code file, it is assumed to be a script file, which is a file of data for an interpreter. If the first two characters of the script file are `#!`, `exec()` expects an interpreter path name to follow. `exec()` then attempts to execute the specified interpreter as a separate process to read the entire script file. If a call to `exec()` fails, `sh` is spawned to interpret the script file. All non-exported aliases, functions, and named parameters are removed in this case. If the shell command file does not have read permission, or if the *setuid* and/or *setgid* bits are set on the file, the shell executes an agent to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is also executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last `HISTSIZE` (default 128) commands entered from a terminal device is saved in a *history* file. The file `$HOME/.sh_history` is used if the `HISTFILE` variable is not set or writable. A shell can access the commands of all *interactive* shells that use the same named `HISTFILE`. The special command `fc` is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to `fc`, the value of the parameter `FCEDIT` is used. If `FCEDIT` is not defined, `/bin/ed` is used. The edited command is printed and re-executed upon leaving the editor. The editor name `-` is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if `r` is aliased to `fc -e -`, typing `r bad=good c` re-executes the most recent command that starts with the letter `c` and replaces the first occurrence of the string `bad` with the string `good`.

In-line Editing Options

Normally, each command line typed at a terminal device is followed by a new-line ('Return' or 'Line-Feed'). If either the `emacs`, `gmacs`, or `vi` option is set, the user can edit the command line. An editing option is automatically selected each time the `VISUAL` or `EDITOR` variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept Return as carriage-return without line-feed and that a space (' ') must overwrite the current character on the screen. ADM terminal users should set the "space – advance" switch to "space". Hewlett-Packard terminal users should set the straps to 'bcGHxZ etX'.

The editing modes enable the user to look through a window at the current line. The default window width is 80, unless the value of `COLUMNS` is defined. If the line is longer than the window width minus two, a mark displayed at the end of the window notifies the user. The mark is a `>`, `<`, or `*` if the line extends on the right, left, or both sides of the window, respectively. As the cursor moves and reaches the window boundaries, the window is centered about the cursor.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading `^` in the string restricts the match to begin at the first character in the line.

Emacs Editing Mode

This mode is invoked by either the `emacs` or `gmacs` option. Their sole difference is their handling of `^T`. To edit, the user moves the cursor to the point needing correction and inserts or deletes characters or words. All editing commands are control characters or escape sequences. The notation for control characters is caret (`^`) followed by the character. For example, `^F` is the notation for control-**F**. This is entered by depressing 'f' while holding down the 'Ctrl' (control) key. The 'Shift' key is *not* depressed. (The notation `^?` indicates the DEL (delete) key.)

The notation for escape sequences is `M-` followed by a character. For example, `M-f` (pronounced Meta f) is entered by depressing ESC (ASCII `033`) followed by 'f'. (`M-F` would be the notation for ESC followed by 'Shift' (capital) 'F'.)

All edit commands operate from any place on the line (not only at the beginning). Neither the "Return" nor the "Line Feed" key is entered after edit commands, except when noted.

`^F`	Move cursor forward (right) one character.
`M-f`	Move cursor forward one word. (The editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)

`^B`	Move cursor backward (left) one character.
`M-b`	Move cursor backward one word.
`^A`	Move cursor to start of line.
`^E`	Move cursor to end of line.
`^]`*char*	Move cursor forward to character *char* on current line.
`M-^]`*char*	Move cursor backward to character *char* on current line.
`^X^X`	Interchange the cursor and mark.
erase	(User defined erase character as defined by the **stty** command, usually `^H` or `#`.) Delete previous character.
`^D`	Delete current character.
eof	End-of-file character, normally `^D`, terminates the shell if the current line is null.
`M-d`	Delete current word.
`M-^H`	(Meta-backspace) Delete previous word.
`M-h`	Delete previous word.
`M-^?`	(Meta-DEL) Delete previous word (if your interrupt character is `^?` (DEL, the default) this command will not work).
`^T`	Transpose current character with next character in **emacs** mode. Transpose two previous characters in **gmacs** mode.
`^C`	Capitalize current character.
`M-c`	Capitalize current word.
`M-l`	Change the current word to lowercase.
`^K`	Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less that the current cursor position, then delete from the given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from the cursor up to the given position.
`^W`	Kill from the cursor to the mark.
`M-p`	Push the region from the cursor to the mark on the stack.
kill	(User-defined kill character, as defined by the *stty*(1) command, usually `^G` or `@`.) Kill the entire current line. If two *kill* characters are entered in succession, all subsequent consecutive kill characters cause a line feed (useful when using paper terminals).
`^Y`	Restore last item removed from line. (Yank item back to the line.)
`^L`	Line feed and print current line.
`^@`	(Null character) Set mark.
`M-`	(Meta space) Set mark.
`^J`	(New-line) Execute the current line.
`^M`	(Return) Execute the current line.
`^P`	Fetch previous command. Each time `^P` is entered, the previous command in the history list is accessed.
`^N`	Fetch next command. Each time `^N` is entered the next command in the history list is accessed.
`M-<`	Fetch the least recent (oldest) history line.
`M->`	Fetch the most recent (youngest) history line.
`^R`*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *String* is terminated by a "Return" or "New-Line". If *string* is preceded by a `^`, the matched line must begin with *string*. If *string* is omitted, the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.
`^O`	Operate – Execute the current line and fetch the next line relative to current line from the history file.
`M-`*digits*	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are `^F`, `^B`, *erase*, `^C`, `^D`, `^K`, `^R`, `^P`, `^N`, `^]`, `M-.`, `M-_`, `M-b`, `M-c`, `M-d`, `M-f`, `M-h`, `M-l` and `M-^H`.
`M-`*letter*	Soft-key – Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value is inserted on the input queue. This *letter* must not be one of the above meta-functions.
`M-.`	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.

`M-_`	Same as `M-.`.
`M-*`	Attempt file name generation on the current word.
`M-ESC`	File name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.
`M-=`	List files matching current word pattern as if an asterisk were appended.
`^U`	Multiply parameter of next command by 4.
`\`	Escape next character. Editing characters, the user's erase, kill and interrupt (normally `^?`) characters may be entered in a command line or in a search string if preceded by a `\`. The `\` removes the next character's editing features (if any).
`^V`	Display version of the shell.
`M-#`	Insert a `#` at the beginning of the line and execute it. This causes a comment to be inserted in the history file.

Vi Editing Mode

There are two typing modes. Entering a command puts you into *input* mode. To edit, the user enters *control* mode by typing ESC and moves the cursor to the point needing correction, then inserts or deletes characters or words. Most control commands accept an optional repeat *count* prior to the command.

In vi mode on most systems, canonical processing is initially enabled and the command is echoed again if the speed is 1200 baud or greater and contains any control characters, or if less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

Setting the `viraw` option always disables canonical processing on the terminal. This mode is implicit for systems that do not support two alternate end-of-line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase	Delete previous character. (*Erase* is a user-defined erase character, as defined by the `stty` command, usually `^H` or `#`.)
`^W`	Delete the previous blank separated word.
`^D`	Terminate the shell.
`^V`	Escape next character. Editing characters, erase or kill characters may be entered in a command line or in a search string if preceded by a `^V`. `^V` removes the next character's editing features (if any).
`\`	Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands move the cursor. The designation [*count*] causes a repetition of the command the cited number of times.

[*count*]`l`	Cursor forward (right) one character.
[*count*]`w`	Cursor forward one alphanumeric word.
[*count*]`W`	Cursor to the beginning of the next word that follows a blank.
[*count*]`e`	Cursor to end of word.
[*count*]`E`	Cursor to end of the current blank-delimited word.
[*count*]`h`	Cursor backward (left) one character.
[*count*]`b`	Cursor backward one word.
[*count*]`B`	Cursor to preceding blank separated word.
[*count*]`|`	Cursor to column *count*. Default is 1.
[*count*]`f`*c*	Find the next character *c* in the current line.
[*count*]`F`*c*	Find the previous character *c* in the current line.
[*count*]`t`*c*	Equivalent to `f` followed by `h`.
[*count*]`T`*c*	Equivalent to `F` followed by `l`.
[*count*]`;`	Repeats the last single character find command, `f`, `F`, `t`, or `T`.
[*count*]`,`	Reverses the last single character find command.
`0`	Cursor to start of line.
`^`	Cursor to first nonblank character in line.
`$`	Cursor to end of line.

Search Edit Commands

These commands access your command history.

[*count*] k	Fetch previous command. Each time k is entered, the next earlier command in the history list is accessed.
[*count*]-	Equivalent to k.
[*count*]j	Fetch next command. Each time j is entered, the next later command in the history list is accessed.
[*count*]+	Equivalent to j.
[*count*]G	The command number *count* is fetched. The default is the first command in the history list.
/*string*	Search backward through history for a previous command containing *string*. *string* is terminated by a "Return" or "New-line". If *string* is preceded by a ^, the matched line must begin with *string*. If *string* is null, the previous string is used.
?*string*	Same as / but search in the forward direction.
n	Search for next match of the last pattern to / or ? commands.
N	Search for next match of the last pattern to / or ?, but in reverse direction. Search history for the *string* entered by the previous / command.

Text Modification Edit Commands

These commands will modify the line.

a	Enter input mode and enter text after the current character.
A	Append text to the end of the line. Equivalent to $a.
[*count*]c*motion*	
c[*count*]*motion*	Move cursor to the character position specified by *motion*, deleting all characters between the original cursor position and new position, and enter input mode. If *motion* is c, the entire line is deleted and input mode entered.
C	Delete the current character through the end of line and enter input mode. Equivalent to c$.
S	Equivalent to cc.
D	Delete the current character through the end of line. Equivalent to d$.
[*count*]d*motion*	
d[*count*]*motion*	Move cursor to the character position specified by *motion*, deleting all characters between the original cursor position and new position. If *motion* is d, the entire line will be deleted.
i	Enter input mode and insert text before the current character.
I	Insert text before the beginning of the line. Equivalent to the two-character sequence 0i.
[*count*]P	Place the previous text modification before the cursor.
[*count*]p	Place the previous text modification after the cursor.
R	Enter input mode and replace characters on the screen with characters you type overlay fashion.
[*count*]r*c*	Replace the current character with *c*.
[*count*]x	Delete current character.
[*count*]X	Delete preceding character.
[*count*].	Repeat the previous text modification command.
~	Invert the case of the current character and advance the cursor.
[*count*]_	Causes the *count* word of the previous command to be appended at the current cursor location and places the editor in input mode at the end of the appended text. The last word is used if *count* is omitted.
*	Appends an * to the current word and attempts file name generation. If no match is found, the bell rings. If a match is found, the word is replaced by the matching string and the command places the editor in input mode.
ESC	
\	Attempt file name completion on the current word. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

Other Edit Commands

[*count*]y*motion*	
y[*count*]*motion*	Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.
Y	Yanks from current position to end of line. Equivalent to y$.

u	Undo the last text modifying command.
U	Undo all the text modifying commands performed on the line.
[*count*]**v**	Returns the command `fc -e ${VISUAL:-${EDITOR:-vi}}` *count* in the input buffer. If *count* is omitted, the current line is used.
^L	Line feed and print current line. Has effect only in control mode.
^J	(New line) Execute the current line, regardless of mode.
^M	(Return) Execute the current line, regardless of mode.
#	Equivalent to **I#** [Return]. Sends the line after inserting a **#** in front of the line and after each new-line. Useful for inserting the current command line in the history list without executing it.
=	List the filenames that match the current word if an asterisk were appended to it.
@letter	The user's alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value is inserted on the input queue for processing.

Special Commands

The following simple commands are executed in the shell process. They permit input/output redirection. Unless otherwise indicated, file descriptor 1 is the default output location and the exit status, when there are no syntax errors, is zero. Commands that are marked with a † or †† are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words following a command preceded by †† that are in the format of a variable assignment are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the = sign and word-splitting and file-name generation are not performed.

†**:** [*arg* ...]
The command only expands parameters. A zero exit code is returned.

†**.***file*[*arg* ...]
Read and execute commands from *file* and return. The commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed.

††**alias** [**-tx**] [*name*[=*value*] ...]
Alias with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **-t** option is used to set and list tracked aliases. The value of a tracked alias is the full path name corresponding to the given *name*. The value of a tracked alias becomes undefined when the value of **PATH** is reset, but the alias remains tracked. Without the **-t** option, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **-x** option is used to set or print exported aliases. An exported alias is defined across sub-shell environments. Alias returns true unless a *name* is given for which no alias has been defined.

bg [*job* ...]
Puts the specified *jobs* into the background. The current job is put in the background if *job* is unspecified. See *Jobs* for a description of the format of *job*.

†**break** [*n*]
Exit from the enclosing **for while until** or **select** loop, if any. If *n* is specified, break *n* levels.

cd [**-L**|**-P**] [*arg*]
cd *old new*
This command can take either of two forms. In the first form, it changes the current directory to *arg*. If *arg* is -, the directory is changed to the previous directory. The **-L** option (default) preserves logical naming when treating symbolic links. **cd -L ..** moves the current directory one path component closer to the root directory. The **-P** option preserves the physical

path when treating symbolic links. **cd -P ..** changes the working directory to the parent directory of the current directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (`:`). If **CDPATH** is null or undefined, the default value is the current directory. Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a `/`, the search path is not used. Otherwise, each directory in the path is searched for *arg*.
The second form of `cd` substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.

`command` [*arg* ...]
: Treat *arg* as a command, but disable function lookup on *arg*.

†`continue` [*n*]
: Resume the next iteration of the enclosing `for while until` or `select` loop. If *n* is specified, resume at the *n*-th enclosing loop.

`echo` [*arg* ...]
: See *echo*(1) for usage and description.

†`eval` [*arg* ...]
: Reads the arguments as input to the shell and executes the resulting command(s).

†`exec` [*arg* ...]
: Parameter assignments remain in effect after the command completes. If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given, the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 opened with this mechanism are closed when invoking another program.

†`exit` [*n*]
: Causes the shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end-of-file also causes the shell to exit, except when a shell has the *ignoreeof* option set. (See `set` below.)

††`export` [*name* [=*value*] ...]
††`export -p`
: The given *names* are marked for automatic export to the *environment* of subsequently executed commands. When the -p option is specified, *export* writes to the standard output the names and values of all exported variables, in a format with the proper use of quoting, so that it is suitable for re-input to the shell as commands that achieve the same exporting results.

`fc` [`-e` *ename*] [`-nlr`] [*first* [*last*]]
`fc -e -` [*old*=*new*] [*command*]
: In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands typed at the terminal. The arguments *first* and *last* can be specified as a number or string. A given string is used to locate the most recent command. A negative number is used to offset the current command number. The `-l` option causes the commands to be listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, the value of the parameter **FCEDIT** (default `/bin/ed`) is used as the editor. Once editing has ended, the commands (if any) are executed. If *last* is omitted, only the command specified by *first* is used. If *first* is not specified, the default is the previous command for editing and −16 for listing. The `-r` option reverses the order of the commands and `-n` suppresses command numbers when listing. In the latter, *command* is re-executed after the substitution *old*=*new* is performed.

`fg` [*job* ...]
: Brings each *job* into the foreground in the order specified. If no *job* is specified, the current job is brought into the foreground. See *Jobs* for a description of the format of *job*.

`getopts` *optstring name* [*arg* ...]
: Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option

argument begins with a `+` or a `-`. An option not beginning with `+` or `-` or the argument `--` ends the options. *optstring* contains the letters that *getopts* recognizes. If a letter is followed by a `:`, that option is expected to have an argument. The options can be separated from the argument by blanks.

`getopts` places the next option letter it finds inside variable *name* each time it is invoked with a `+` prepended when *arg* begins with a `+`. The index of the next *arg* is stored in `OPTIND`. The option argument, if any, gets stored in `OPTARG`. If no option is found, or the option found does not take an argument, `OPTARG` is removed from the environment.

A leading `:` in *optstring* causes `getopts` to store the letter of an invalid option in `OPTARG`, and to set *name* to `?` for an unknown option and to `:` when a required option is missing. Otherwise, `getopts` prints an error message. The exit status is non-zero when there are no more options.

`jobs` [`-lnp`] [*job* ...]
: Lists information about each given job or all active jobs if *job* is not specified. The `-l` option lists process ids in addition to the normal information. The `-n` option only displays jobs that have stopped or exited since last notified. The `-p` option causes only the process group to be listed. See *Jobs* for a description of the format of *job*.

`kill -s` *signal_name process* ...
`kill -l`
`kill` [`-`*sig*] *process* ...
: Sends either the `TERM` (terminate) signal or the specified signal to the specified jobs or processes. Signals are given either by number or name. A signal name can be a number or one of the signals listed in *signal*(5)), with or without the `SIG` prefix. Uppercase and lowercase signal names are interpreted identically. In addition, the name `SIGNULL` is recognized and represents the signal value 0. The signal names are listed by `kill -l`. No default exists; merely typing `kill` does not affect the current job. If the signal being sent is `TERM` (terminate) or `HUP` (hangup), the job or process is sent a `CONT` (continue) signal when stopped. The *process* argument can be either a process ID or job. If the first argument to `kill` is a negative integer, it is interpreted as a *sig* argument and not as a process group.

`let` *arg* ...
: Each *arg* is a separate *arithmetic expression* to be evaluated. See *Arithmetic Evaluation* above for a description of arithmetic expression evaluation. The exit status is 0 if the value of the last expression is non-zero, and 1 otherwise.

†`newgrp` [*arg* ...]
: Equivalent to `exec newgrp` *arg* ....

`print` [`-Rnprsu`[*n*]] [*arg*`...`]
: The shell output mechanism. With no options or with option `-` or `--` the arguments are printed on standard output as described by *echo*(1). Raw mode, `-R` or `-r`, ignores the escape conventions of *echo*. The `-R` option will print all subsequent arguments and options other than `-n`. The `-p` option causes the arguments to be written onto the pipe of the process spawned with `|&` instead of standard output. The `-s` option causes the arguments to be written onto the history file instead of standard output. The `-u` option can be used to specify a one-digit file descriptor unit number *n* on which the output will be placed. The default is 1. If the `-n` option is used, no new-line character is added to the output.

`pwd` [`-L`|`-P`]
: With no arguments prints the current working directory (equivalent to `print -r - $PWD`). The `-L` option (default) preserves the logical meaning of the current directory and `-P` preserves the physical meaning of the current directory if it is a symbolic link (see *cd* and *ln*(1)).

`read` [`-prsu`[*n*]] [*name*`?`*prompt*] [*name* ...]
: The shell input mechanism. One line is read and is broken up into words using the characters in `IFS` as separators. In `-r` raw mode, `\` at the end of a line does not signify line continuation. The first word is assigned to the first *name*, the second word to the second *name*, etc., with remaining words assigned to the last *name*. The `-p` option causes the input line to be taken from the input pipe of a process spawned by the shell using `|&`. If the `-s` option is present, the input is saved as a command in the history file. The `-u` option can be used to specify a one-

digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is 0. If *name* is omitted, **REPLY** is used as the default *name*. The return code is 0, unless an end-of-file is encountered. An end-of-file with the **-p** option causes cleanup for this process so that another process can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* when the shell is interactive. If the given file descriptor is open for writing and is a terminal device, the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2. The return code is 0, unless an end-of-file is encountered.

††**readonly** [*name*[=*value*] ...]
††**readonly -p**

The given *names* are marked read only and these names cannot be changed by subsequent assignment. When the **-p** option is specified, **readonly** writes to the standard output the names and values of all read-only variables, in a format with the proper use of quoting so that it is suitable for re-input to the shell as commands that achieve the same attribute-setting results.

†**return** [*n*]

Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted, the return status is that of the last command executed. Only the low 8 bits of *n* are passed back to the caller. If **return** is invoked while not in a *function* or a **.** script, it has the same effect as an **exit** command.

set [±**aefhkmnopstuvx** | ±o *option*] ... [±**A** *name*] [*arg* ...]

The following options are used for this command:

-A	Array assignment. Unset the variable *name* and assign values sequentially from the list *arg*. If **+A** is used, the variable *name* is not unset first.
-a	All subsequent defined parameters are automatically exported.
-e	If the shell is non-interactive and if a command fails, execute the **ERR** trap, if set, and exit immediately. This mode is disabled while reading profiles.
-f	Disables file name generation.
-h	Each command whose name is an *identifier* becomes a tracked alias when first encountered.
-k	All parameter assignment arguments (not just those that precede the command name) are placed in the environment for a command.
-m	Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. This option is turned on automatically for interactive shells.
-n	Read commands and check them for syntax errors, but do not execute them. The **-n** option is ignored for interactive shells.
-o	The **-o** argument takes any of several *option* names, but only one *option* can be specified with each **-o** option. If none is supplied, the current option settings are printed. The **-o** argument *option* names follow:

allexport	Same as **-a**.
bgnice	All background jobs are run at a lower priority.
errexit	Same as **-e**.
emacs	Puts you in an **emacs** style in-line editor for command entry.
gmacs	Puts you in a **gmacs** style in-line editor for command entry.
ignoreeof	The shell will not exit on end-of-file. The command **exit** must be used.
keyword	Same as **-k**.
markdirs	All directory names resulting from file name generation have a trailing / appended.
monitor	Same as **-m**.
noclobber	Prevents redirection > from truncating existing files. Requires >| to truncate a file when turned on.
noexec	Same as **-n**.
noglob	Same as **-f**.
nolog	Do not save function definitions in history file.
nounset	Same as **-u**.
privileged	Same as **-p**.

`verbose`	Same as `-v`.
`trackall`	Same as `-h`.
`vi`	Start insert mode using a `vi`-style, in-line editor until an escape character is received (`033`). This puts you in move mode. A return sends the line.
`viraw`	Each character is processed as it is typed in *vi* mode.
`xtrace`	Same as `-x`.

`-p`
Disables processing of the `$HOME/.profile` file and uses the file `/etc/suid_profile` instead of the `ENV` file. This mode is on whenever the effective uid (gid) is not equal to the real uid (gid). Turning this off causes the effective uid and gid to be set to the real uid and gid.

`-s`
Sort the positional parameters.

`-t`
Exit after reading and executing one command.

`-u`
Treat unset parameters as an error when substituting.

`-v`
Print shell input lines as they are read.

`-x`
Print commands and their arguments as they are executed.

`-`
Turns off `-x` and `-v` options and stops examining arguments for options.

`--`
Do not change any of the options; useful in setting `$1` to a value beginning with `-`. If no arguments follow this option, the positional parameters are unset.

Using `+` instead of `-` before an option causes the option to be turned off. These options can also be used when invoking the shell. The current set of options can be examined by using `$-`.

Unless `-A` is specified, the remaining *arg* arguments are positional parameters and are assigned consecutively to `$1`, `$2`, ...

If neither arguments nor options are given, the values of all names are printed on the standard output. The `set` command followed only by `+` lists the names of all shell variables.

†`shift` [*n*]
The positional parameters from `$`*n*+1 ... are renamed `$1` ...; default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to `$#`.

`test` [*expr*]
Evaluate conditional expression *expr*. See *test*(1) for usage and description. The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. The following additional primitive expressions are allowed:

`-L` *file*	True if *file* is a symbolic link.
`-e` *file*	True if *file* exists.
file1 `-nt` *file2*	True if *file1* is newer than *file2*.
file1 `-ot` *file2*	True if *file1* is older than *file2*.
file1 `-ef` *file2*	True if *file1* has the same device and i-node number as *file2*.

†`times`
Print the accumulated user and system times for the shell and for processes run from the shell.

†`trap` [*arg*] [*sig* ...]
The *arg* is a command read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or name of the signal. Trap commands are executed in signal number order. Any attempt to set a trap on a signal that was ignored upon entering the current shell is ineffective. If *arg* is omitted or is -, all traps for *sig* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *sig* is `DEBUG`, then *arg* is executed after each command. If *sig* is `ERR`, *arg* is executed whenever a command has a non-zero exit code. If *sig* is `0` or `EXIT` and the `trap` statement is executed inside the body of a function, the command *arg* is executed after the function completes. If *sig* is `0` or `EXIT` for a `trap` set outside any function, the command *arg* is executed on exit from the shell.

The `trap` command with no arguments prints a list of commands associated with each signal number.

†† **`typeset`** [±**`LRZfilrtux`** [*n*]] [*name*[=*value*]] ...

Parameter assignments remain in effect after the command completes. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

- `-L` Left justify and remove leading blanks from *value*. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. When the *name* is assigned, the value is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the `-Z` option is also set. The `-R` option is turned off.
- `-R` Right justify and fill with leading blanks. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left-filled with blanks or truncated from the end if the parameter is reassigned. The `-L` option is turned off.
- `-Z` Right justify and fill with leading zeros if the first non-blank character is a digit and the `-L` option has not been set. If *n* is non-zero it defines the width of the field; otherwise it is determined by the width of the value of first assignment.
- `-f` Cause *name* to refer to function names rather than parameter names. No assignments can be made to the *name* declared with the **`typeset`** statement. The only other valid options are `-t` (which turns on execution tracing for this function) and `-x` (which allows the function to remain in effect across shell procedures executed in the same process environment).
- `-i` Parameter is an integer. This makes arithmetic faster. If *n* is non-zero it defines the output arithmetic base; otherwise the first assignment determines the output base.
- `-l` Convert all uppercase characters to lowercase. The uppercase `-u` option is turned off.
- `-r` Any given *name* is marked "read only" and cannot be changed by subsequent assignment.
- `-t` Tag the named parameters. Tags are user definable and have no special meaning to the shell.
- `-u` Convert all lowercase characters to uppercase characters. The lowercase `-l` option is turned off.
- `-x` Mark any given *name* for automatic export to the environment of subsequently executed commands.

Using `+` instead of `-` causes these options to be turned off. If no *name* arguments are given but options are specified, a list of names (and optionally the values) of the parameters that have these options set is printed. Using `+` instead of `-` retains the values to be printed. If neither names nor options are given, the names and attributes of all parameters are printed.

`ulimit` [*n*]

If *n* is given, impose a size limit of *n* 512-byte blocks on files written by child processes (files of any size can be read). If *n* is not given, the current limit is printed.

`umask` [`-S`] [*mask*]

The user file-creation mask is set to *mask*. *mask* can either be an octal number or a symbolic value as described in *umask*(1). If a symbolic value is given, the new umask value is the complement of the result of applying *mask* to the complement of the previous umask value. If *mask* is omitted, the current value of the mask is printed. The `-S` option prints the current value of the mask in symbolic format. The output from either format can be used as the mask operand to a subsequent invocation of *umask*.

`unalias` *name* ...
`unalias -a` The parameters given by the list of *name*s are removed from the *alias* list. The `-a` option is provided to remove all the *alias* definitions from the current shell execution environment.

`unset [-fv]` *name* ...
: The parameters given by the list of *name*s are unassigned; that is, their values and attributes are erased. Read-only variables cannot be unset. If the `-f` option is set, *name*s refer to function names. If the `-v` option is set, *name*s refer to variable names. Unsetting **ERRNO, LINENO, MAILCHECK, OPTARG, OPTIND, RANDOM, SECONDS, TMOUT,** and **_** removes their special meaning even if they are subsequently assigned to.

†`wait` [*job*]
: Wait for the specified *job* to terminate or stop, and report its status. This status becomes the return code for the `wait` command. If *job* is not given, `wait` waits for all currently active child processes to terminate or stop. The termination status returned is that of the last process. See *Jobs* for a description of the format of a *job*.

`whence [-pv]`*name* ...
: For each *name*, indicate how it would be interpreted if used as a command name. The `-v` option produces a more verbose report. The `-p` option does a path search for *name* even if *name* is an alias, a function, or a reserved word.

Shell Invocation

If the shell is invoked by `exec()`, and the first character of argument zero (`$0`) is `-`, the shell is assumed to be a login shell and commands are read first from `/etc/profile`, then from either `.profile` in the current directory or `$HOME/.profile`, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV**, if the file exists. If the `-s` option is not present and *arg* is, a path search is performed on the first *arg* to determine the name of the script to execute. When running `sh` with *arg*, the script *arg* must have read permission and any `setuid` and `getgid` settings will be ignored. Commands are then read as described below. The following options are interpreted by the shell when it is invoked:

`-c` *string*
: If the `-c` option is present, commands are read from *string*.

`-s`
: If the `-s` option is present or if no arguments remain, commands are read from the standard input. Shell output, except for the output of some of the *Special commands* listed above, is written to file descriptor 2.

`-i`
: If the `-i` option is present or if the shell input and output are attached to a terminal (as reported by `tty()`), the shell is interactive. In this case `SIGTERM` is ignored (so that `kill 0` does not kill an interactive shell) and `SIGINT` +1 is caught and ignored (so that `wait` is interruptible). In all cases, `SIGQUIT` is ignored by the shell. (See *signal*(5).)

`-r`
: If the `-r` option is present, the shell is a restricted shell.

The remaining options and arguments are described under the `set` command above.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the collating sequence used in evaluating pattern matching notation for file name generation.

`LC_CTYPE` determines the classification of characters as letters, and the characters matched by character class expressions in pattern matching notation.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5))*isusedinstead*of **LANG.** If any internationalization variable contains an invalid setting, `sh` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

RETURN VALUE

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (also see the `exit` command above). If the shell is being used non-interactively, execution of the shell file is abandoned. Runtime errors detected by the shell are reported by printing the command or function name and the error condition. If the line number on which the error occurred is greater than one, the line number is also printed in brackets (`[ ]`) after the command or function name.

WARNINGS

The file descriptors 10 and 14 through 20 are used by the POSIX shell internally. Applications using these and forking a subshell, should not depend upon them surviving in the subshell, or its descendants.

If a command which is a *tracked alias* is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to load and execute the original command. Use the `-t` option of the `alias` command to correct this situation.

If you move the current directory or one above it, `pwd` may not give the correct response. Use the `cd` command with a full path name to correct this situation.

Some very old shell scripts contain a caret (^) as a synonym for the pipe character (|). Note however, `sh` does not recognize the caret as a pipe character.

If a command is piped into a shell command, all variables set in the shell command are lost when the command completes.

Using the `fc` built-in command within a compound command will cause the entire command to disappear from the history file.

The built-in command `.` *file* reads the entire file before any commands are executed. Therefore, `alias` and `unalias` commands in the file will not apply to any functions defined in the file.

Traps are not processed while the shell is waiting for a foreground job. Thus, a trap on `CHLD` is not executed until the foreground job terminates.

The `export` built-in command does not handle arrays properly. Only the first element of an array is exported to the *environment*.

Background processes started from a non-interactive shell cannot be accessed by using job control commands.

In an international environment, character ordering is determined by the setting of `LC_COLLATE`, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particularly when using range expressions in file name generation patterns. For example, the command,

```
rm [a-z]*
```

might be expected to match all file names beginning with a lowercase alphabetic character. However, if dictionary ordering is specified by `LC_COLLATE`, it would also match file names beginning with an uppercase character (as well as those beginning with accented letters). Conversely, it would fail to match letters collated after `z` in languages such as Danish or Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form:

```
rm [[:lower:]]*
```

This uses `LC_CTYPE` to determine character classes and works predictably for all supported languages and codesets. For shell scripts produced on non-internationalized systems (or without consideration for the above dangers), it is recommended that they be executed in a non-NLS environment. This requires that `LANG`, `LC_COLLATE`, etc., be set to "C" or not set at all.

Be aware that the value of the `IFS` variable in the user's environment affects the behavior of scripts.

AUTHOR

`sh` was developed by AT&T.

FILES

`/etc/passwd`	to find home directories
`/etc/profile`	read to set up system environment
`/etc/suid_profile`	security profile
`$HOME/.profile`	read to set up user's custom environment
`/tmp/sh*`	for here-documents

SEE ALSO

cat(1), cd(1), echo(1), env(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), gtty(2), pipe(2), signal(5), umask(2), ulimit(2), wait(2), rand(3C), a.out(4), profile(4), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

`sh`: SVID2, XPG2, XPG3, POSIX.2

`.`: SVID2, XPG2, XPG3, POSIX.2

`:`: SVID2, XPG2, XPG3, POSIX.2

`break`: SVID2, XPG2, XPG3, POSIX.2

`command`: POSIX.2

`continue`: SVID2, XPG2, XPG3, POSIX.2

`eval`: SVID2, XPG2, XPG3, POSIX.2

`exec`: SVID2, XPG2, XPG3, POSIX.2

`exit`: SVID2, XPG2, XPG3, POSIX.2

`export`: SVID2, XPG2, XPG3, POSIX.2

`read`: SVID2, XPG2, XPG3, POSIX.2

`set`: SVID2, XPG2, XPG3, POSIX.2

`shift`: SVID2, XPG2, XPG3, POSIX.2

`time`: SVID2, XPG2, XPG3

`trap`: SVID2, XPG2, XPG3, POSIX.2

`unset`: SVID2, XPG2, XPG3, POSIX.2

NAME

sh - overview of various system shells

SYNOPSIS

POSIX Shell:

sh [±**aefhikmnoprstuvx**] [±o *option*] ... [-c *string*] [*arg* ...]

Bourne Shell:

sh [- -**acefhiknrstuvx** ...] [*arg* ...]
rsh [- -**acefhiknrstuvx** ...] [*arg* ...]

Korn Shell:

ksh [±**aefhikmnoprstuvx**] [± o *option*] ... [-c *string*] [*arg* ...]
rksh [±**aefhikmnoprstuvx**] [±o *option*] ... [-c *string*] [*arg* ...]

C Shell:

csh [-**cefinstvxTVX**] [*command_file*] [*argument_list* ...]

Key Shell:

keysh

DESCRIPTION

Remarks:

The POSIX.2 standard requires that, on a POSIX-compliant system, executing the command **.sh** activates the POSIX shell (located in file **/bin/posix/sh** on HP-UX systems), and executing the command **man sh** produces an on-line manual entry that displays the syntax of the POSIX shell command-line.

However, the **sh** command has historically been associated with the conventional Bourne shell, which could confuse some users. To meet standards requirements and also clarify the relationships of the various shells and where they reside on the system, this entry provides command-line syntax and a brief description of each shell, and lists the names of the manual entries where each shell is described in greater detail.

Shell Descriptions

The HP-UX operating system supports the following shells:

sh	POSIX-conformant command programming language and commands interpreter residing in file **/bin/posix/sh**. Can execute commands read from a terminal or a file. This shell conforms to current POSIX standards in effect at the time the HP-UX system release was introduced, and is similar to the Korn shell in many respects. Similar in many respects to the Korn shell, the POSIX shell contains a history mechanism, supports job control, and provides various other useful features.
sh	Bourne-shell command programming language and commands interpreter residing in file **/bin/sh**. Can execute commands read from a terminal or a file. This shell lacks many features contained in the POSIX and Korn shells.
ksh	Korn-shell command programming language and commands interpreter residing in file **/bin/ksh**. Can execute commands read from a terminal or a file. This shell, like the POSIX shell, contains a history mechanism, supports job control, and provides various other useful features.
csh	A command language interpreter that incorporates a command history buffer, C-language-like syntax, and job control facilities.
rsh	Restricted version of the Bourne-shell command interpreter. Sets up a login name and execution environment whose capabilities are more controlled (restricted) than normal user shells.
rksh	restricted version of the Korn-shell command interpreter Sets up a login name and execution environment whose capabilities are more controlled (restricted) than normal user shells.
keysh	An extension of the standard Korn Shell that uses hierarchical softkey menus and context-sensitive help.

To obtain:	Use the command:
Bourne Shell	`/bin/sh ...`
Korn Shell	`/bin/ksh ...`
POSIX Shell	`/bin/posix/sh ...`
C Shell	`/bin/csh ...`
Key Shell	`/usr/bin/keysh`

These shells can also be the default invocation, depending on the entry in the **/etc/passwd** file. See also *chsh*(1).

Whether the **sh** command invokes the Bourne Shell or the POSIX Shell depends on the setting of the **PATH** environment variable.

The default **PATH** in file **/etc/profile** is set to invoke the POSIX shell.

WARNINGS

Many manual entries contain descriptions of shell behavior or describe program or application behavior similar to "the shell" with a reference to "see *sh*(1)".

These various references to *sh*(1) generally pertain to the Bourne shell, and may or may not reflect the behavior of the POSIX shell in specific situations. Exercise due caution when noting references to shell behavior because the various shells were invented in different years by different people, and each has its own unique characteristics.

SEE ALSO

For more information on the various individual shells, see:

sh-bourne(1)	Bourne Shell (**/bin/sh**) description.
ksh(1)	Korn Shell (**/bin/ksh**) description.
sh-posix(1)	POSIX Shell (**/bin/posix/sh**) description.
csh(1)	C Shell (**/bin/csh**) description.
keysh(1)	Key Shell (**/usr/bin/keysh**) description.

NAME

shar - make a shell archive package

SYNOPSIS

shar [*options*] [*file* | *dir*] ... > *package*

DESCRIPTION

shar bundles the named files and directories into a single distribution package suitable for mailing or moving. The files can contain any data, including executables. The resulting package, written to standard output, is a shell script file that can be edited (to add messages at the beginning, etc.).

To unpack *package*, use the *sh*(1) command with the package name as an argument as follows:

> `sh` *package*

When unpacking, the files and directories in *package* are written to the path names recorded in the archive.

If a directory is specified and the `-d` option is not given, all files beneath that directory are archived.

If a special file is specified, the appropriate **mknod** commands are emitted to recreate the file (see *mknod*(1)).

shar protects the contained files from mail processing, if necessary, by inserting an @ character at the beginning of each line. If the file contains unusual data, the data is transformed into **uuencode** format, and a **uudecode** script is included in *package* so that the package can still be unpacked correctly by **sh**. See WARNINGS for more information about mailers and file modifications.

Access modes are preserved for both directories and files.

Options

shar recognizes the following options:

-a Assume that files can be shipped, regardless of their contents; do not protect them specially. **shar** is conservative, and might decide to **uuencode** a file containing special characters (such as Ctrl-G) that the user knows do not need protection.

-A Suppress warning messages regarding optional access control list entries. **shar** does not archive optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file having optional access control list entries.

-b Archive files under their base names, regardless of the original path names specified. The contents are thus unpacked into the current directory instead of to the originally specified path names. This allows you to archive files from many directories but unpack them into a single directory. It also allows you to unpack, for example, `/etc/termcap` into `./termcap` instead of overwriting the original one in `/etc`.

-c Append to the package a simple data-integrity check using **wc** to ensure that the contents were not damaged in transit (see *wc*(1)). This check is performed automatically after unpacking. Also see WARNINGS below.

-C Insert a line of the form `--- cut here ---` before the archive.

-d If a directory is specified, do not transmit its contents, but rather only create the empty directory.

-D*dir* Cause the archive to contain code that notifies the user if his or her current directory is not the same as *dir*, which must be an absolute path. If the user is not in *dir*, the unpacking can be continued by responding **yes** to the archive's question.

-e Cause the archive to contain code that prevents **shar** from unpacking files that would overwrite existing files.

-f*file* Read a list of file names from *file* and archive those files as if they were given as arguments.

-h Follow symbolic links as if they were normal files or directories. If this option is not specified, **shar** archives the link.

-m Retain modification and access times on files when they are unpacked.

`-o`	Preserve user and group ownership on files and directories.
`-r`	Cause the archive to contain code requiring that the user unpacking it be `root`. This is useful for processing system archives.
`-s`	Perform error checking using `sum` (see *sum*(1)). Both `-c` and `-s` can be specified for better error checking. Also see WARNINGS below.
`-t`	Write diagnostics and messages directly to your terminal instead of to the standard error. This is useful when invoking `shar` from programs (such as `vi` that normally combine standard error with standard output. Specifying `-t` also invokes the `-v` (verbose) option.
`-u`	Assume that the remote site has `uudecode` for unpacking. If this option is not specified, a version of `uudecode` is sent and compiled if any non-ASCII files are archived.
`-v`	Announce archived file names as they are packed. The `-t` option determines the destination for these announcements.
`-Z`	Compress files using `compress` (see *compress*(1)).

Most options are flagged in the header of the resulting package, thereby recording the format of the archive. The name of the archiver, system, and time/date of the archive are also recorded in the header.

EXAMPLES

To archive all files under your home directory, type:

```
cd; shar -cmos .
```

or

```
shar -cmos $HOME
```

To preserve your `/dev` directory, type:

```
shar -mor /dev >save_dev_files
```

To send your newest programs in directory `newstuff` in your home directory to a friend, type:

```
cd; shar -cmos newstuff | mailx -s 'new source' friend
```

RETURN VALUE

`shar` returns zero if successful; non-zero if problems with arguments occur.

DIAGNOSTICS

If the `-b` option is specified, `shar` refuses to archive directories.

WARNINGS

The modification and access time restoration does not take time zones into account.

Files with new-line characters in their names scramble the table of contents.

Non-ASCII files with white space in their names do not unpack.

If a mailer such as *elm*(1) is used to transfer *package* to another system and the mailer is configured to expand tabs (by default or otherwise), any file in the archive will be modified if it contains tabs. If the `-c` or `-s` option is used to create the archive, the data-integrity check will fail during unpacking of any files in *package* that contain tab characters that were converted to spaces. (Some mailers that expand tabs when transferring files over a network may or may not expand tabs when transferring files to the sender or other users on the local system.) If an editor is used to modify any of the files in *package*, the data-integrity check will also fail for the files that were changed.

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

AUTHOR

`shar` was invented in the public domain. This version of `shar` was developed by HP.

FILES

```
/dev/tty
/tmp/unpack$$*        (for unpacking non-ASCII files)
```

SEE ALSO

ar(1), compress(1), cpio(1), find(1), tar(1), acl(5).

NAME

shl - shell layer manager

SYNOPSIS

`shl`

DESCRIPTION

`shl` provides a means for interacting with more than one shell from a single terminal by using shell layers. A layer is a shell that is bound to a virtual device. The virtual device can be manipulated like an actual terminal by using `stty` and `ioctl()` (see *stty*(1) and *ioctl*(2)). Each layer has its own process group ID. The user controls these layers by using the commands described below.

The current layer is the layer that can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To block the output of a layer when it is not current, the `stty` option `loblk` can be set within the layer.

The `stty` character `swtch` (set to `^Z` if NUL) is used to switch control to `shl` from a layer. `shl` has its own prompt, >>>, to distinguish it from a layer.

Definitions

A *name* is a sequence of characters delimited by a space, tab, or new-line character. Only the first eight characters are significant. When provided as an argument to the `create`, `login`, or `name` commands, *name* cannot be of the form *n* or (*n*), where *n* is a decimal number.

Commands

The following commands can be issued from the `shl` prompt level. Any unique prefix is accepted.

`create` [-[*name*] | *name* [*command*]]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer is created with a name of the form (*n*), where *n* is the number of the next available slot in an internal table. Future references to this layer can be made with or without the parentheses. If *name* is followed by a command, that command is executed in the layer instead of a shell. If - is the first argument, a "login shell" is created in the layer. The shell prompt variable `PS1` is set to the name of the layer followed by a space.

`login` [*name*]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer is created with a name of the form (*n*), where *n* is the number of the next available slot in an internal table. Future references to this layer can be made with or without the parentheses. *name* is a "login" type of layer in which the user receives a prompt for user name and password.

`name` [*oldname*] *newname*
: Rename the layer *oldname*, calling it *newname*. If *oldname* is not specified, the current layer name is changed.

`!` [*command*]
: Invoke a sub-shell and execute *command*. If no *command* is given, a shell is executed according to the `SHELL` environment variable.

`block` *name* [*name* ...]
: For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the `stty loblk` option within the layer.

`delete` *name* [*name* ...]
: For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the `SIGHUP` signal (see *signal*(5)).

`help` (or `?`)
: Print the syntax of the `shl` commands.

`layers` [`-l`] [*name* ...]
: For each *name*, list the layer name and its process group. The `-l` option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

`resume` [*name*]
: Change the status of the layer referred to by *name* to that of current layer. If no argument is given, the last existing current layer is changed.

`toggle` Change the status of the previous current layer to that of current layer.

`unblock` *name* [*name* ...]
: For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the `stty -loblk` option within the layer.

`quit` Exit `shl`. All layers are sent the `SIGHUP` signal.

name Change the status of the layer referred to by *name* to that of current layer. Any unique prefix is accepted.

WARNINGS

The behavior of the `block` and `unblock shl` commands is not guaranteed when the SHELL environment variable is set to `/bin/csh` (for *csh*(1)) or `/bin/ksh` (for *ksh*(1)), or when the shell saves and restores the tty state (defined in *termio*(7)) before and after each command is invoked interactively from that shell. For both `/bin/csh` and `/bin/ksh`, the `loblk` or `-loblk` options of `stty` can be used from within the layer to block or unblock the output of that layer.

DEPENDENCIES

Series 800

The `login` command is not currently supported.

FILES

`$SHELL` Variable containing path name of the shell to use (default is `/bin/sh`).

SEE ALSO

sh(1), stty(1), ioctl(2), signal(5).

STANDARDS CONFORMANCE

`shl`: SVID2, XPG2

NAME

showcdf - show the actual path name matched for a CDF

SYNOPSIS

`showcdf [-chs]` *name* ...

DESCRIPTION

`showcdf` displays on the standard output the actual path name matched for each path name specified in the argument list. If any *name* is a context-dependent file (CDF), the actual file matched for that name is displayed. If any *name* is not a CDF, the name is simply displayed.

Options

`showcdf` recognizes the following options:

`-c`	Suppress the printing of non-CDF files. Only files whose path contains a CDF are displayed.
`-h`	Hides, rather than displays, any unnecessary parts of the specified path names. If any *name* is a context-dependent file (CDF), any portions of *name* that match the current context are removed from the path.
`-s`	Suppress all output. This option is useful when testing for the exit value from `showcdf`.

RETURN VALUE

Upon completion, `showcdf` returns one of the following exit values:

0	No CDF I c s were specified.
1	At least one CDF was specified.
2	An error occurred (for example, a file is inaccessible).

EXAMPLES

The following example shows the output of `showcdf` when applied to the files `/etc/inittab` and `/usr/adm/sulog`, which are CDFs, and `/etc/passwd`, which is not a CDF:

```
$ showcdf /etc/inittab /usr/adm/sulog /etc/passwd
/etc/inittab+/donald
/usr/adm+/donald/sulog
/etc/passwd
```

```
$ showcdf -c /etc/inittab /usr/adm/sulog /etc/passwd
/etc/inittab+/donald
/usr/adm+/donald/sulog
```

The following example shows the output of `showcdf -h` when applied to `/etc/inittab+/donald` and `/etc/inittab+/minnie`, where `donald` matches the current context, but `minnie` does not.

```
$ showcdf -h /etc/inittab+/donald /etc/inittab+/minnie
/etc/inittab
/etc/inittab+/minnie
```

AUTHOR

`showcdf` was developed by HP.

SEE ALSO

getcdf(3C), cdf(4).

NAME

size - print section sizes of object files

SYNOPSIS

`size` [`-d`] [`-o`] [`-x`] [`-V`] *files*

DESCRIPTION

`size` produces section size information for each section in the object files. The size of the text, data and bss (uninitialized data) sections are printed along with the total size of the object file. If an archive file is input to the `size` command, the information for all archive members is displayed.

Options

`size` recognizes the following options:

`-d`	Print sizes in decimal. This is the default.
`-o`	Print sizes in octal.
`-x`	Print sizes in hexadecimal.
`-V`	Supply `size`-command version information.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`size:` *name*`: cannot open`	*name* cannot be read.
`size:` *name*`: bad magic`	*name* is not an appropriate object file.

EXAMPLES

Compare the sizes of the text, data, and bss sections for two versions of a program:

```
size ./version1 ./version2
```

DEPENDENCIES

Series 300/400

The `-V` option is not supported.

Series 700/800

An additional `-v` option causes `size` to print a verbose list of the subspaces in the object files. Each subspace is listed on a separate line with its size, physical address, and virtual address.

SEE ALSO

as(1), cc(1), ld(1). a.out(4), ar(4).

STANDARDS CONFORMANCE

`size`: SVID2, XPG2

NAME

sleep - suspend execution for an interval

SYNOPSIS

`sleep` *time*

DESCRIPTION

`sleep` suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command periodically, as in:

```
while true
do
  command
     sleep 37
done
```

RETURN VALUE

`sleep` exits with one of the following values:

0 The execution was successfully suspended for *time* seconds, or a SIGALRM signal was received.

>0 If the *time* operand is missing, is not a decimal integer, is negative, or is greater than **UINT_MAX**, `sleep` returns with exit status 2.

SEE ALSO

alarm(2), sleep(3C).

STANDARDS CONFORMANCE

`sleep`: SVID2, XPG2, XPG3, POSIX.2

NAME

slp - set printing options for a non-serial printer

SYNOPSIS

`slp` [`-a`] [`-b`] [`-c` *cols*] [`-d`] [`-i` *indent*] [`-k`] [`-l` *lines*] [`-n`] [`-o`] [`-r`] [`-C` *pages*] [`-O` *pages*]

DESCRIPTION

`slp` sets printer formatting options such as the number of lines per page, number of characters per line, and indentation. These characteristics are controlled by the printer driver as described in *lp*(7). `slp` acts on the current standard output.

Options

`slp` recognizes the following options and arguments:

`-a`	Report all option settings.
`-b`	Specify a character printer where backspace characters pass through the driver unchanged. The absence of this option indicates a line printer. The driver takes the necessary action to accommodate backspace characters.
`-o`	Resets the printer back to line-printer mode.
`-c`*cols*	Limit the number of columns to be printed to *cols*. Characters beyond the specified limit are truncated.
`-d`	Reset options to default for the device. This action is not taken until the next open occurs on the device.
`-i`*indent*	Indent *indent* columns before printing the first column.
`-k`	Select cooked mode. Cooked mode must be used with a cooked device special file which is identified by an lp mnemonic that is not preceded by the character `r`.
`-l`*lines*	Specify the number of *lines* per page. The last new-line character of each page is changed to a form-feed.
`-n`	Set the page size to infinity. Since the last new-line of the page is never encountered, no new-line characters are changed to form-feeds.
`-r`	Select a raw mode for graphics dumps. All other options are ignored except `-a`. If the `-r` option is not given, `-k` is assumed.
`-C`*pages*	Eject zero or more *pages* after the final close of the device.
`-O`*pages*	Eject zero or more *pages* when the device is opened.

EXAMPLES

In a typical case, the printer is set to 80 columns, no indentation, with no form-feeds between pages:

```
slp -c80 -i0 -n >/dev/lp
```

WARNINGS

Use of the `slp` command in conjunction with the `lp` spooler (see *lp*(1)) might cause undesirable side effects. The spooler model files make assumptions regarding the configuration and can get confused when the default values are altered. Although most options can be altered without difficulty, special problems sometimes result from adjusting the number of lines and the number of columns per page.

DEPENDENCIES

Series 300/400

The value of *cols* is forced into the range of 1 to 227, the value of *indent* from 0 to 227, and the value of *lines* from 1 to **MAXSHORT**.

The uppercase-only flag, the no-overprint flag, the raw-mode flag, and no-page-eject-on-open-or-close flag can be selected (enabled) by appropriate use of the minor number in the `mknod` command (see *mknod*(1M)). See the *HP-UX System Administrator Manual* for details.

If the no-page-eject-on-open-or-close flag is enabled by the `mknod` command, the `-C` and `-O` options are ignored.

AUTHOR

`slp` was developed by HP.

SEE ALSO

lp(1), ioctl(2), lp(7).

(Requires Optional SNA Networking Software)

NAME

sna3179g, sna3270, sna3770, liblu62.a - IBM* 3179G/3192G, 3270, 3777 terminal emulator and APPC API

Remarks:

The following descriptions are provided as a very brief overview of selected SNA (Systems Network Architecture) utilities that are available as optional software for use with the HP-UX operating system. For more information refer to system documentation provided with the software. For purchasing information, contact your nearest HP Sales and Support Office.

DESCRIPTION

sna3179g IBM 3179G color graphics display terminal emulator. **sna3179g** runs in the foreground as an X11 Window System client program. To start **sna3179g**, the SNALink product, **snanuc**, must be running. Refer to the *HP-UX SNA3179G and Gateway / SNA3179G Reference Manual* for more detailed information.

sna3270 Provides the standard subset functionality of an IBM 3278 terminal or IBM 3287 Printer. Connection to the SNA host is provided through **snanuc** which must be running before **sna3270** can be used. These two processes together make the HP 9000 system appear to the rest of the SNA network to be a 3274 Cluster Controller with attached 3278 Display Stations and 3287 Printers. Additionally, sna3270 is capable of handling double-byte characters using IBM's Double-Byte Character Set (DBCS) protocol. Supported DBCS languages include Traditional Chinese, Japanese, and Korean.

sna3770 Emulates the major features of an IBM 3777 Model 1 Communications Terminal. Software emulation of a card reader, card punch, printer, basic exchange data sets (diskette), and console provide the capability for remote job entry and file transfer to an SNA host.

The **sna3770** command starts one or more background monitors which send files queued by *send3770*(1) and receive files sent by the SNA host. It also does any necessary initialization of SNA3770 system files.

liblu62.a Gives application programmers working on an HP 9000 the means to implement Advanced Program-to-Program Communications (APPC) applications distributed between an HP 9000 and an IBM-compatible host. **liblu62.a** is a set of library functions which allow HP 9000 programmers to create APPC applications, commonly called Transaction Programs (TPs), that can exchange data and control information with cooperating TPs on the host.

SEE ALSO

SNA system software documentation provided with optional SNA software.

* IBM is a trademark of International Business Machines Corporation.

NAME

soelim - eliminate .so's from nroff input

SYNOPSIS

`soelim` [file ... *]*

DESCRIPTION

`soelim` reads the specified files or the standard input and performs the textual inclusion implied by `nroff` directives of the form

`.so` *some_file*

when they appear at the beginning of input lines. This is useful when using programs such as *tbl*(1) that do not normally do this, allowing placement of individual tables or other text objects in separate files to be run as a part of a large document.

An argument consisting of a single minus (`-`) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using `'` instead of `.` at the start of the line as in:

```
'so /usr/lib/tmac.s
```

EXAMPLES

`soelim` is often used in a context similar to the following:

```
soelim exum?.n | tbl | nroff -mm | col | lp
```

WARNINGS

The format of the source commands must involve no strangeness — exactly one blank must precede and no blanks follow the file name.

SEE ALSO

more(1), nroff(1), tbl(1).

NAME

softbench - SoftBench Software Development Environment

DESCRIPTION

The **SoftBench** and **Encapsulator** products are designed to improve programmer and team productivity by providing a software development environment that:

- Facilitates rapid, interactive program development and simplifies program maintenance and porting in a distributed computing environment.
- Is easy to learn and use.
- Can be easily customized to leverage a customer's existing environment, processes and tools.

SoftBench

SoftBench is an integrated set of X11/Motif window-based programming tools and a Tool Integration Platform. Together they provide an integrated software development environment targeted at the program construction, test, and maintenance phases of software development.

The SoftBench product is composed of:

- A language-sensitive **Program Editor** and **Program Builder** to support rapid, interactive program construction.
- A **Static Analyzer** and **Program Debugger** for understanding the structure and behavior of complex applications in order to support program test, maintenance, and porting.
- A **Development Manager** used to manage the versions of files the SoftBench tools operate on.

SoftBench programming tools are integrated via services that include:

- A tool communication architecture designed to support a task-oriented environment of cooperative tools.
- Support for a distributed software development environment allowing remote tool execution and remote data access.
- A graphical, OSF/Motif user interface.
- A pervasive, interactive help system.

Encapsulator

Encapsulator delivers the customizability benefit of SoftBench. It allows users to customize and extend the SoftBench environment by:

- Automating custom development processes. Encapsulator is used to define actions to be executed whenever specific events occur in the SoftBench environment.
- Adding the SoftBench graphical user interface to existing HP-UX utilities, customer tools, and third-party tools.

These extensions are made to the environment without modifying the source code of the tools that are encapsulated.

Product Information

SoftBench and Encapsulator run on HP 9000 Series 300/400 and HP 9000 Series 700/800 computers under HP-UX and on other platforms. Contact your HP Sales Representative for ordering information.

AUTHOR

SoftBench was developed by HP.

SEE ALSO

The following references are contained in other manuals shipped with SoftBench software, and are not included in this manual.

softbench (1), encapsulate (1), encaprun (1), softbench (5), softbuild (1), softdebug (1), softdm (1), softedit (1), softeditsrv (1), softmsg (1), softstatic (1).

INTERNATIONAL SUPPORT

SoftBench supports both 8-bit and 16-bit languages.

NAME

sort - sort or merge files

SYNOPSIS

`sort` [`-m`] [`-o` *output*] [`-bdfinruM`] [`-t` *char*] [`-k` *keydef*] [`-y` [*kmem*]] [`-z` *recsz*] [`-T` *dir*] [*file* ...]

`sort` [`-c`] [`-bdfinruM`] [`-t` *char*] [`-k` *keydef*] [`-y` [*kmem*]] [`-z` *recsz*] [`-T` *dir*] [*file* ...]

Obsolete forms:

`sort` [`-mu`] [`-o` *output*] [`-bdfilnrM`] [`-t` *char*] [`-y` [*kmem*]] [`-z` *recsz*] [`-T` *dir*] [*+pos1* [*-pos2*]] [*file* ...]

`sort` [`-c`] [`-u`] [`-bdfilnrM`] [`-t` *char*] [`-y` [*kmem*]] [`-z` *recsz*] [`-T` *dir*] [*+pos1* [*-pos2*]] [*file* ...]

DESCRIPTION

`sort` performs one of the following functions:

1. Sorts lines of all the named files together and writes the result to the specified output.
2. Merges lines of all the named (presorted) files together and writes the result to the specified output.
3. Checks that a single input file is correctly presorted.

The standard input is read if `-` is used as a file name or no input files are specified.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line. Ordering is lexicographic by characters using the collating sequence of the current locale. If the locale is not specified or is set to the `POSIX` locale, then ordering is lexicographic by bytes in machine-collating sequence. If the locale includes multi-byte characters, single-byte characters are machine-collated before multi-byte characters.

Behavior Modification Options

The following options alter the default behavior:

`-c` Check that the single input file is sorted according to the ordering rules. No output is produced; the exit code is set to indicate the result.

`-m` Merge only; the input files are assumed to be already sorted.

`-o` *output* The argument given is the name of an output file to use instead of the standard output. This file can be the same as one of the input files.

`-u` Unique: suppress all but one in each set of lines having equal keys. If used with the `-c` option, check to see that there are no lines with duplicate keys, in addition to checking that the input file is sorted.

`-y` [*kmem*] The amount of main memory used by the sort can have a large impact on its performance. If this option is omitted, `sort` begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, `sort` starts using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, `-y` *0* is guaranteed to start with minimum memory. By convention, `-y` (with no argument) starts with maximum memory.

`-z` *recsz* The size of the longest line read is recorded in the sort phase so that buffers can be allocated during the merge phase. If the sort phase is omitted via the `-c` or `-m` options, a popular system default size will be used. Lines longer than the buffer size will cause `sort` to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

`-T` *dir* Use *dir* as the directory for temporary scratch files rather than the default directory, which is is one of the following, tried in order: the directory as specified in the `TMPDIR` environment variable; `/usr/tmp`, and finally, `/tmp`.

Ordering Rule Options

When ordering options appear before restricted sort key specifications, the ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the ordering options override all global ordering options for that key.

The following options override the default ordering rules:

`-d` Quasi-dictionary order: only alphanumeric characters and blanks (spaces and tabs), as defined by `LC_CTYPE` are significant in comparisons (see *environ*(5)). The `-d` option is ignored for languages with multi-byte characters; all characters are significant.

`-f` Fold letters. Prior to being compared, all lowercase letters are effectively converted into their uppercase equivalents, as defined by `LC_CTYPE`. The `-f` option is ignored for languages with multi-byte characters; all characters are collated unfolded.

`-i` In non-numeric comparisons, ignore all characters which are non-printable, as defined by `LC_CTYPE`. For the ASCII character set, octal character codes 001 through 037 and 0177 are ignored. For languages with multi-byte characters, the `-i` option is ignored.

`-n` The sort key is restricted to an initial numeric string consisting of optional blanks, an optional minus sign, zero or more digits with optional radix character, and optional thousands separators. The radix and thousands separator characters are defined by `LC_NUMERIC`. The field is sorted by arithmetic value. An empty (missing) numeric field is treated as arithmetic zero. Leading zeros and plus or minus signs on zeros do not affect the ordering. The `-n` option implies the `-b` option (see below).

`-r` Reverse the sense of comparisons.

`-l` This option is ignored. Previously it was used to activate sorting using the collation rules associated with the user's `LANG` variable (see *environ*(5)). Language-sensitive collation is now the standard behavior.

`-M` Compare as months. The first several non-blank characters of the field are folded to uppercase and compared with the *langinfo*(3C) items `ABMON_1` < `ABMON_2` < ... < `ABMON_12`. An invalid field is treated as being less than `ABMON_1` string. For example, American month names are compared such that `JAN` < `FEB` < ... < `DEC`. An invalid field is treated as being less than all months. The `-M` option implies the `-b` option (see below).

Field Separator Options

The treatment of field separators can be altered using the options:

`-t` *char* Use *char* as the field separator character; *char* is not considered to be part of a field (although it can be included in a sort key). Each occurrence of *char* is significant (for example, <*char*><*char*> delimits an empty field). If `-t` is not specified, <blank> characters will be used as default field separators; each maximal sequence of <blank> characters that follows a non-<blank> character is a field separator.

`-b` Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the `-b` option is specified before the first `-k` option (+*pos1* argument), it is applied to all `-k` options (+*pos1* arguments). Otherwise, the `-b` option can be attached independently to each `-k` *field_start* or *field_end* option (+*pos1* or (-*pos2* argument; see below). Note that the `-b` option is only effective when restricted sort key specifications are given.

Restricted Sort Key

`-k` *keydef* The *keydef* argument defines a restricted sort key. The format of this definition is

field_start [*type*][**,** *field_end*[*type*]]

which defines a key field beginning at *field_start* and ending at *field_end*. The characters at positions *field_start* and *field_end* are included in the key field, providing that *field_end* does not precede *field_start*. A missing *field_end* means the end of the line. Fields and characters within fields are numbered starting with `1`. Note that this is different than the obsolete form of restricted sort keys, where numbering starts at `0`. See WARNINGS below.

Specifying *field_start* and *field_end* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field.

The arguments *field_start* and *field_end* each have the form *m* **.***n* which are optionally followed by one or more of the *type* options `b`, `d`, `f`, `i`, `n`, `r`, or `M`. These modifiers have the functionality for this key only, that their command-line counterparts have for the entire record.

A *field_start* position specified by *m* **.***n* is interpreted to mean the *n*th character in the *m*th field. A missing *n* means `.1`, indicating the first character of the *m*th field. If the `-b` option is in effect, *n* is counted from the first non-blank character in the *m*th field.

A *field_end* position specified by *m* **.***n* is interpreted to mean the *n*th character in the *m*th field. If *n* is missing, the *m*th field ends at the last character of the field. If the `-b` option is in effect, *n* is counted from the first non-<blank> character in the *m*th field.

Multiple `-k` options are permitted and are significant in command line order. A maximum of 10 `-k` options can be given. If no `-k` option is specified, a default sort key of the entire line is used. When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant. If all the specified keys compare equal, the entire record is used as the final key.

The `-k` option is intended to replace the obsolete [*+pos1* [*+pos2*]] notation, using *field_start* and *field_end* respectively. The fully specified [*+pos1* [*+pos2*]] form:

> *+w* **.***x*-*y* **.***z*

is equivalent to:

> `-k` *w*+1**.***x*+1**,***y* **.**0 (if *z* == 0)
> `-k` *w*+1**.***x*+1**,***y*+1**.***z* (if *z* > 0)

Obsolete Restricted Sort Key

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field.

pos1 and *pos2* each have the form *m* **.***n* optionally followed by one or more of the flags `bdfinrM`. A starting position specified by *+m* **.***n* is interpreted to mean character *n*+1 in field *m*+1. A missing **.***n* means **.**0, indicating the first character of field *m*+1. If the `b` flag is in effect, *n* is counted from the first non-blank in field *m*+1; *+m* **.**`0b` refers to the first non-blank character in field *m*+1.

A last position specified by *-m* **.***n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing **.***n* means **.**0, indicating the last character of the *m*th field. If the `b` flag is in effect, *n* is counted from the last leading blank in field *m*+1; *-m* **.**`1b` refers to the first non-blank in field *m*+1.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the default ordering rules applied to the sort.

`LC_CTYPE` determines the behavior of character classification for the `-d`, `-f`, and `-i` options.

`LC_NUMERIC` determines the definition of the radix and thousands separator characters for the `-n` option.

`LC_TIME` determines the month names for the `-M` option.

`LANG` determines the language in which messages are displayed.

If either `LC_COLLATE`, `LC_CTYPE`, `LC_NUMERIC`, or `LC_TIME` is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of `POSIX` (see *lang*(5)) is used. If any of the internationalization variable contains an invalid setting, `sort` behaves as if all internationalization variables were set to `POSIX`. See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Sort the contents of `infile` with the second field as the sort key:

```
sort -k 2,2 infile
```

Sort, in reverse order, the contents of `infile1` and `infile2`, placing the output in `outfile` and using the first two characters of the second field as the sort key:

```
sort -r -o outfile -k 2.1,2.2 infile1 infile2
```

Sort, in reverse order, the contents of `infile1` and `infile2`, using the first non-blank character of the fourth field as the sort key:

```
sort -r -k 4.1b,4.1b infile1 infile2
```

Print the password file (`/etc/passwd`) sorted by numeric user ID (the third colon-separated field):

```
sort -t: -k 3n,3 /etc/passwd
```

Print the lines of the presorted file `infile`, suppressing all but the first occurrence of lines having the same third field:

```
sort -mu -k 3,3 infile
```

DIAGNOSTICS

`sort` exits with one of the following values:

- **0** All input files were output successfully, or `-c` was specified and the input file was correctly presorted.
- **1** Under the `-c` option, the file was not ordered as specified, or if the `-c` and `-u` options were both specified, two input lines were found with equal keys. This exit status is not returned if the `-c` option is not used.
- **>1** An error occurred such as when one or more input lines are too long.

When the last line of an input file is missing a new-line character, `sort` appends one, prints a warning message, and continues.

If an error occurs when accessing the tables that contain the collation rules for the specified language, `sort` prints a warning message and defaults to the `POSIX` locale.

If a `-d`, `-f`, or `-i` option is specified for a language with multi-byte characters, `sort` prints a warning message and ignores the option.

WARNINGS

Numbering of fields and characters within fields (`-k` option) has changed to conform to the POSIX standard. Beginning at HP-UX Release 9.0, the `-k` option numbers fields and characters within fields, starting with `1`. Prior to HP-UX Release 9.0, numbering started at `0`.

A field separator specified by the `-t` option is recognized only if it is a single-byte character.

The character type classification categories `alpha`, `digit`, `space`, and `print` are not defined for multi-byte characters. For languages with multi-byte characters, all characters are significant in comparisons.

FILES

```
/usr/tmp/stm???
/tmp/stm???
```

SEE ALSO

comm(1), join(1), uniq(1), collate8(4), environ(5), hpnls(5), lang(5).

STANDARDS CONFORMANCE

`sort`: SVID2, XPG2, XPG3, POSIX.2

NAME

spell, hashmake, spellin, hashcheck - find spelling errors

SYNOPSIS

`spell` [`-v`] [`-b`] [`-x`] [`-l`] [`-i`] [*+local_file*] [*files*]

`/usr/lib/spell/hashmake`

`/usr/lib/spell/spellin` *n*

`/usr/lib/spell/hashcheck` *spelling_list*

DESCRIPTION

`spell` collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

`spell` ignores most `troff`, `tbl`, and *eqn* constructions.

Options

`spell` recognizes the following options:

`-v`	All words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.
`-b`	British spelling is checked. Besides preferring `centre`, `colour`, `programme`, `speciality`, `travelled`, etc., this option insists upon `-ise` in words such as `standardise`.
`-x`	Every plausible stem is printed with = for each word.

By default, `spell` follows chains of included files much like `deroff` (see *deroff*(1)) which recognizes the `troff/nroff` intrinsics `.so` and `.nx` *unless* the names of such included files begin with `/usr/lib`. If the `-l` option is used, `spell` follows the chains of *all* included files. With the `-i` option, `spell` ignores all chains of included files.

If the *+local_file* option is used, words found in *local_file* are removed from `spell`'s output. *local_file* is the name of a user-provided file containing a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files can be specified by name arguments, indicated below with their default settings (see FILES and VARIABLES). Copies of all output are accumulated in the history file. The stop list filters out misspellings (such as `thier=thy-y+ier`) that would otherwise pass.

Three routines help maintain and check the hash lists used by `spell`:

`hashmake`	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
`spellin` *n*	Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error.
`hashcheck`	Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

EXAMPLES

To check spelling of a single *word*:

```
echo word | spell
```

If *word* is spelled correctly, a prompt is returned. If it is spelled incorrectly, *word* is printed before the prompt is returned. To check spelling of multiple words, they can also be typed as a group on the same command line:

```
    echo worda wordb wordc ... | spell
```

To create a personal spelling list that incorporates the words already present in the default American spelling list file `/usr/lib/spell/hlista`:

```
    cat /usr/lib/spell/hlista | /usr/lib/spell/hashcheck >tmp1
    /usr/lib/spell/hashmake <addwds >>tmp1
    sort -u -o tmp1 tmp1
    /usr/lib/spell/spellin 'wc -l <tmp1' <tmp1 >hlista
```

To modify the default British spelling list file `/usr/lib/spell/hlistb`, replace all occurrences of `hlista` with `hlistb` in the above example.

To add words to the default spelling list, change login to `root`, change the current working directory to `/usr/lib/spell` and execute the commands listed in the above example.

WARNINGS

The spelling list's coverage is uneven. When undertaking the use of `spell` as a new tool, it may be advisable to monitor the output for several months to gather local additions. Typically, these are kept in a separate local file that is added to the hashed *spelling_list* via `spellin`, as shown above.

The British spelling feature was developed by an American.

Start-up versions of files `hlista`, `hlistb`, and `hstop` are available in directory `/etc/newconfig`. If these files or a suitable equivalent are not present in directory `/usr/lib/spell`, `spell` complains:

```
    spell: cannot initialize hash table
    spell: cannot initialize hash table
```

FILES

`/usr/lib/spell/hlist`[ab]	hashed spelling lists, American & British
`/usr/lib/spell/hstop`	hashed stop list
`/usr/lib/spell/spellhist`	history file
`/usr/lib/spell/spellprog`	executable program file

VARIABLES

D_SPELL	Your hashed spelling list (default is D_SPELL=`/usr/lib/spell/hlist`[ab])
H_SPELL	Spelling history (default is H_SPELL=`/usr/lib/spell/spellhist`).
S_SPELL	Your hashed stop list (default is S_SPELL=`/usr/lib/spell/hstop`).
TMPDIR	Directory for temporary files; overrides the default `/tmp`.

SEE ALSO

deroff(1), sed(1), sort(1), tbl(1), tee(1).

STANDARDS CONFORMANCE

spell: SVID2, XPG2, XPG3

NAME

split - split a file into pieces

SYNOPSIS

`split` [`-l` *line_count*] [`-a` *suffix_length*] [*file* [*name*]]

`split` [`-b` *n*[`k`|`m`]] [`-a` *suffix_length*] [*file* [*name*]]

Obsolescent:
`split` [-*n*] [*file* [*name*]]

DESCRIPTION

`split` reads *file* and writes it in pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with `aa` appended, and so on lexicographically, up to `zz` (only ASCII letters are used, a maximum of 676 files). If no output *name* is given, `x` is the default.

If no input *file* is given, or if `-` is given instead, the standard input file is used.

OPTIONS

`split` recognizes the following command-line options and arguments:

`-l` *line_count*	The input file is split into pieces *line_count* lines in size.
`-a` *suffix_length*	*suffix_length* letters are used to form the suffix of the output filenames. This option allows creation of more than 676 output files. The output file names created cannot exceed the maximum file name length allowed in the directory containing the files.
`-b` *n*	The input file is split into pieces *n* bytes in size.
`-b` *n*`k`	The input file is split into pieces $n \times 1024$ bytes in size. No space separates the *n* from the `k`.
`-b` *n*`m`	The input file is split into pieces $n \times 1\,048\,576$ bytes in size. No space separates the *n* from the `m`.
-*n*	The input file is split into pieces *n* lines in size. This option is obsolescent and is equivalent to using the `-l` *line_count* option.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

SEE ALSO

bfs(1), csplit(1).

STANDARDS CONFORMANCE

`split`: SVID2, XPG2, XPG3

(Requires Optional ALLBASE/SQL Software)

NAME

sqlgen - generate command files to unload, reload an ALLBASE/SQL relational DataBase Environment

SYNOPSIS

`sqlgen` [`-i`] [`-e` *editorname*]

REMARKS

The ALLBASE/SQL product must be installed on the system before *sqlgen* can be used.

DESCRIPTION

`sqlgen` invokes the ALLBASE/SQL utility program for generating command files to unload and reload an ALLBASE/SQL relational database environment (DBEnvironment). `sqlgen` can only be executed by ALLBASE/SQL users with DBA authority over the DBEnvironment.

Options

`sqlgen` recognizes the following command-line options and arguments:

`-e` *editorname*	Set the current editor for use with *sqlgen*.
`-i`	Cause *sqlgen* to write its input to the standard output.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

AUTHOR

`sqlgen` was developed by HP.

FILES

`/usr/lib/sqldaemon`	cleanup daemon program file.
`/usr/lib/hpsqlproc`	ALLBASE/SQL program file.
`/usr/bin/sqlgen`	SQLGEN program file.
`/usr/lib/hpsqlcat`	ALLBASE/SQL message catalog file.
`/usr/lib/nls/$LANG/hpsqlcat`	Localized ALLBASE/SQL message catalog file.

SEE ALSO

ALLBASE/SQL Database Administration Guide.

(Requires Optional ALLBASE/SQL Software)

NAME

sqlmig - migrate an ALLBASE/SQL DBEnvironment to a new release

SYNOPSIS

`sqlmig`

REMARKS

The ALLBASE/SQL product must be installed on the system before `sqlmig` can be used.

DESCRIPTION

`sqlmig` invokes the ALLBASE/SQL migration program to migrate an ALLBASE/SQL relational database environment (DBEnvironment) from one release to another. No options are available with this command. `sqlmig` can be executed by all system users; however, only the creator of the DBEnvironment can migrate the DBEnvironment to a new release.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported. `MIGLOGSZ` determines the size of the log that is used (default 400).

AUTHOR

`sqlmig` was developed by HP.

FILES

`/usr/lib/sqldaemon`	cleanup daemon program file.
`/usr/bin/sqlmig`	executable `sqlmig` program file.
`/usr/lib/nls/$LANG/hpsqlcat`	localized ALLBASE/SQL message catalog file.

SEE ALSO

ALLBASE/SQL Database Administration Guide.
ALLBASE/SQL Reference Manual.

(Requires Optional ALLBASE/SQL Software)

NAME

sqlutil - maintain and configure an ALLBASE/SQL DBEnvironment

SYNOPSIS

`sqlutil`

REMARKS

The ALLBASE/SQL product must be installed on the system before `sqlutil` can be used.

DESCRIPTION

`sqlutil` invokes the SQL utility program to maintain and reconfigure an ALLBASE/SQL relational DataBase Environment (DBEnvironment). No options are available with this command. `sqlutil` can be executed on all DataBase Environment Configuration (DBECon) files by all system users; however, only the creator (or a user with designated permission) of the DBEnvironment configuration file can modify it.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

AUTHOR

`sqlutil` was developed by Hewlett-Packard.

FILES

`/usr/lib/sqldaemon`	cleanup daemon program file.
`/usr/lib/hpsqlproc`	ALLBASE/SQL program file.
`/usr/bin/sqlutil`	SQL utility program file.
`/usr/lib/hpsqlcat`	ALLBASE/SQL message catalog file.
`/usr/lib/nls/$LANG/hpsqlcat`	Localized SQL message catalog file.

SEE ALSO

ALLBASE / SQL Database Administration Guide.
ALLBASE / SQL Reference Manual.

NAME

ssp - remove multiple line-feeds from output

SYNOPSIS

ssp

DESCRIPTION

ssp (single-space) removes redundant blank lines from the standard input and sends the result to the standard output. All blank lines at the beginning of a file are removed, and all multiple blank lines elsewhere in the file (including end-of-file) are reduced to a single blank line.

ssp is typically used in pipelines such as

```
    nroff -ms file1 | ssp
```

ssp is equivalent to the 4.2BSD `cat -s` command.

To remove all blank lines from a file except at beginning of file, use `rmnl` (see *rmnl*(1)). To remove all blank lines from a file including beginning of file, use `rmnl` piped to **ssp**, or **ssp** piped to `rmnl`.

SEE ALSO

cat(1), rmnl(1).

NAME

strings - find the printable strings in an object or other binary file

SYNOPSIS

`strings` [`-a`] [`-t` *format*] [`-n` *number*] [*file*] ...

Obsolescent:
`strings` [`-a`] [`-o`] [`-`*number*] [*file*] ...

DESCRIPTION

`strings` looks for ASCII strings in a file. If no *file* is specified, standard input is used. A string is any sequence of four or more printing characters ending with a new-line or null character.

`strings` is useful for identifying random object files and many other things.

Options

`strings` recognizes the following options:

`-a` By default, *strings* looks only in the initialized data space of object files (as recognized by their magic numbers). If this flag is used, the entire file is inspected. This flag is always set if standard input is being read or the file is not recognized as an object file. For backward compatibility, `-` is understood as a synonym for `-a`.

`-t` *format* Write each string preceded by its byte offset from the start of the file. The format is dependent on the single character used as the *format* option-argument:

d The offset is written in decimal.

o The offset is written in octal.

x The offset is written in hexadecimal.

`-n` *number*
Specify *number* as the minimum string length, rather than the default 4.

`-o`
Each string is preceded by its offset in the file (in octal). This option is obsolescent and is equivalent to specifing the `-t` *o* option.

`-`*number*
Specify *number* as the minimum string length, rather than the default 4. This option is obsolescent and is equivalent to using the `-n` *number* option. RE

AUTHOR

`strings` was developed by the University of California, Berkeley.

WARNINGS

The algorithm for identifying strings is extremely primitive.

SEE ALSO

od(1).

NAME

strip - strip symbol and line number information from an object file

SYNOPSIS

`strip` [`-l`] [`-x`] [`-r`] [`-V`] *filename* ...

DESCRIPTION

`strip` removes the symbol table and line number information from object files, including archives. Thereafter, no symbolic debugging access is available for that file; thus, this command is normally run only on production modules that have been debugged and tested. The effect is identical to using the `-s` option of *ld*.

Options

The amount of information stripped from the symbol table can be controlled by using any of the following options:

`-l`	Strip line number information only; do not strip any symbol table information.
`-x`	Do not strip static or external symbol information.
`-r`	Reset the relocation indexes into the symbol table.
`-V`	Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, `strip` complains and terminates without stripping *filename* unless the `-r` option is used.

If `strip` is executed on an archive file (see *ar*(4)), the archive symbol table is removed. The archive symbol table must be restored by executing `ar` with its `s` operator (see *ar*(1)) before the archive can be link-edited by `ld` command (se *ld*(1)). `strip` instructs the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce file storage overhead consumed by the object file.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`strip:` *name*`: cannot open`
name cannot be read.

`strip:` *name*`: bad magic`
name is not an appropriate object file.

`strip:` *name*`: relocation entries present; cannot strip`
name contains relocation entries and the `-r` option was not specified. Symbol table information cannot be stripped.

EXAMPLES

Strip symbol table and debug information from the shared library `libfoo.sl` in the current directory to reduce it's size. Symbol information required to use the library is preserved:

```
strip ./libfoo.sl
```

DEPENDENCIES

Series 300/400

If *name* is a relocatable file, `strip` removes the local symbols from it. If *name* is an archive file, `strip` removes the local symbols from any `a.out`-format files it finds in the archive. Certain libraries (such as those residing in `/lib`) have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

The `-l`, `-x`, `-r`, and `-V` options are not supported.

`strip` removes debug information from any `a.out` file it finds while stripping an archive file (see *ar*(1)).

`strip` does not warn the user that stripping an archive file removes the archive symbol table.

Currently, `strip` does not complain about stripping relocatable files. It strips such files by removing debugging information only.

Series 700/800

The `-l` and `-x` options are synonymous because the symbol table contains only static and external symbols. Either option strips only symbolic debugging information and unloadable data. The `-r` option allows strip to be run on relocatable files, in which case the effect is also to strip only symbolic debugging information and unloadable data.

FILES

Series 300/400

`/tmp/s*`	temporary files

Series 700/800

`/usr/tmp/strp??????`	temporary files

SEE ALSO

ar(1), as(1), cc(1), ld(1), a.out(4), ar(4).

STANDARDS CONFORMANCE

`strip`: SVID2, XPG2, XPG3, POSIX.2

NAME

stty - set the options for a terminal port

SYNOPSIS

`stty [-a | -g | ` *options* `]`

DESCRIPTION

`stty` sets or reports current settings of certain terminal I/O options for the device that is the current standard input. The command takes four forms:

`stty`	Report the settings of a system-defined set of options;
`stty -a`	Report all of current option settings;
`stty -g`	Report current settings in a form that can be used as an argument to another `stty` command.
`stty` *options*	Set terminal I/O options as defined by *options*.

For detailed information about the modes listed below from Control Modes through Local Modes as they relate to asynchronous lines, see *termio*(7). For detailed information about the modes listed under Hardware Flow Control Modes below, see *termiox*(7). Options in the Combination Modes group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. *options* are selected from the following:

Control Modes

`rows` *number*	Set the terminal window row size equal to *number*.
`columns` *number*	Set the terminal window column size (width) equal to *number*. `cols` can be used as an abbreviation for `columns`.
`parenb (-parenb)`	Enable (disable) parity generation and detection.
`parodd (-parodd)`	Select odd (even) parity.
`cs5 cs6 cs7 cs8`	Select character size (see *termio*(7)).
`0`	Hang up phone line immediately.
`50 75 110 134.5 150 200 300 600 900 1200 1800 2400 3600 4800 7200 9600 19200 38400 57600 115200 240400 460800 exta extb`	Set terminal baud rate to the number given, if possible (some hardware interfaces do not support all of the speeds listed here). Speeds above 38400 are supported on Series 300/400 and 700 only.
`ispeed` *number*	Set terminal input baud rate to *number*. If *number* is zero, the input baud rate is set to the value of the output baud rate.
`ospeed` *number*	Set terminal output baud rate to *number*. If *number* is zero, the modem control lines are released, which in turn disconnects the line.
`hupcl (-hupcl)`	Hang up (do not hang up) modem connection on last close.
`hup (-hup)`	Same as `hupcl (-hupcl)`.
`cstopb (-cstopb)`	Use two (one) stop bits per character.
`cread (-cread)`	Enable (disable) the receiver.
`crts (-crts)`	Enable (disable) request-to-send.
`clocal (-clocal)`	Assume a line without (with) modem control.
`loblk (-loblk)`	Block (do not block) output from a non-current layer.

Input Modes

`ignbrk (-ignbrk)`	Ignore (do not ignore) break on input.
`ienqak (-ienqak)`	Enable (disable) ENQ-ACK Handshaking.
`brkint (-brkint)`	Signal (do not signal) INTR on break.

`ignpar (-ignpar)`	Ignore (do not ignore) parity errors.
`parmrk (-parmrk)`	Mark (do not mark) parity errors (see *termio*(7)).
`inpck (-inpck)`	Enable (disable) input parity checking.
`istrip (-istrip)`	Strip (do not strip) input characters to seven bits.
`inlcr (-inlcr)`	Map (do not map) new-line character to carriage return (CR) on input.
`igncr (-igncr)`	Ignore (do not ignore) CR on input.
`icrnl (-icrnl)`	Map (do not map) CR to a new-line character on input.
`iuclc (-iuclc)`	Map (do not map) uppercase alphabetic characters to lowercase on input.
`ixon (-ixon)`	Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
`ixany (-ixany)`	Allow any character (only DC1) to restart output.
`ixoff (-ixoff)`	Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

`opost (-opost)`	Post-process output (do not post-process output; ignore all other output modes).
`olcuc (-olcuc)`	Map (do not map) lowercase alphabetics to uppercase on output.
`onlcr (-onlcr)`	Map (do not map) new-line character to a carriage-return/new-line character sequence on output.
`ocrnl (-ocrnl)`	Map (do not map) CR to new-line character on output.
`onocr (-onocr)`	Do not (do) output CRs at column zero.
`onlret (-onlret)`	On the terminal, a new-line character performs (does not perform) the CR function.
`ofill (-ofill)`	Use fill characters (use timing) for delays.
`ofdel (-ofdel)`	Fill characters are DELs (NULs).
`cr0 cr1 cr2 cr3`	Select style of delay for carriage returns (see *termio*(7)).
`nl0 nl1`	Select style of delay for new-line characters (see *termio*(7)).
`tab0 tab1 tab2 tab3`	Select style of delay for horizontal tabs (see *termio*(7).
`bs0 bs1`	Select style of delay for backspaces (see *termio*(7)).
`ff0 ff1`	Select style of delay for form-feeds (see *termio*(7)).
`vt0 vt1`	Select style of delay for vertical tabs (see *termio*(7)).

Local Modes

`isig (-isig)`	Enable (disable) the checking of characters against the special control characters INTR and QUIT.
`icanon (-icanon)`	Enable (disable) canonical input (ERASE and KILL processing).
`iexten (-iexten)`	Enable (disable) any implementation-defined special control characters not currently controlled by *icanon*, *isig*, or *ixon*.
`xcase (-xcase)`	Canonical (unprocessed) uppercase and lowercase presentation.
`echo (-echo)`	Echo back (do not echo back) every character typed.
`echoe (-echoe)`	Echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode erases the ERASEed character on many CRT terminals. However, it does *not* keep track of column position and, as a result, may not correctly erase escaped characters, tabs, and backspaces.

`echok` (`-echok`)	Echo (do not echo) a new-line character after a KILL character.
`lfkc` (`-lfkc`)	(obsolete) Same as `echok` (`-echok`).
`echonl` (`-echonl`)	Echo (do not echo) new-line character.
`noflsh` (`-noflsh`)	Disable (enable) flush after INTR or QUIT.
`tostop` (`-tostop`)	Enable (disable) generation of SIGTTOU signals when background jobs attempt output.

Hardware Flow Control Modes

The following options are reserved for use with those devices that support hardware flow control through the termiox interface. If the functionality is supported, this interface must be used.

`rtsxoff` (`-rtsxoff`)	enable (disable) RTS hardware flow control on input (see *termiox*(7))
`ctsxon` (`-ctsxon`)	enable (disable) CTS hardware flow control on output (see *termiox*(7))

Control Assignments

control-character c	Set *control-character* to *c*, where *control-character* is `erase`, `kill`, `intr`, `quit`, `eof`, `eol`, `min`,or `time` (`min` and `time` are used with `-icanon`; see *termio*(7)). For systems that support job control, `susp` and `dsusp` characters can also be set. For systems that support shell layers (see *shl*(1)) `swtch` can also be set. If *c* is preceded by an (escaped from the shell) circumflex (^), the value used is the corresponding control character (for example, `^d` represents **Ctrl-d**); ^? is interpreted as DEL and ^- is interpreted as undefined.
`line` *i*	Set line discipline to *i* where the value of *i* ranges from zero through 127 decimal (See *termio*(7)).

Combination Modes

`evenp` or `parity`	Enable `parenb` and `cs7`.
`oddp`	Enable `parenb`, `cs7`, and `parodd`.
`-parity`, `-evenp`, or `-oddp`	Disable `parenb` and set `cs8`.
`raw` (`-raw` or `cooked`)	Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing). See WARNINGS.
`nl` (`-nl`)	Unset (set) `icrnl` and `onlcr` . In addition `-nl` unsets `inlcr`, `igncr`, `ocrnl`, and `onlret`.
`lcase` (`-lcase`)	Set (unset) `xcase`, `iuclc`, and `olcuc`.
`LCASE` (`-LCASE`)	Same as `lcase` (`-lcase`).
`tabs` (`-tabs` or `tab3`)	Preserve (expand to spaces) tabs when printing.
`ek`	Reset ERASE and KILL characters back to default # and @.
`sane`	Reset all modes to some reasonable values.
`term`	Set all modes suitable for the terminal type *term*, where *term* is one of `tty33`, `tty37`, `vt05`, `tn300`, `ti700`, `hp`, or `tek`.

Reporting Functions

`size`	Print terminal window size to standard output in a rows-and-columns format.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the valid control characters for printing.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, `stty` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

The command:

```
stty kill '^X' intr '^C'
```

sets the delete-line character to `^X` (Ctrl-X) and the interrupt character to `^C`. This command is usually found in the `.login` or `.profile` file so that `^X` and `^C` need not be set by the user at each login session.

WARNINGS

Use of `raw` mode produces certain side effects which have varied from release to release in the past and may vary in the future. Relying on these side effects in applications can lead to unreliable results in the future and is therefore discouraged.

DEPENDENCIES

Refer to the DEPENDENCIES section of *termio*(7) for a further description of capabilities that are not supported.

SEE ALSO

shl(1), tabs(1), ioctl(2), termio(7).

STANDARDS CONFORMANCE

`stty`: SVID2, XPG2, XPG3, POSIX.2

NAME

su - become super-user or another user

SYNOPSIS

`su` [`-`][*name* [*arg* ...]]

DESCRIPTION

`su` allows one to become another user without logging out. The default user *name* is `root` (i.e., super-user).

To use `su`, the appropriate password must be supplied (unless you are already *root*). If the password is correct, `su` executes a new shell with the real and effective user ID, real and effective group ID, and group access list set to that of the specified user. The previously defined **HOME** and **ENV** environment variables are removed unless the `-` option is specified. The new shell is the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or `/bin/sh` if none is specified (see *sh*(1)). To restore normal user ID privileges, type an EOF to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell, permitting the super-user to run shell procedures with restricted privileges. When using programs such as the Bourne shell (see *sh-bourne*(1)), an *arg* of the form `-c` *string* executes *string* via the shell and an arg of `-r` gives the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is similar in behavior to the Bourne shell. If the first argument to `su` is a `-`, the environment is changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is `-`, thus causing first the system's profile (`/etc/profile`) and then the specified user's profile (`.profile` in the new **HOME** directory) to be executed. Otherwise, the environment is passed along unchanged, except that **$PATH**, is unconditionally set to `/bin:/etc:/usr/bin` for `root`. Note that if the optional program used as the shell is `/bin/sh`, the user's `.profile` can check *arg0* for `-sh` or `-su` to determine if it was invoked by the `login` or `su` command (see *login*(1) or *su*(1), respectively). If the user's program is other than `/bin/sh`, then `.profile` is invoked with an *arg0* of *-program* by both `login` and `su`.

The `-` option always resets **$PATH** to `/bin:/etc:/usr/bin` for the super-user, and `/bin:/etc:/usr/bin` for all others. However, the files `/etc/profile` and `.profile` are normally executed anyway, thus restoring the intended value of **$PATH**.

All attempts to become another user are logged in `/usr/adm/sulog`, including failures. Successful attempts are flagged with `+`, failures with `-`.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**.

If any internationalization variable contains an invalid setting, `su` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Characters in the 7-bit USASCII code sets are supported in login names (see *ascii*(5)).

EXAMPLES

Become user `bin` while retaining the previously exported environment (**$HOME** and **$ENV** are no longer defined):

```
su bin
```

Become user `bin` but change the environment to what would be expected if `bin` had originally logged in:

```
su - bin
```

Execute *command* with the temporary environment and permissions of user `bin`:

```
su - bin -c command args
```

WARNINGS

After a successful password has been supplied (if needed), `su` uses information from `/etc/passwd` and

`/etc/logingroup` to determine the user's group id and group access list. If `/etc/group` is linked to `/etc/logingroup`, and group membership for the user trying to log in is managed by the Network Information Service (NIS), and no NIS server is able to respond, **su** waits until a server does respond.

FILES

`$HOME/.profile`	user's profile
`/etc/logingroup`	system's default group access list file
`/etc/passwd`	system's password file
`/etc/profile`	system's profile
`/usr/adm/sulog`	log of all attempts

VARIABLES

HOME	User's home directory
LOGNAME	User's login name
PATH	Command name search path
PS1	Default prompt
SHELL	Name of the user's shell

SEE ALSO

env(1), login(1), sh(1), initgroups(3C), group(4), passwd(4), profile(4), environ(5).

STANDARDS CONFORMANCE

`su`: SVID2, XPG2

NAME

sum - print checksum and block or byte count of file(s)

SYNOPSIS

`sum [-r] [-p] [`*file* `...]`

Remarks

`sum` is obsolescent and should not be used in new applications that are intended to be portable between systems. Use `cksum` instead (see *cksum*(1)).

DESCRIPTION

`sum` calculates and prints to standard output a checksum for each named file, and also prints the size of the file in 512 byte blocks, rounded up.

The default algorithm is a 16-bit sum of the bytes in which overflow is ignored. Alternate algorithms can be selected with the `-r` and `-p` options.

Standard input is used if no file names are given.

`sum` is typically used to verify data integrity when copying files between systems.

Options

`sum` recognizes the following options:

`-r`	Use an alternate algorithm in which the 16-bit sum is right rotated with each byte in computing the checksum.
`-p`	Use the 32-bit cyclical redundancy check (CRC) algorithm used by `cksum`.

RETURN VALUE

`sum` returns the following values upon completion:

0	All files were processed succesfully.
>0	One or more files could not be read or some other error occurred.

If an inaccessible file is encountered, `sum` continues processing any remaining files, but the final exit status is affected.

DIAGNOSTICS

Read error conditions are indistinguishable from end of file on most devices; check the block or byte count.

SEE ALSO

cksum(1), wc(1), pdf(4).

STANDARDS CONFORMANCE

`sum`: SVID2, XPG2, XPG3

NAME

tabs - set tabs on a terminal

SYNOPSIS

`tabs` [*tabspec*] [+m *n*] [-T *type*]

DESCRIPTION

`tabs` sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

If you are using a non-HP terminal, you should keep in mind that behavior will vary for some tab settings.

Four types of tab specification are accepted for *tabspec*: "canned", repetitive, arbitrary, and file. If no `tabspec` is given, the default value is `-8`; i.e., HP-UX "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the left-most column on a terminal, even one whose column markers begin at 0.

-code Gives the name of one of a set of "canned" tabs. Recognized *code*s and their meanings are as follows:

`-a` 1,10,16,36,72
Assembler, IBM S/370, first format

`-a2` 1,10,16,40,72
Assembler, IBM S/370, second format

`-c` 1,8,12,16,20,55
COBOL, normal format

`-c2` 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

```
<:t-c2 m6 s66 d:>
```

`-c3`
1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than `-c2`. This is the recommended format for COBOL. The appropriate format specification is:

```
<:t-c3 m6 s66 d:>
```

`-f`
1,7,11,15,19,23
FORTRAN

`-p`
1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

`-s`
1,10,55
SNOBOL

`-u`
1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

-n A repetitive specification requests tabs at columns 1+*n*, 1+2×*n*, etc. Of particular importance is the value `-8`: this represents the HP-UX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the `nroff -h` option for high-speed output (see *nroff*(1)). Another special case is the value `-0`, implying no tabs at all.

n1,*n2*,... The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

--*file* If the name of a file is given, `tabs` reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as `-8`. This type of specification can be used to ensure that a tabbed file is printed with correct tab settings, and is suitable for use with the `pr` command (see *pr*(1)):

```
tabs -- file; pr file
```

Any of the following can be used also; if a given option occurs more than once, the last value given takes effect:

-T*type* `tabs` usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term*(5). If no **-T** option is supplied, `tabs` searches for the **$TERM** value in the *environment* (see *environ*(5)). If no *type* can be found, `tabs` tries a sequence that works for many terminals.

+m*n* The margin argument can be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. The normal (left-most) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** option is given explicitly.

Tab and margin setting is performed via the standard output.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text within file as single and/or multi-byte characters.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `tabs` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`illegal tabs`
Arbitrary tabs are ordered incorrectly.

`illegal increment`
A zero or missing increment found in an arbitrary specification.

`unknown tab code`
A "canned" code cannot be found.

`can't open`
--*file* option was used and file cannot be opened.

`file indirection`
--*file* option was used and the specification in that file points to yet another file. Indirection of this form is not permitted.

WARNINGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

`tabs` clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

SEE ALSO

nroff(1), pr(1), tset(1), environ(5), term(5).

STANDARDS CONFORMANCE

`tabs`: SVID2, XPG2, XPG3

NAME

tail - deliver the last part of a file

SYNOPSIS

`tail` [`-f`] [`-b` *number*] [*file*]
`tail` [`-f`] [`-c` *number*] [*file*]
`tail` [`-f`] [`-n` *number*] [*file*]

Obsolescent:

`tail` [±[*number*][`l`|`b`|`c`] [`-f`] [*file*]

DESCRIPTION

`tail` copies the named *file* to the standard output beginning at a designated place. If no *file* is named, standard input is used.

Command Forms

`tail` can be used in three forms as indicated above:

`tail -b`*number*...	Copy file starting at *number* blocks from end or beginning of file.
`tail -c`*number*...	Copy file starting at *number* characters from end or beginning of file.
`tail -n`*number*... or `tail` *number*...	Copy file starting at *number* lines from end or beginning of file.

`tail` with no options specified is equivalent to `tail -n10` ....

Options and Command-Line Arguments

`tail` recognizes the following options and command-line arguments:

`-f`	Follow option. If the input file is a regular file or if *file* specifies a FIFO, do not terminate after the last line of the input file has been copied, but read and copy further bytes from the input file when they become available (`tail` enters an endless loop wherein it sleeps for one second then attempts to read and copy further records from the input file). This is useful when monitoring text being written to a file by another process. If no *file* argument is specified and the input is a pipe (FIFO), the `-f` option is ignored.
number	Decimal integer indicating quantity of output to be copied, measured in units specified by accompanying option. If *number* is preceded by a `+` character, copy operation starts *number* units from beginning of file. If *number* is preceded by a `-` character or the option name, copy operation starts *number* units from end of file. If *number* is not preceded by a `b`, `c`, or `n` option, `-n` is assumed. If both the option and *number* are not specified, `-n 10` is assumed.
`-b`*number*	Copy file beginning *number* 512-byte blocks from end or beginning of file. If *number* is not specified, `-b10` is assumed. See *number* description above.
`-c`*number*	Copy file beginning *number* characters from end or beginning of file. If *number* is not specified, `-c10` is assumed. See *number* description above.
`-n`*number*	Copy file beginning *number* lines from end or beginning of file. If *number* is not specified, `-n10` is assumed. See *number* description above.
file	Name of file to be copied. If not specified, the standard input is used.

Obsolescent Form

In the obsolescent form, option letters can be concatenated after the *number* argument to select blocks, characters, or lines. If this syntax is used, ±*number* must be the first argument given. If *number* is not specified, −10 is assumed. This version is provided for backward compatibility only. The forms discussed previously are recommended for portability.

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Print the last three lines in file `file1` to the standard output, and leave `tail` in "follow" mode:

```
tail -fn 3 file1
     or
tail -3 -f file1
```

Print the last 15 characters of file `logfile` followed by any lines that are appended to `logfile` after `tail` is initiated until it is killed:

```
tail -fc15 logfile
     or
tail -f -c 15 logfile
```

Three ways to print an entire file:

```
tail -b +1 file
tail -c +1 file
tail -n +1 file
```

SEE ALSO

dd(1), head(1).

WARNINGS

Tails relative to end-of-file are stored in a 20-Kbyte buffer, and thus are limited in length. Therefore, be wary of the results when piping output from other commands into `tail`.

Various kinds of anomalous behavior may occur with character special files.

STANDARDS CONFORMANCE

`tail`: SVID2, XPG2, XPG3, POSIX.2

NAME

tar - tape file archiver

SYNOPSIS

`tar` *key* [*arg* ...] [*file* | `-C` *directory*] ...

DESCRIPTION

`tar` saves and restores archives of files on a magnetic tape, a flexible disk, or a regular file. Its actions are controlled by the *key* argument. *key* is a string of characters containing an optional version letter, exactly one function letter, and possibly one or more function modifiers. The *key* string can be preceded by a hyphen (`-`) (as when specifying options in other HP-UX commands), but it is not necessary.

Next, an additional *arg* argument is required for each of the function modifiers `b` or `f` used (see below). If both `b` and `f` modifiers are specified, the order of the associated *arg* arguments must match the order of the modifiers.

Each *file* argument specifies a file being saved or restored. If *file* is a directory name, it refers to the files and (recursively) subdirectories contained in that directory.

The version portion of the *key* determines in which format `tar` writes the archive. `tar` can read either format regardless of the version. The version is specified by one of the following letters:

- **N** Write a new (POSIX) format archive. The new format allows file names up to 256 characters in length, and correctly archives and restores the following file types: regular files, character and block special devices, links, symbolic links, directories, and FIFO special files. The new version also stores the user and group name of each file and attempts to use these names to determine the *uid* and *gid* of a file when restoring with the `p` function modifier. This is the new default format.
- **O** Write an old (pre-POSIX) format archive.

The function portion of the *key* is specified by one of the following letters:

- `c` Create a new archive; write from the beginning of the archive instead of starting after the last *file*. Any previous information in the archive is overwritten.
- `r` Add the named *file* to the end of the archive.
- `t` List the names of all the files on the archive. Adding the `v` function modifier expands this listing to include the file modes and owner numbers. The names of all files are listed each time they occur on the tape.
- `u` Add any named *file* to the archive if it is not already present or has been modified since last written on that archive.
- `x` Extract the named *file* from the archive and restore it to the system. If a named *file* matches a directory whose contents was written to the archive, this directory is (recursively) extracted. If a named *file* on tape does not exist on the system, the *file* is created as follows:
  - The user, group, and other protections are restored from the tape.
  - The modification time is restored from the tape unless the `m` function modifier is specified.
  - The file owner (*uid*) and group owner (*gid*) are normally that of the restoring process.
  - The set-user-ID, set-group-ID, and sticky bits are not set automatically. The `o` and `p` function modifiers control the restoration of protection; see below for more details.

  If the files exist, their modes are not changed, but the set-user-ID, set-group-ID and sticky bits are cleared. If no *file* argument is given, the entire content of the archive is extracted. Note that if several files with the same name are on the archive, the last one overwrites all earlier ones.

The following function modifiers can be used in addition to the function letters listed above (note that some modifiers are incompatible with some functions):

A Suppress warning messages that `tar` did not archive a file's access control list. By default, `tar` writes a warning message for each file with optional ACL entries.

b Cause `tar` to use the next *arg* argument as the blocking factor for archive records. The default is 20; the maximum is at least 20. However, if the `f -` modifier is used to specify the standard input, the default blocking factor is 1.

Blocking factor is determined automatically when reading 9-track tapes (key letters `x` and `t`). [On 9-track tapes, the physical tape record length is the same as the block size. The block size is defined as the logical record size times the blocking factor (number of logical records per block).] The blocking factor must be specified when reading flexible disks and cartridge tapes if they were written with a blocking factor other than the default.

If a `tar` file is read using a blocking factor not equal to the blocking used when the file was written, an error may occur at the end of the file but there may or may not be an actual error in the read. To prevent this problem, a blocking factor of `1` can be used, although performance may be reduced somewhat. `tar` writes logical records of 512 bytes, independent of how logical records may be defined elsewhere by other programs (such as variable-length records (lines) within an ASCII text file).

f Cause `tar` to use the next *arg* argument as the name of the archive instead of `/dev/rmt/0m`. If the name of the file is `-`, `tar` writes to the standard output or reads from the standard input, whichever is appropriate, and the default blocking factor becomes 1. Thus, `tar` can be used as the head or tail of a pipeline. `tar` can also be used to move hierarchies with the command:

```
cdfromdir; tar cf - . | (cd  todir ; tar xf -)
```

h
Force `tar` to follow symbolic links as if they were normal files or directories. Normally, `tar` does not follow symbolic links.

H
Cause all entries in hidden directories (context-dependent files) to be written to the archive (see *cdf*(4)). Normally, `tar` only writes the entry in the CDF that matches the context of the `tar` process. See *getcontext*(2). This modifier works only when writing archives. When reading archives, `tar` automatically restores hidden directories if the archive was created with the `H` modifier.

l
Tell `tar` to complain if it cannot resolve all of the links to the files being saved. If `l` is not specified, no error messages are printed.

m
Tell `tar` not to restore the modification time written on the archive. The modification time of the file will be the time of extraction.

o
Suppress writing certain directory information that older versions of `tar` cannot handle on input. `tar` normally writes information specifying owners and modes of directories in the archive. Earlier versions of `tar`, when encountering this information, give error messages of the form:

name`/: cannot create.`

This function modifier suppresses writing that information.

When `o` is used for reading, it causes the extracted *file* to take on the user and group ID (*uid* and *gid*) of the user running the program rather than those of the tape. This is the default for the ordinary user and can be overridden, to the extent that system protections allow, by using the `p` function modifier.

p
Cause *file* to be restored to the original modes and ownerships written on the archive, if possible. This is the default for the superuser, and can be overridden by the `o` function modifier. If system protections prevent the ordinary user from executing `chown()`, the error is ignored, and the ownership is set to that of the restoring process (see *chown*(2)). Set-user-ID, set-group-ID, and sticky bit information are restored as

allowed by the protections defined by `chmod()` if the *chown* operation above succeeds.

# d
Specify a particular tape drive and density where # is a tape drive number (`0`, ..., `7`), and *d* is the density: (`l` = low (800 bpi), `m` = medium (1600 bpi), or `h` = high (6250 bpi)). This modifier selects the drive on which the nine-track tape is mounted. The default is `0m`.

v
Normally, `tar` does its work silently. The **v** (verbose) function modifier causes `tar` to type the name of each file it treats, preceded by the function letter. With the `t` function, **v** gives more information about the tape entries than just the name.

V
Same as the **v** function modifier except that when using the `t` option, `tar` also prints out a letter indicating the type of the archived file.

w
Cause `tar` to print the action being taken, followed by the name of the file, then wait for the user's confirmation. If the user answers **y**, the action is performed. Any other input means "no".

The following option can be included in the file list:

-C*directory* `tar` performs a `chdir()` to *directory* (see *chdir*(2)). This allows multiple directories not related by a close or common parent to be archived using short relative path names.

When end-of-tape is reached, `tar` prompts the user for a new special file and continues.

If a nine-track tape drive is used as the output device, it must be configured in Berkeley-compatibility mode (see *mt*(7)).

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings output when listing the contents of an archive with the `-v option.`

LANG determines the language equivalent of `y` (for yes/no queries).

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `tar` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

ERRORS

`tar` complains about bad key characters and tape read/write errors.

`tar` complains if not enough memory is available to hold the link tables.

EXAMPLES

Create a new archive on `/dev/rfd.0` and copies `file1` and `file2` onto it, using the default blocking factor of 20. The *key* is made up of one function letter (**c**) and two function modifiers (**v**, and **f**):

```
tar cvf /dev/rfd.0 file1 file2
```

Archive files from `/usr/include` and `/etc`:

```
tar cv -C /usr/include -C /etc
```

WARNINGS

The default version has changed from `O` to **N** beginning with HP-UX Release 8.0.

Due to internal limitations in the header structure, not all file names of fewer than 256 characters fit when using the **N** version key. If a file name does not fit, `tar` prints a message and does not archive the file.

Link names are still limited to 100 characters when using the **N** version key.

There is no way to ask for the *n*-th occurrence of a file.

Tape errors are handled ungracefully.

The `u` function key can be slow.

If the archive is on a flexible disk or cartridge tape, and if the blocking factor specified on output is not the default, the same blocking factor must be specified on input. This is because the blocking factor is not explicitly stored on the archive. Updating the archive without following this rule can destroy it.

Some previous versions of `tar` have claimed to support selective listing of file names using the `t` function key with a list. This appears to be an error in the documentation because the capability does not appear in the original source code.

There is no way to restore an absolute path name to a relative position.

`tar` always pads information written to an archive up to the next multiple of the block size. Therefore, if you are creating a small archive and write out one block of information, `tar` reports that one block was written, but the actual size of the archive might be larger if the `b` function modifier is used.

Note that `tar c0m` is not the same as `tar cm0`.

Do not create archives on block special devices. Attempting to do so can causes excessive wear, leading to premature drive hardware failure.

Access Control Lists

Access control list descriptions in this entry apply only to standard HP-UX operating systems. If HP-UX BLS software has been installed, access control lists are handled differently. Refer to HP-UX BLS documentation for information about access control lists in the HP-UX BLS environment.

AUTHOR

`tar` was developed by AT&T, the University of California, Berkeley, HP and POSIX.

FILES

```
/dev/rmt/*
/dev/rfd.*
/tmp/tar*
```

DEPENDENCIES

Series 700/800

The `-r` and `-u` options are not supported on QIC devices. If these options are used with QIC devices, `tar` fails and displays the message:

```
tar: option not supported for QIC devices
```

SEE ALSO

ar(1), cpio(1), mt(7), getcontext(2), cdf(4), acl(5).

STANDARDS CONFORMANCE

`tar`: SVID2, XPG2, XPG3

NAME

tbl - format tables for nroff

SYNOPSIS

`tbl` [`-TX`] [*file* ...]

DESCRIPTION

`tbl` is a preprocessor that formats tables for *nroff*(1). The input files are copied to the standard output, except for lines between `.TS` and `.TE` command lines, which are assumed to describe tables and are reformatted by `tbl`. (The `.TS` and `.TE` command lines are not altered by `tbl`).

`.TS` is followed by global options. The available global options are:

`center`	center the table (default is left-adjust);
`expand`	make the table as wide as the current line length;
`box`	enclose the table in a box;
`doublebox`	enclose the table in a double box;
`allbox`	enclose each item of the table in a box;
`tab` (*x*)	use the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi-colon (`;`).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, indicate where vertical bars are to appear between columns, or determine column width, inter-column spacing, etc. The available key-letters are:

`c`	center item within the column;
`r`	right-adjust item within the column;
`l`	left-adjust item within the column;
`n`	numerically adjust item in the column: units positions of numbers are aligned vertically;
`s`	span previous item on the left into this column;
`a`	center longest line in this column, then left-adjust all other lines in this column with respect to that centered line;
`^`	span down previous entry in this column;
`_`	replace this entry with a horizontal line;
`=`	replace this entry with a double horizontal line.

The characters `B` and `I` stand for the bold (font position 3) and italic (font position 2) fonts, respectively; the character `|` indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by `.TE`. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only `_` or `=`, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only `_` or `=`, then that item is replaced by a single or double line.

The `-TX` option forces `tbl` to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (such as line printers).

If no file names are given as arguments (or if `-` is specified as the last argument), `tbl` reads the standard input, and thus can be used as a filter. When used with `neqn`, `tbl` should be used first to minimize the volume of data passed through pipes (see *neqn*(1)).

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the interpretation of text as single- and/or multi-byte characters.

`LC_NUMERIC` determines the radix character used in numerical data.

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` or `LC_NUMERIC` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `tbl` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

If we redefine the tab character to a semicolon, then the input:

```
.TS
center box tab(;) ;
cB s s
cI | cI s
^  | c  c
l  | n  n.
Household Population
_
Town;Households
;Number;Size
=
Bedminster;789;3.26
Bernards Twp.;3087;3.74
Bernardsville;2018;3.30
Bound Brook;3425;3.04
Bridgewater;7897;3.81
Far Hills;240;3.19
.TE
```

yields:

Household Population		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

The `tbl` command is used most often with `nroff` and `col` (see *col*(1)). A common usage is:

```
tbl filename | nroff -m macro_package_name  | col
```

WARNINGS

See WARNINGS under *nroff*(1).

SEE ALSO

col(1), mm(1), neqn(1), nroff(1), soelim(1), mm(5).

`tbl` tutorial in *Text Formatters Users Guide.*

NAME

tcio - Command Set 80 CS/80 Cartridge Tape Utility

SYNOPSIS

`tcio -o[dervVZ]` [`-l` *number* [`-n` *limit*]] [`-S` *buffersize*] [`-T` *tty*] *file*

`tcio -i[drvZ]` [`-l` *number* [`-n` *limit*]] [`-S` *buffersize*] [`-T` *tty*] *file*

`tcio -u[rvV]` [`-l` *number*] [`-m` *blocknumber*] *file*

DESCRIPTION

`tcio` is designed to optimize the data transfer rate between certain cartridge tape units and the controlling computer. When used in conjunction with other utilities (such as `cpio` a significant improvement in throughput can be obtained, in addition to reducing the wear and tear on the tape cartridges and drives. With autochanger mechanisms, `tcio` provides the capability of loading a specified cartridge, or automatically switching to successive cartridges as needed. With the utility operation, `tcio` provides functions that are unique to cartridge tapes.

`tcio` commands take one of the following forms:

`tcio -o` (copy out) Reads the standard input and writes the data to the CS/80 Cartridge Tape Unit specified by *file*.

`tcio -i` (copy in) Reads the CS/80 Cartridge Tape Unit specified by *file* and writes the data to the standard output.

`tcio -u` (utility) Performs utility functions on the cartridge tape, such as unload, mark, and/or verify the cartridge.

In all cases, *file* must refer to a character special file associated with a CS/80 cartridge tape unit.

During input and output operations, `tcio` enables immediate report mode on cartridge tape units that support this mode (see DEPENDENCIES). During writing, this mode enables the drive to complete a write transaction with the host before the data has actually been written to the tape from the drive's buffer. This allows the host to start gathering data for the next write request while the data for the previous request is still in the process of being written. During reading, this mode enables the drive to read ahead after completing a host read request. This allows the drive to gather data for future read requests while the host is still processing data from the previous read request. Under favorable conditions, immediate report mode allows the drive to stream the tape continuously across multiple read/write requests, rather than having to reposition the tape between each read/write request. See *ct*(7) for more information.

By default, `tcio` writes a tape mark in the first block on each tape to prevent the tape from being image restored onto a disk. It also uses the last block on each tape to hold a flag indicating whether or not the tape is the last tape in a multi-tape sequence.

Options

Every `tcio` command must be followed by a `-o`, `-i`, or `-u` option to indicate the type of operation being performed. In addition, the following command options are recognized. They can be specified in any order, but all must precede the *file* name. Options without parameters can be listed individually (each preceded by a `-`) or grouped together. Options with parameters require the parameter, and must be listed individually.

`-d` Print a checksum to standard error. The checksum is a 32-bit unsigned addition of all bytes written to or read from the tape, providing an extra check of data validity (in addition to tape verification). The checksum value is only reported to the user, and is not written on the media; thus, the user must manually record and check it. The checksum is valid *only* if the same number of bytes are read from the tape as were written to it; in other words, the checksum as a data verification test is meaningless unless the `-e` option was used when writing the tape. This option is independent of the verbose modifier.

`-e` Cause a tape mark to be written on the nearest 1024-byte boundary following the end of the data. When a tape containing an end-of-data tape mark is read back, the read terminates upon encountering the tape mark. Thus, by using this option, the checksums generated by the input and output operations are guaranteed to agree.

-r Unload the tape from the drive. On autochanger units, the tape is returned to the magazine.

-v Verbose mode; prints information and error messages to standard error.

-V This option turns off tape verification. Some cartridge tape units (see DEPENDENCIES) provide hardware for verifying the data output to the tape (called "read-while-write"). For these units software-driven verification is somewhat redundant, and this option is suggested as a means of reducing wear on tape heads and transport mechanisms. However, read-while-write verification does not completely eliminate all risk of data loss, so software verification may still be desired in situations where data preservation is critically important.

For drives that do not have the read-while-write hardware, a separate verification operation is suggested. Thus, it is recommended that this option not be used with drives that do not support read-while-write.

-Z Prevents `tcio` from writing a file mark in the first and last blocks. This option should be used with care because a tape without a tape mark in block zero can be image-restored to a disk.

-l *number* This option is intended solely for autochanging tape drives. For input or output operations (`-i` or `-o`) the `-l` option selects the cartridge specified by *number* from the magazine as the first cartridge used in the transfer. For utility operations (`-u` option), `tcio` loads the cartridge specified by *number* into the drive. (Note: the autochanger must be in selective mode for the autochanger options to work properly.) Whitespace between `-l` and *number* is optional.

-m *blocknumber*
This option writes a tape mark on a tape at the specified block. A tape mark in block zero of the tape prevents it from being image-restored to a disk. Whitespace between **-m** and *blocknumber* is optional.

-n *limit* This option specifies the maximum number of cartridges to be allowed in a multitape transfer. It applies only to autochanger type units, and must be preceded by the `-l` option. Thus, `-l` starts the transfer by loading cartridge *number* and uses at most *limit* cartridges. If `-l` is specified without **-n**, `tcio` quietly assumes all remaining cartridges (in ascending order) in the magazine. Whitespace between **-n** and *limit* is optional.

-S *buffersize* Enable specification of buffer size. This option forces allocation of a block of memory to be used in reading or writing the tape. The size of the buffer in bytes is 1024 times the value specified for *buffersize*. If *buffersize* is less than 4, it is silently increased to 4. A *buffersize* greater than 64 is silently decreased to 64. If *buffersize* is not specified, `tcio` allocates a 64K-byte buffer. Whitespace between **-S** and *buffersize* is optional.

On tape units that support immediate report, a significant performance increase can often be obtained by using a smaller buffer. 8 Kbytes is the recommended buffer size for these units. On tape units that do not support the immediate report mode, or on tape units that share a controller with a disk (see DEPENDENCIES) that is simultaneously being accessed, an increase in performance can usually be obtained with a larger buffer. 64K bytes, the default, is the recommended buffer size for these units.

-T *tty* Specify *tty* as an alternative to `/dev/tty`. Normally `/dev/tty` is opened by `tcio` when terminal interaction is required. The specified file *tty* is opened instead of `/dev/tty`. Whitespace between **-T** and *tty* is optional. If no input device is available, use `/dev/null`.

EXAMPLES

Copy the contents of a directory into an archive:

```
ls | cpio -o | tcio -o /dev/rct/c0d1
```

Restore it:

```
    tcio -i /dev/rct/c0d1 | cpio -i
```

Unload the cartridge from the drive (without verifying the tape):

```
    tcio -urV /dev/rct/c0d1
```

Copy all files in the current directory to the tape specified by the device file **/dev/rct/c1d0s2**. The device has a read-while-write head, so verify is turned off; a buffer size (option **-S**) of 8 blocks (8 Kbytes) is to be used:

```
    ls | cpio -o | tcio -oV -S 8 /dev/rct/c1d0s2
```

Assume that the cartridge tape unit is an autochanger on controller 2, with 8 tapes in the magazine. Start writing with cartridge 3, and use at most 4 cartridges before prompting the user for additional media:

```
    find usr -cpio | tcio -oV -S 8 -l 3 -n 4 /dev/rct/c2
```

DEPENDENCIES

HP 7941CT, HP 9144A, HP 9145A, and HP 35401

These cartridge tape devices contain read-while-write hardware and support immediate report mode.

HP 7942, HP 7946

These cartridge tape devices contain read-while-write hardware and support immediate report mode. Use of a small buffer size is not recommended with these shared-controller devices when simultaneous access to the disk is also required because the intervening disk accesses prevent proper tape streaming.

HP 7908, HP 7911, HP 7912, and HP 7914

These cartridge tape devices do not contain read-while-write hardware, and therefore do not support immediate report mode.

AUTHOR

`tcio` was developed by HP.

SEE ALSO

ct(7).

HP-UX System Administrator Manuals.

NAME

tee - pipe fitting

SYNOPSIS

`tee` [`-i`][`-a`][*file*] ...

DESCRIPTION

`tee` transcribes the standard input to the standard output and makes copies in the *files*. The `-i` option ignores interrupts; the `-a` option causes the output to be appended to the *files* rather than overwriting them.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, `tee` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

Upon completion, `tee` returns one of the following values:

0	Successful completion.
>0	Error condition occured.

EXAMPLES

Write a list of users to the screen and also append the list to the file `hunt`:

```
who | tee -a hunt
```

STANDARDS CONFORMANCE

`tee`: SVID2, XPG2, XPG3, POSIX.2

NAME

telnet - user interface to the TELNET protocol

SYNOPSIS

`telnet` [*host* [*port*]]

DESCRIPTION

`telnet` is used to communicate with another host using the TELNET protocol. If `telnet` is invoked without arguments, it enters command mode, indicated by its prompt (`telnet>`). In this mode, it accepts and executes the commands listed below. If `telnet` is invoked with arguments, it performs an `open` command (see below) with those arguments.

Once a connection has been opened, `telnet` enters an input mode. The input mode entered will be either "character at a time" or "line by line", depending on what the remote system supports.

In "character at a time" mode, most text typed is immediately sent to the remote host for processing.

In "line by line" mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The "local echo character" (initially `^E`) can be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the `localchars` toggle is TRUE (the default in line mode; see below), the user's `quit` and `intr` characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see `toggle autoflush` and `toggle autosynch` below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of `quit` and `intr`).

While connected to a remote host, `telnet` command mode can be entered by typing the `telnet` "escape character" (initially `^]`). When in command mode, the normal terminal editing conventions are available.

`telnet` supports eight-bit characters when communicating with the server on the remote host. To use eight-bit characters you may need to reconfigure your terminal or the remote host appropriately (see *stty*(1)). Furthermore, you may have to use the `binary` toggle to enable an 8-bit data stream between `telnet` and the remote host. Note that some remote hosts may not provide the necessary support for eight-bit characters.

If, at any time, `telnet` is unable to read from or write to the server over the connection, the message `Connection closed by foreign host.` is printed on standard error, `telnet` exits with a value of 1.

Commands

The following commands are available in command mode. You need only type enough of each command to uniquely identify it (this is also true for arguments to the `mode`, `set`, `toggle`, and `display` commands).

`open` *host* [*port*]
: Open a connection to the named host at the indicated port. If no port is specified, `telnet` attempts to contact a TELNET server at the standard TELNET port. The hostname can be either the official name or an alias as understood by `gethostbyname()` (see *gethostent*(3N)), or an Internet address specified in the dot notation as described in *hosts*(4). If no hostname is given, `telnet` prompts for one.

`close`
: Close a TELNET session. If the session was started from command mode, `telnet` returns to command mode; otherwise `telnet` exits.

`quit`
: Close any open TELNET session and exit `telnet`. An end of file (in command mode) will also close a session and exit.

`z`
: Suspend `telnet`. If `telnet` is run from a shell that supports job control, (such as *csh*(1) or *ksh*(1)), the `z` command suspends the TELNET session and returns the user to the shell that invoked `telnet`. The job can then be resumed with the `fg` command (see *csh*(1) or *ksh*(1)).

`mode` *Mode*
: Change `telnet`'s user input mode to *Mode*. *Mode* can be `character` (for "character at a time" mode) or `line` (for "line by line" mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the

requested mode is entered. In `character` mode, `telnet` sends each character to the remote host as it is typed. In `line` mode, `telnet` gathers user input into lines and transmits each line to the remote host when the user types carriage return, linefeed, or EOF (normally `^D`; see *stty*(1)). Note that setting line-mode also sets local echo. Applications that expect to interpret user input character by character (such as `more`, `csh`, `ksh`, and `vi`) do not work correctly in line mode.

`status` Show current status of `telnet`. `telnet` reports the current escape character. If `telnet` is connected, it reports the host to which it is connected and the current `mode`. If `telnet` is not connected to a remote host, it reports `No connection.`

`display` [*argument* ...]
Displays all or some of the `set` and `toggle` values (see below).

`?` [*command*] Get help. With no arguments, `telnet` prints a help summary. If a command is specified, `telnet` prints the help information available about that command only. Help information is limited to a one-line description of the command.

`!` [*shell_command*]
Shell escape. The `SHELL` environment variable is checked for the name of a shell to use to execute the command. If no *shell_command* is specified, a shell is started and connected to the user's terminal. If `SHELL` is undefined, `/bin/sh` is used.

`send` *arguments*
Sends one or more special character sequences to the remote host. Each *argument* can have any of the following values (multiple *arguments* can be specified with each `send` command):

`escape`	Sends the current `telnet` escape character (initially `^]`).
`synch`	Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work to some systems -- if it doesn't work, a lower case "r" may be echoed on the terminal).
`brk`	Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.
`ip`	Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.
`ao`	Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output *from* the remote system *to* the user's terminal.
`ayt`	Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.
`ec`	Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.
`el`	Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.
`ga`	Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.
`nop`	Sends the TELNET NOP (No OPeration) sequence.
`?`	Prints out help information for the `send` command.

`set` *variable_name value*
Set any one of a number of `telnet` variables to a specific value. The special value `off` turns off the function associated with the variable. The values of variables can be interrogated by using the `display` command. The following *variable_names* can be specified:

`echo`
This is the value (initially `^E`) which, when in line-by-line mode, toggles between doing local

echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, for example, a password).

`escape`
: This is the `telnet` escape character (initially `^[`) which causes entry into `telnet` command mode (when connected to a remote system).

`interrupt`
: If `telnet` is in `localchars` mode (see `toggle localchars` below) and the *interrupt* character is typed, a TELNET IP sequence (see `send ip` above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's `intr` character.

`quit`
: If `telnet` is in `localchars` mode (see `toggle localchars` below) and the `quit` character is typed, a TELNET BRK sequence (see `send brk` above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's `quit` character.

`flushoutput`
: If `telnet` is in `localchars` mode (see `toggle localchars` below) and the `flushoutput` character is typed, a TELNET AO sequence (see above) is sent to the remote host. The initial value for the flush character is `^O`.

`erase`
: If `telnet` is in `localchars` mode (see `toggle localchars` below), **and** if `telnet` is operating in character-at-a-time mode, then when this character is typed, a TELNET EC sequence (see `send` *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's `erase` character.

`kill`
: If `telnet` is in `localchars` mode (see `toggle localchars` below), *and* if `telnet` is operating in character-at-a-time mode, then when this character is typed, a TELNET EL sequence (see `send el` above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's `kill` character.

`eof` If `telnet` is operating in line-by-line mode, entering this character as the first character on a line causes this character to be sent to the remote system. The initial value of the `eof` character is taken to be the terminal's `eof` character.

`toggle` *arguments ...*
Toggle (between TRUE and FALSE) various flags that control how `telnet` responds to events. More than one argument can be specified. The state of these flags can be interrogated by using the `display` command. Valid arguments are:

`localchars`
: If TRUE, the `flush`, `interrupt`, `quit`, `erase`, and `kill` characters (see `set` above) are recognized locally, and transformed into appropriate TELNET control sequences (respectively `ao`, `ip`, `brk`, `ec`, and `el`; see `send` above). The initial value for this toggle is **TRUE** in line-by-line mode, and **FALSE** in character-at-a-time mode.

`autoflush`
: If `autoflush` and `localchars` are both TRUE, whenever the `ao`, `intr`, or `quit` characters are recognized (and transformed into TELNET sequences – see `set` above for details), `telnet` refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET *Timing Mark* option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE.

`autosynch`
: If `autosynch` and `localchars` are both TRUE, when either the `intr` or `quit` character is typed (see `set` above for descriptions of the `intr` and `quit` characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure *should* cause the remote system to begin discarding all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

`binary`
Enable or disable the TELNET BINARY option on both input and output. This option should be enabled in order to send and receive 8-bit characters to and from the TELNET server.

`crlf`
If TRUE, end-of-line sequences are sent as an ASCII carriage-return and line-feed pair. If FALSE, end-of-line sequences are sent as an ASCII carriage-return and NUL-character pair. The initial value for this toggle is FALSE.

`crmod`
Toggle carriage return mode. When this mode is enabled, any carriage return characters received from the remote host are mapped into a carriage return and a line feed. This mode does not affect those characters typed by the user; only those received. This mode is only required for some hosts that require the client to do local echoing, but output "naked" carriage returns. The initial value for this toggle is FALSE.

`echo`
Toggle local echo mode or remote echo mode. In local echo mode, user input is echoed to the terminal by the local `telnet` before being transmitted to the remote host. In remote echo, any echoing of user input is done by the remote host. Applications that handle echoing of user input themselves, such as C shell, Korn shell, and `vi` (see *csh*(1), *ksh*(1), and *vi*(1), do not work correctly with local echo.

`options`
Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations are displayed. Options sent by `telnet` are displayed as `''SENT''`, while options received from the TELNET server are displayed as `''RCVD''`. The initial value for this toggle is FALSE.

`netdata`
Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

`?` Displays the legal `toggle` commands.

RETURN VALUE
In the event of an error, or if the TELNET connection is closed by the remote host, `telnet` returns a value of 1. Otherwise it returns zero.

DIAGNOSTICS
The following diagnostic messages are displayed by `telnet`:

`telnet/tcp: Unknown service`
`telnet` was unable to find the TELNET service entry in the *services*(4) database .

hostname`: Unknown host`
`telnet` was unable to map the host name to an Internet address. Your next step should be to contact the system administrator to check whether there is an entry for the remote host in the `hosts` database (see *hosts*(4)).

`?Invalid command`
An invalid command was typed in `telnet` command mode.

system call`>:` ...
An error occurred in the specified system call. See the appropriate manual entry for a description of the error.

AUTHOR
`telnet` was developed by the University of California, Berkeley.

SEE ALSO
csh(1), ksh(1), login(1), rlogin(1), stty(1), telnetd(1M), hosts(4), services(4). termio(7),

NAME

test - condition evaluation command

SYNOPSIS

`test` *expr*

`[` *expr* `]`

DESCRIPTION

`test` evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned. `test` also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

`-r` *file*	true if *file* exists and is readable.
`-w` *file*	true if *file* exists and is writable.
`-x` *file*	true if *file* exists and is executable.
`-f` *file*	true if *file* exists and is a regular file.
`-d` *file*	true if *file* exists and is a directory.
`-c` *file*	true if *file* exists and is a character special file.
`-b` *file*	true if *file* exists and is a block special file.
`-p` *file*	true if *file* exists and is a named pipe (fifo).
`-u` *file*	true if *file* exists and its set-user-ID bit is set.
`-g` *file*	true if *file* exists and its set-group-ID bit is set.
`-k` *file*	true if *file* exists and its sticky bit is set.
`-s` *file*	true if *file* exists and has a size greater than zero.
`-H` *file*	true if *file* exists and is a hidden directory (see *cdf*(4)).
`-h` *file*	true if *file* exists and is a symbolic link.
`-t` [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
`-z` *s1*	true if the length of string *s1* is zero.
`-n` *s1*	true if the length of the string *s1* is non-zero.
s1 `=` *s2*	true if strings *s1* and *s2* are identical.
s1 `!=` *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 `-eq` *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons `-ne`, `-gt`, `-ge`, `-lt`, and `-le` can be used in place of `-eq`.

These primaries can be combined with the following operators:

`!`	unary negation operator.
`-a`	binary AND operator.
`-o`	binary OR operator (`-a` has higher precedence than `-o`).
`(` *expr* `)`	parentheses for grouping.

Note that all the operators and flags are separate arguments to `test`. Note also that parentheses are significant to the shell and therefore must be escaped.

`test` is interpreted directly by the shell, and therefore does not exist as a separate executable program.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Exit if there are not two or three arguments:

```
if [ $# -12 2 -o $# -gt 3 ]; then exit 1; fi
```

Create a new file containing the text string `default` if the file does not already exist:

```
[ ! -f thisfile ] && echo default > thisfile
```

Wait for myfile to become non-readable:

```
while test -r myfile
do
    sleep 30
done
echo '"myfile" is no longer readable'
```

WARNINGS

When the `[` form of this command is used, the matching `]` must be the final argument, and both must be separate arguments from the arguments they enclose (white space delimiters required.

Parentheses and other special shell metacharacters intended to be handled by test must be escaped or quoted when invoking `test` from a shell.

Script writers should be careful when dealing with user-supplied input that could be confused with primaries and operators. Unless the application writer can control all the sources of input to the script, invocations such as:

```
test "$1" -a "$2"
```

or

```
[ "$1"  -o  "$2" ]
```

should be written as:

```
test "$1" && test "$2"
```

or

```
[ "$1" ] || [ "$2" ]
```

to avoid problems if the user supplies values such as `$1` set to `!` and `$2` set to the null string. However, note that `-a` has higher precedence than `-o` in `test`, while `&&` and `||` have equal precedence in the shell.

Another way to avoid such problems when comparing strings is to insert some non-operator character at the beginning of both operands:

```
test "X$response" = "Xexpected string"
```

This approach does not work with numeric comparisons or the unary operators because it would affect the operand being checked.

AUTHOR

`test` was developed by the University of California, Berkeley and HP.

SEE ALSO

find(1), sh(1), cdf(4).

STANDARDS CONFORMANCE

`test`: SVID2, XPG2, XPG3, POSIX.2

`[`: SVID2, XPG2, XPG3, POSIX.2

NAME

tftp - trivial file transfer program

SYNOPSIS

`tftp` [*host*]

DESCRIPTION

`tftp` is the user interface to the Internet TFTP (Trivial File Transfer Protocol), that allows users to transfer files to and from a remote machine. The remote *host* can be specified on the command line, in which case `tftp` uses *host* as the default host for future transfers (see the `connect` command below).

Commands

Once `tftp` is running, it issues the prompt `tftp>` and recognizes the following commands:

`connect` *host-name* [*port*]
: Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections betweeen transfers; thus, the `connect` command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the `connect` command; the remote host can be specified as part of the `get` or `put` commands.

`mode` *transfer-mode*
: Set the mode for transfers; *transfer-mode* can be one of `ascii` or `binary` (default is `ascii`).

`put` *file*
`put` *localfile remotefile*
`put` *file1 file2 ... fileN remotedirectory*
: Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form *host* **:** *filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a UNIX-like machine.

`get` *filename*
`get` *remotename localname*
`get` *file1 file2 ... fileN*
: Get a file or set of files from the specified *source*s. *source* can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form *host* **:** *filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

`quit` Exit `tftp`. Typing the end-of-file character also causes an exit.

`verbose` Toggle verbose mode.

`trace` Toggle packet tracing.

`status` Show current status.

`rexmt` *retransmission-timeout*
: Set the per-packet retransmission timeout, in seconds.

`timeout` *total-transmission-timeout*
: Set the total transmission timeout, in seconds.

`ascii` Shorthand for "mode ascii"

`binary` Shorthand for "mode binary"

`?` [*command-name ...*]
: Print help information.

WARNINGS

Since there is no user-login or validation within the TFTP protocol, the remote site probably has some sort of file-access restrictions in place. The exact methods are specific to each site and are therefore difficult to document here.

AUTHOR

`tftp` was developed by the University of California, Berkeley.

SEE ALSO

tftpd(1m).

NAME

time - time a command

SYNOPSIS

`time` *command*

DESCRIPTION

command is executed. Upon completion, `time` prints the elapsed time during the command, the time spent in the system, and the time spent executing the command. Times are reported in seconds.

Execution time can depend on the performance of the memory in which the program is running.

The times are printed on standard error.

SEE ALSO

`times` command in *sh*(1), *timex*(1), *times*(2).

STANDARDS CONFORMANCE

`time`: SVID2, XPG2, XPG3

NAME

timex - time a command; report process data and system activity

SYNOPSIS

`timex [-o][-p[fhkmrt]][-s]` *command*

DESCRIPTION

`timex` reports in seconds the elapsed time, user time, and system time spent in execution of the given *command*. Optionally, process accounting data for *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of `timex` is written on the standard error.

Options

`-o`	Report the total number of blocks read or written and total characters transferred by *command* and all its children.
`-p[fhkmrt]`	List process accounting records for *command* and all its children. The suboptions `f`, `h`, `k`, `m`, `r`, and `t` modify the data items reported. They behave as defined in *acctcom*(1M). The number of blocks read or written and the number of characters transferred are always reported.
`-s`	Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

EXAMPLES

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex -opskmt sh
```

session commands

EOT

WARNINGS

Process records associated with *command* are selected from the accounting file `/usr/adm/pacct` by inference, since process genealogy is not available. Background processes having the same user-ID, terminal-ID, and execution time window are spuriously included.

SEE ALSO

sar(1), acctcom(1M).

STANDARDS CONFORMANCE

`timex`: SVID2

NAME

top - display and update information about the top processes on the system

SYNOPSIS

`top` [`-s` *time*] [`-d` *count*] [`-q`] [`-u`] [`-n` *number*]

DESCRIPTION

`top` displays the top processes on the system and periodically updates the information. Raw CPU percentage is used to rank the processes.

Options

`top` recognizes the following command-line options:

`-s` *time*	Set the delay between screen updates to *time* seconds. The default delay between updates is 5 seconds.
`-d` *count*	Show only *count* displays, then exit. A display is considered to be one update of the screen. This option is used to select the number of displays to be shown before the program exits.
`-q`	This option runs the `top` program at the same priority as if it is executed via a `nice -20` command so that it will execute faster (see *nice*(1)). This can be very useful in discovering any system problem when the system is very sluggish. This option is accessibly only to users who have appropriate privileges.
`-u`	User ID (uid) numbers are displayed instead of usernames. This improves execution speed by eliminating the additional time required to map uid numbers to user names.
`-n` *number*	Show only *number* processes per screen. Note that this option is ignored if *number* is greater than the maximum number of processes that can be displayed per screen.

Screen-Control Commands

When displaying multiple-screen data, `top` recognizes the following keyboard screen-control commands:

`j`	Display next screen if the current screen is not the last screen.
`k`	Display previous screen if the current screen is not the first screen.
`t`	Display the first (top) screen.

Program Termination

To exit the program and resume normal user activities, type `q` at any time.

Display Description

Three general classes of information are displayed by `top`:

`System Data:`

The first few lines at the top of the display show general information about the state of the system, including:

- System name and current time.
- Load averages in the last one, five, and fifteen minutes.
- Number of existing processes and the number of processes in each state (sleeping, waiting, running, starting, zombie, and stopped).
- Percentage of time spent in each of the processor states (user, nice, system, idle, interrupt and swapper) per processor on the system.
- Average value for each of the processor states (only on multi-processor systems).

`Memory Data`

Includes virtual and real memory in use (with the amount of memory considered "active" in parentheses) and the amount of free memory.

`Process Data`

Information about individual processes on the system. When process data cannot fit on a single screen, `top` divides the data into two or more screens. To view multiple-screen data, use the `j`, `k`, and `t` commands described previously. Note that the system- and memory-data displays are present in each screen of multiple-screen process data.

Process data is displayed in a format similar to that used by *ps*(1):

`CPU`	Processor number on which the process is executing (only on multi-processor systems).
`TTY`	Terminal interface used by the process.
`PID`	Process ID number.
`USERNAME`	Name of the owner of the process. When the `-u` option is specified, the user ID (uid) is displayed instead of `USERNAME`.
`PRI`	Current priority of the process.
`NI`	Nice value ranging from –20 to +20.
`SIZE`	Total size of the process in kilobytes. This includes text, data, and stack.
`RES`	Resident size of the process in kilobytes. The resident size information is, at best, an approximate value.
`STATE`	Current state of the process. The various states are `sleep`, `wait`, `run`, `idl`, `zomb`, or `stop`.
`TIME`	Number of system and CPU seconds the process has consumed.
`%WCPU`	Weighted CPU (central processing unit) percentage.
`%CPU`	Raw CPU percentage. This field is used to sort the top processes.
`COMMAND`	Name of the command the process is currently running.

EXAMPLES

`top` can be executed with or without command-line options. To display five screens of data at two-second intervals then automatically exit, use:

```
top -s2 -d5
```

AUTHOR

`top` was developed by HP and William LeFebvre of Rice University.

NAME

touch - update access, modification, and/or change times of file

SYNOPSIS

`touch` [`-amc`] [`-r` *ref_file*] [`-t` *time*] *file_name* ...

Obsolescent:

`touch` *time_str file_name* ...

DESCRIPTION

`touch` updates the access, modification, and last-change times of each argument. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used.

The `-r` and `-t` options are mutually exclusive.

Options

The following options are available:

`-a` Change the access time of *file_name* to *time*, or to the current time if *time* is not specified. Do not change the modification time unless `-m` is also specified.

`-m` Change the modification time of *file_name* to *time*, or to the current time if *time* is not specified. Do not change the access time unless `-a` is also specified.

`-c` Silently prevent `touch` from creating the file if it did not previously exist. Do not write any diagnostic messages concerning this condition.

`-r` *ref_file*
Use the corresponding time of *ref_file* instead of the current time.

`-t` *time* Use the specified *time* instead of the current time. The option argument is a decimal number of the form:

[[CC] YY] MMDDhhmm [.SS]

where each two digits represents the following:

CC	The first two digits of the year.
YY	The second two digits of the year.
MM	The month of the year (01-12).
DD	The day of the month (01-31).
hh	The hour of the day (00-23).
mm	The minute of the hour (00-59).
SS	The second of the minute (00-61).

If neither *CC* nor *YY* is given, the current year is assumed. If *YY* is specified, but *CC* is not, *CC* is derived as follows:

If *YY* is:	*CC* becomes:
69-99	19
00-68	20

If the resulting time value precedes the Epoch (00:00:00 January 1, 1970 Greenwich Mean Time), `touch` exits immediately with an error status.

The range for SS is 00 through 61 rather than 00 through 59 to accommodate leap seconds. If SS is 60 or 61, and the resulting time, as affected by the `TZ` environment variable, does not refer to a leap second, the resulting time is one second after a time where SS is 59. If SS is not given a value, it is assumed to be 0.

The syntax shown by the second SYNOPSIS line is recognized when neither the `-r` option, the `-t` option, nor the `--` option delimiter is specified, and the first operand consists of all decimal digits. This operand is interpreted as the *time* argument instead of as a file name. However, in this case, *time_str* is assumed to be of the form:

MMDDhhmm[YY]

This is for backward compatibility. The `-t` form given above is recommended for future portability. The `--` option delimiter can be used before the first *file_name* if there is a possibility that *file_name* consists of all digits, in order to ensure that the first syntax is used.

`touch` succeeds *only* when invoked by the *owner* of the file if any of the following are true:

- A time is specified,
- Only the access time of the file is being updated, or
- Only the modification time of the file is being updated.

In addition, `touch` succeeds when invoked by a user with write permission on the file if both of the following are true:

- No time is specified, *and*
- *Both* the access time and modification time of the file are being updated.

EXTERNAL INFLUENCES

Environmental Variables

`TZ` If the time is specified via the `-t` option, `TZ` is used to interpret the time for the specified time zone.

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`touch` returns zero if all *file_name* arguments were successfully changed.

`touch` returns non-zero and prints out a diagnostic message if an invalid time or a time earlier than the Epoch was specified with the `-t` option, or if the `-r` and `-t` options were both specified, or if one or more of the *file_name* arguments could not be accessed.

EXAMPLES

The following command sets the modification and access times of the file named "bastille" to midnight, July 14, 1989, creating the file if it does not already exist.

```
touch -t 8907140000 bastille
```

The following command does the same thing using the backward-compatible syntax:

```
touch 0714000089 bastille
```

The following command sets the time of the two files named "0714000089" and "bastille" to the current time, creating them if they do not exist:

```
touch -- 0714000089 bastille
```

To create a zero-length file, use any of the following:

```
cat /dev/null >file
cp /dev/null file
```

DEPENDENCIES

NFS:

An attempt to touch a file owned by the super-user on a remote server might fail, even if the invoking user has write permission on the file.

SEE ALSO

date(1), utime(2).

STANDARDS CONFORMANCE

`touch`: SVID2, XPG2, XPG3, POSIX.2

NAME

tput - query terminfo database

SYNOPSIS

tput [-T *type*] *capname*

DESCRIPTION

`tput` uses the `terminfo` database to make terminal-dependent capabilities and information available to the shell (see *terminfo*(4)). `tput` outputs a string if the attribute (*capname*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, `tput` simply sets the exit code (0 for TRUE, 1 for FALSE), and produces no output.

Command-Line Arguments

`tput` recognizes the following command-line arguments:

-T*type*	Indicates the type of terminal. Normally this flag is unnecessary because the default is taken from the environment variable **TERM.**
capname	Indicates the attribute from the `terminfo` database. See *terminfo*(4). In addition, the following *capnames* are supported:

`clear`	Echo the clear-screen sequence for the current terminal.
`init`	Echo the initialize sequence for the current terminal.
`reset`	Echo the sequence that will reset the current terminal.

The words `clear`, `init`, and `reset` are replaced by their local language equivalent (see EXTERNAL INFLUENCES below).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the translation of the words `clear`, `init`, and `reset`.

LC_ALL determines the locale to use. This overrides settings of other environment variables.

LC_MESSAGES determines the language to use for messages.

TERM determines the terminal type if the **-T** option is not specified.

EXAMPLES

Echo clear-screen sequence for the current terminal.

```
tput clear
```

Print the number of columns for the current terminal.

```
tput cols
```

Print the number of columns for the hp2623 terminal.

```
tput -Thp2623 cols
```

Set shell variable
`bold` to stand-out-mode sequence for current terminal.

```
bold='tput smso'
```

This might be followed by a prompt:

```
echo "${bold}Please type in your name: \c"
```

Set exit code to indicate if current terminal is a hardcopy terminal.

```
tput hc
```

DIAGNOSTICS

`tput` fails, prints an error message, and returns the corresponding value when any of the following conditions are encountered.

2	Usage error.
3	Bad terminal type.

4	Bad capability name.

In addition, if a capability name is requested for a terminal that has no value for that capability name (such as `tput -Thp2623 vt`), `-1` is printed.

FILES

`/usr/lib/terminfo/?/*`	terminfo data base
`/usr/include/curses.h` `/usr/include/term.h`	definition files

SEE ALSO

stty(1), terminfo(4).

STANDARDS CONFORMANCE

`tput`: SVID2

NAME

tr - translate characters

SYNOPSIS

`tr` [`-cds`] [*string1* [*string2*]]

DESCRIPTION

`tr` copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options `-cds` can be used:

`-c`	Complements the set of characters in *string1* with respect to all the characters contained in the current character set.
`-d`	Deletes all input characters in *string1*.
`-s`	Squeezes all strings of repeated output characters that appear in *string2* to single characters.

The following abbreviation conventions can be used to introduce ranges of characters or repeated characters into the strings:

c1-*c2* or [*c1*-*c2*]	Stands for the ordered string of collating elements *c1* through *c2*, inclusive.
[`:`*class*`:`] or [[`:`*class*`:`]]	Stands for the string of characters in type *class*, where *class* must be one of **`alpha`**, **`upper`**, **`lower`**, **`digit`**, **`xdigit`**, **`alnum`**, **`space`**, **`print`**, **`punct`**, **`graph`**, or **`cntrl`**. Character classes are expanded in collation order.
[`=`*c*`=`] or [[`=c=`]]	Stands for an equivalence class; i.e., all characters that are defined as having the same primary collation sequence number as *c*.
[`.`*cc*`.`]	Stands for the collating element *cc*. Multi-character collating elements must be represented in this manner to distinguish them from a list of single-character collating elements.
[*a*`*`*n*]	Stands for *n* repetitions of *a*. If the first digit of *n* is `0`, *n* is considered octal; otherwise, *n* is treated as a decimal value. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character `\` can be used as in the shell to remove special meaning from any character in a string. In addition, `\` followed by 1, 2, or 3 octal digits represents the character whose ASCII code is given by those digits.

An ASCII NUL character in *string1* or *string2* can be represented only as an escaped character; i.e. as `\000`, but is treated like other characters and translated correctly if so specified. NUL characters in the input are not stripped out unless the option `-d "\000"` is given.

WARNINGS

`a-z` now represents the range of collating elements from `a` thru `z`, inclusive and not the three characters `a`, `-`, and `z`. The three characters `a`, `-`, and `z` can be represented as `az-` or `a\-z`.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the order of collating elements, the members of equivalence classes, the order in which character classes are expanded, and the identification of multi-character collating elements.

`LC_CTYPE` determines the interpretation of text as single- and/or multi-byte characters, the characters matched by character classes, and the current universe of characters when using the `-c` option.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `tr` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

`tr` exits with one of the following values:

0 All input was processed successfully.

>0 An error occurred.

EXAMPLES

For the ASCII character set and default collation sequence, create a list of all the words in *file1*, one per line in *file2*, where a word is taken to be a maximal string of alphabetics. Quote the strings to protect the special characters from interpretation by the shell (012 is the ASCII code for a new-line (line feed) character:

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

Same as above, but for all character sets and collation sequences:

```
tr -cs "[:alpha:]" "[\012*]" <file1 >file2
```

Translate all lowercase characters in *file1* to uppercase and write the result to standard output.

Note that character classes specified in either *string1* or *string2* are expanded in collation order. Consequently, this example does not produce the desired effect in locales in which there is not a one-to-one mapping between lowercase and uppercase letters, or in which lowercase and uppercase letters collate in a different relative order:

```
tr "[:lower:]" "[:upper:]" <file1
```

Use an equivalence class to identify accented variants of the base character `e` in *file1*, strip them of diacritical marks and write the result to *file2*:

```
tr "[=e=]" "[e*]" <file1 >file2
```

Translates instances of the multi-character collating element `ch` to uppercase only in locales which support two-to-one collation. Note that the single characters `c` and `h` are not affected by this operation unless they are part of a `ch` character sequence.

```
tr "[.ch.]" "[.CH.]" <file1 >file2
```

SEE ALSO

ed(1), sh(1), ascii(5), environ(5), lang(5), regexp(5).

STANDARDS CONFORMANCE

`tr`: SVID2, XPG2, XPG3, POSIX.2

NAME
true, false - return exit status zero or one respectively

SYNOPSIS
true

false

DESCRIPTION
The command **true** does nothing, and returns exit code zero. The command **false** does nothing, and returns exit code one. They are typically used to construct command procedures.

RETURN VALUE
Exit values are:

0	always from *true*.
1	always from *false*.

EXAMPLES
This command loop repeats without end:

```
while true
     do
          command
     done
```

WARNINGS
true is typically used in shell scripts. Some shells provide a built-in version of **true** (and **false**) that is more efficient than the standalone versions.

SEE ALSO
csh(1), ksh(1), sh(1), sh-bourne(1), sh-posix(1).

STANDARDS CONFORMANCE
true: SVID2, XPG2, XPG3, POSIX.2

false: SVID2, XPG2, XPG3, POSIX.2

NAME

tset, reset - terminal-dependent initialization

SYNOPSIS

`tset` [*options*] [`-m` [*ident*] [*test baudrate*] `:`*type*] ... [*type*]

`reset`

DESCRIPTION

`tset` sets up the terminal when logging in on an HP-UX system. It does terminal-dependent processing, such as setting erase and kill characters, setting or resetting delays, and sending any sequences needed to properly initialize the terminal. It first determines the *type* of terminal involved, then does the necessary initializations and mode settings. The type of terminal attached to each HP-UX port is specified in the `/etc/ttytype` data base. Type names for terminals can be found in the files under the `/usr/lib/terminfo` directory (see *terminfo*(4)). If a port is not wired permanently to a specific terminal (not hardwired), it is given an appropriate generic identifier, such as `dialup`.

`reset` performs a similar function, setting the terminal to a sensible default state.

In the case where no arguments are specified, `tset` simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual entry concerns itself with mode and environment initialization, typically done once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script (`.profile` for Bourne shell, Korn shell, and POSIX shell users, see *sh*(1) and *ksh*(1)), or `.login` for *csh*(1) users) it is desirable to give information about the type of terminal that will normally be used on ports that are not hardwired. These ports are identified in `/etc/ttytype` as `dialup` or `plugboard`, etc. To specify what terminal type you usually use on these ports, the `-m` (map) option flag is followed by the appropriate port type identifier, an optional baud rate specification, and the terminal type. (The effect is to "map" from some conditions to a terminal type; that is, to tell `tset` that "If I am on this kind of port, I will probably be on this kind of terminal".) If more than one mapping is specified, the first applicable mapping prevails. A missing port type identifier matches all identifiers. A *baudrate* is specified as with `stty` (see *stty*(1)), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate *test* can be any combination of **>**, **=**, **<**, **@**, and **!**. **@** is a synonym for **=**, and **!** inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to `-m` within single quotes; users of *csh*(1) must also put a \
before any **!** used.

Thus,

```
tset -m 'dialup>300:2622' -m 'dialup:2624' -m 'plugboard:?2623'
```

causes the terminal type to be set to an HP 2622 if the port in use is a dialup at a speed greater than 300 baud, or to an HP 2624 if the port is otherwise a dialup (i.e. at 300 baud or less). If the *type* finally determined by `tset` begins with a question mark, the user is asked for verification that the type indicated is really the one desired. A null response means to use that type; otherwise, another type can be entered. Thus, in the above case, if the user is on a plugboard port, he or she will be asked whether or not he or she is actually using an HP 2623.

If no mapping applies and a final *type* option, not preceded by a `-m`, is given on the command line, that type is used. Otherwise, the identifier found in the `/etc/ttytype` data base is taken to be the terminal type. The latter should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by *tset*, and information about the terminal's capabilities to a shell's environment. This can be done using the `-s` option. From Bourne shell (*sh*(1)), the command:

```
eval 'tset -s options...'
```

or using the C shell, (*csh*(1)):

```
set noglob; eval 'tset -s options...'
```

These commands cause `tset` to generate as output a sequence of shell commands which place the variable **TERM** in the environment; see *environ*(5).

Once the terminal type is known, `tset` engages in terminal mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the full line erase or line-kill) characters, and setting special character delays. Tab and new-line expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character (# on standard systems), the erase character is changed to Back space (`^H`).

Options

`tset` recognizes the following options:

- **-e***c* Set the erase character to be the named character *c*; *c* defaults to `^H` (BACKSPACE). The character *c* can either be typed directly, or entered using circumflex notation used here (e.g., the circumflex notation for control-H is `^H`; in Bourne shell the `^` character should be escaped (`\^`)).
- **-k***c* Set the kill character to *c*. The default *c* is `^X`. If *c* is not specified, the kill character remains unchanged unless the original value of the kill character is null, In which case the kill character is set to @.
- **-** Report terminal type. Whatever type is decided on is reported. If no other flags are given, the only effect is to write the terminal type on the standard output. Has no effect if used with `-s`.
- **-s** Generate appropriate commands (depending on current `SHELL` environment variable) to set **TERM**.
- **-I** Suppress transmitting terminal initialization strings.
- **-Q** Suppress printing the `Erase set to` and `Kill set to` messages.
- **-A** Ask the user for the `TERM` type.
- **-S** Output the strings that would be assigned to `TERM` in the environment rather than generating commands for a shell. In Bourne shell, the following is an alternate way of setting **TERM**:

```
set -- 'tset -S ...'
TERM=$1
```

-h
Force a read of `/etc/ttytype`. When `-h` is not specified, the terminal type is determined by reading the environment unless some mapping is specified.

For compatibility with earlier versions of `tset`, the following flags are accepted, but their use is discouraged:

- **-r** Report to the user in addition to other flags.
- **-E***c* Set the erase character to *c* only if the terminal can backspace. *c* defaults to `^H`.

In addition to capabilities described in `terminfo` (see *termio*(7) and *terminfo*(4)), the following boolean terminfo capabilities are understood by `tset` and `reset`, and can be included in the terminfo database for the purpose of terminal setup:

- **UC** "Uppercase" mode sets character mapping for terminals that support only uppercase characters. Equivalent to `stty lcase`.
- **LC** "Lowercase" mode permits input and output of lowercase characters. Equivalent to `stty -lcase`.
- **EP** Set "even parity". Equivalent to `stty parenb -parodd`
- **OP** Set "odd parity". Equivalent to `stty parenb parodd`.
- **NL** Set "new line" mode. Equivalent to `stty onlret`.
- **HD** Set "half-duplex" mode. Equivalent to `stty -echo`.
- **pt** Set "print tabs" mode. Equivalent to `stty tabs`.

EXAMPLES

These examples all assume the Bourne shell (*sh-bourne*(1)). Note that a typical use of `tset` in a `.profile` also uses the `-e` and `-k` options, and often the `-n` or `-Q` options as well. These options have been omitted here to keep the examples small.

Assume, for the moment, that you are on an HP 2622. This is suitable for typing by hand but not for a `.profile` unless you are *always* on a 2622.

```
export TERM; TERM='tset - 2622'
```

Assume you have an HP 2623 at home which you dial up on, but your office terminal is hardwired and known in `/etc/ttytype`.

```
export TERM; TERM='tset - -m dialup:2623'
```

Suppose you are accessing the system through a switching network that can connect any system to any incoming modem line in an arbitrary combination, making it nearly impossible to key on what port you are coming in on. Your office terminal is an HP 2622, and your home terminal is an HP 2623 running at 1200 baud on dial-up switch ports. Sometimes you use someone else's terminal at work, so you want it to verify what terminal type you have at high speeds, but at 1200 baud you are always on a 2623. Note the placement of the question mark and the quotes to protect the `>` and `?` from interpretation by the shell.

```
export TERM; TERM='tset - -m 'switch>1200:?2622' -m 'switch<=1200:2623''
```

All of the above entries fall back on the terminal type specified in `/etc/ttytype` if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an HP 2622. It always asks you what kind of terminal you are on, defaulting to 2622.

```
export TERM; TERM='tset - ?2622'
```

If the file `/etc/ttytype` is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
export TERM; TERM='tset - -m '>1200:2624'
```

FILES

`/etc/ttytype`	port-name to terminal-type mapping data base;
`/usr/lib/terminfo/?/*`	terminal information data base.

VARIABLES

SHELL	if `csh`, generate `csh` commands; otherwise generate `sh` (Bourne or POSIX shell) commands.
TERM	the (canonical) terminal name.

AUTHOR

`tset` was developed by the University of California, Berkeley.

SEE ALSO

csh(1), sh(1), stty(1), ttytype(4), environ(5).

NAME

tsm - Terminal Session Manager

SYNOPSIS

`tsm`

DESCRIPTION

`tsm` allows a user to interact with more than one shell or application (session) from a single terminal. Each session is bound to a virtual device emulating the physical terminal. The emulation includes maintaining display state, softkeys, and terminal modes for each session. The virtual device can be manipulated like the actual terminal by using `stty` and `ioctl` (see *stty*(1) and *ioctl*(2)). Additionally `tsm` supports cut and paste between sessions, and provides an interface for a local lp device. Each session has its own process group ID.

Definitions

A session is *current* if it is being displayed and is the recipient of keyboard input.

The *standard search path* is:

```
./
$HOME
$TSMPATH
/usr/tsm/
```

Configuration files and such are searched for in the order indicated and defined by these paths.

Commands

There are two methods of interacting with `tsm`: a pull-down menu, and a command line interface. The pull-down menu (when configured) can be activated from a session by pressing the **tsm menu hot key** (default is `^T`) and should be self explanatory. The command line interface can be activated by pressing the **tsm hot key** (default is `^W`) in a session. Pressing a "hot key" twice passes the "hot key" character to the session instead of activating `tsm` command or menu mode.

Commands to `tsm` generally have single character invocation, in some cases the user is prompted for more input. The following commands can be issued from the `tsm` prompt level:

`0-9` Pressing a number at the command prompt selects the session of the same number to become the current session.

`+` Select the next higher numbered session.

`-` Select the next lower numbered session.

`l` Select the last session.

`?` Display a help screen describing `tsm` commands.

`c` Copy (cut): Three types:

- Text (Lines including new-lines). This is the default. Select with `T` when cut prompt is displayed.
- String (Lines strung together with white space in place of new-lines). Select with `T` when cut prompt is displayed.
- Block (A rectangle). Select with `T` when cut prompt is displayed.

The user is prompted for the "cut extents". The extents are defined by using arrow keys or the keys `u`, `d`, `l`, and `r` to move the cursor as desired. Pressing the space bar aborts the cut operation. The selected text is placed in the **cut buffer** Trailing whitespace and character attribute information are ignored.

`p`
Paste: the contents of the **cut buffer** is echoed to the current session as if it were typed from the keyboard.

`r`
Run a program as a new session. The user is prompted for the program name.

`s`
Start a new session containing a shell.

o
Output the current display to a printer (screen dump). The print mechanism is specified in a file named `.tsmprint` searched for in the standard way. Character attribute information is ignored.

k
Load the softkeys of the current session from a file. To load TSM defaults, specify "file" `+`. To load terminal defaults, specify "file" `-`.

g
Same as `k` above but softkeys are loaded "globally" into all sessions.

x
Access extended tsm commands as described in the `tsm` reference manual or on the `tsm` help screen.

q
Quit `tsm`: `SIGHUP` is sent to all processes started under `tsm`, and `tsm` exits.

EXTERNAL INFLUENCES

In general `tsm` environment variables must be set prior to `tsm` invocation. **TSMLP** is the *lp*(1) name of a printer that gets its output redirected to the printer port of the terminal.

TSMTPATH specifies an alternate search path for tsm files.

TSMTERM specifies an alternate terminal information file to be used by `tsm` instead of that specified by TERM. **TSMHOTKEY** specifies an alternate `tsm` hotkey for invocation ot the `tsm` command line.

WARNINGS

Some operations are not supported on certain terminals.

AUTHOR

`tsm` was developed by Structured Software Solutions, Inc.

FILES

/usr/tsm/.tsm	tsm main configuration file (default). Copy to **$HOME** for user customization.
`/usr/tsm/.tsmkeys`	`tsm` softkey configuration file (default). Copy to **$HOME** for user customization.
`/usr/tsm/term/*`	terminal description files

SEE ALSO

tsm.info(1), tsm.command(1), tsm.lpadmin(3M), shl(1).

NAME

tsm.command - send commands to the Terminal Session Manager (TSM)

SYNOPSIS

`/usr/tsm/bin/tsm.command` *command*

DESCRIPTION

`tsm.command` is used to send a command string programmaticly to the Terminal Session Manager (TSM), as if the string were typed on the TSM command line. `tsm.command` fails unless it is run from inside a TSM session. Actions caused by `tsm.command` affect only the instance of TSM that `tsm.command` is run under. *command* can have any value that is a valid key sequence for the TSM command line. The sequence should not include the "hotkey" character that normally initiates the command line mode of TSM. The sequence should end at the point where TSM exits command mode. If it ends prematurely TSM behaves as though escape was pressed, which exits command mode, usually canceling the command. `\r` should be used to indicate a return key. If no arguments are given on the command line, the program prompts for input from the user. If a `^C` terminates the sequence, the remainder of the sequence is accepted from the user.

AUTHOR

`tsm.command` was developed by Structured Software Solutions, Inc.

SEE ALSO

tsm(1).

NAME

tsm.info - get Terminal Session Manager state information

SYNOPSIS

`/usr/tsm/bin/tsm.info` *request*

DESCRIPTION

`tsm.info` is used to obtain information about TSM. When run from inside a TSM session it returns valid information; otherwise it fails with a non-zero error code. Information returned is written to standard output. *request* can have any of the following values:

`is_a_window`	Succesful (returns zero) if executed from a TSM session, non-zero error code otherwise.
`session_number`	Writes the session number of the session the *tsm.info* command is executed in.
`current_session_number`	Writes the session_number of the TSM session the user currently has active.
`active_session_numbers`	Writes the session numbers (separated by whitespace) of all active sessions (sessions not idle).
`idle_session_numbers`	Writes the session numbers (separated by whitespace) of all idle sessions.
`program_name`	Writes the program name (as assigned in `.tsm` or with *tsm.command*) of the session the `tsm.info` command is executed in.
`program_name_n`	Writes the program name of session *n*.

AUTHOR

`tsm.info` was developed by Structured Software Solutions, Inc.

SEE ALSO

tsm(1)

NAME

tsort - topological sort

SYNOPSIS

`tsort` [*file*]

DESCRIPTION

`tsort` produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input text *file*. If no *file* is specified, the standard input is understood. `tsort` is generally used in conjunction with the `lorder` command to sort the objects to be installed in a library by `ar` (see *lorder*(1) and *ar*(1)).

The input consists of pairs of text items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

DIAGNOSTICS

`Odd data`

There is an odd number of fields in the input file.

WARNINGS

Libraries and object files cannot be `tsort`ed directly.

`tsort` uses a quadratic algorithm that is not considered worth fixing given its typical use of ordering a library archive file.

SEE ALSO

lorder(1).

STANDARDS CONFORMANCE

`tsort`: SVID2, XPG2, XPG3

NAME

tty, pty - get the name of the terminal

SYNOPSIS

`tty [-s]`

`pty [-s]`

DESCRIPTION

`tty` and `pty` print the path name of the user's terminal. The `-s` option inhibits printing of the terminal path name and any diagnostics, providing a means to test only the exit code.

RETURN VALUE

Exit status codes for `tty` are:

2	Invalid options were specified,
1	The standard input is not a terminal or pseudo-terminal,
0	The standard input is a terminal or pseudo-terminal.

Exit status codes for `pty` are:

2	Invalid options were specified,
1	The standard input is not a pseudo-terminal,
0	The standard input is a pseudo-terminal.

DIAGNOSTICS

`not a tty`
standard input is not a terminal or pseudo-terminal for `tty`.

`not a pty`
standard input is not a pseudo-terminal for `pty`.

AUTHOR

`tty` was developed by AT&T. `pty` was developed by HP.

STANDARDS CONFORMANCE

`tty`: SVID2, XPG2, XPG3, POSIX.2

NAME

ttytype - terminal identification program

SYNOPSIS

`ttytype` [`-apsv`] [`-t` *type*]

DESCRIPTION

`ttytype` automatically identifies the current terminal type by sending an identification request sequence to the terminal. This method works for local, modem, and remote terminal connections, as well as for the `hpterm` and `xterm` terminal emulators.

Once the terminal has been identified, `ttytype` prints the terminal's type to the standard output (see *terminfo*(4)). This string is usually used as the value for the `TERM` environment variable.

If `ttytype` is unable to determine the correct terminal type, it prompts the user for the correct terminal identification string.

Options

`ttytype` recognizes the following options:

`-a` Causes `ttytype` to return an ID of "unknown" instead of prompting for the terminal type if auto-identification fails. If this option is not present, `ttytype` interactively prompts the user for the terminal type if it is unable to determine the correct type automatically.

`-p` Causes `ttytype` to prompt for the terminal type before it sends the terminal identification request sequence. If the user responds with only a carriage return, `ttytype` proceeds with the automatic terminal identification process. Any other response is taken as the correct terminal type. Note that the `LINES` and `COLUMNS` variables are not set if the user manually enters a terminal type.

The `-p` option is normally used only for terminals that do not behave well when presented with `ttytype`'s terminal identification request sequence. It gives the user a chance to respond with the correct terminal type before any escape sequences are sent that could have an adverse effect on the terminal.

The `-a` option can be used in conjunction with the `-p` option. The `-a` option only inhibits interactive prompting after `ttytype` has failed to identify the terminal by other means.

`-s` Tells `ttytype` to print a series of shell commands to set the `TERM`, `LINES`, and `COLUMNS` environment variables to appropriate values. In addition, the variable `ERASE` is set to the two-character sequence representing the appropriate erase character for the terminal (DEL for ANSI terminals, backspace for all others). This two-character sequence can then be used as an argument to `stty` or `tset` (see *stty*(1) and *tset*(1)).

The `SHELL` environment variable is consulted to see which shell syntax to use for setting the environment variables. This output is normally used with a command of the form:

```
eval 'ttytype -s'
```

`-t` *type*

`ttytype` normally attempts identification of Wyse, ANSI and HP terminals. The `-t` *type* argument can be used to restrict the inquiry to that required for terminals of the specified type. The accepted types are `ansi`, `hp`, and `wyse`. Multiple `-t` options can be specified.

`-v`

Enable verbose messages to standard error.

EXAMPLES

`ttytype` is most commonly used as part of the login sequence. The following shell script fragment can be used during login shell initialization:

```
#
# If TERM is not set, see if our port is listed in /etc/ttytype.
# If /etc/ttytype doesn't have information for our port, run
```

```
    # ttytype(1) to try to determine the type of terminal we have.
    #
    # To have ttytype(1) prompt for the terminal type before trying
    # to automatically identify the terminal, add the "-p" option
    # to the "ttytype -s" command below.
    #
    if [ -z "$TERM" -o "$TERM" = network ]; then
        unset TERM
        eval 'tset -s -Q'
        if [ -z "$TERM" -o "$TERM" = unknown ]; then
          eval 'ttytype -s'
          tset -Q -e ${ERASE:-\^h} $TERM
        fi
    fi
```

NOTES

Use of the **-s** option is highly recommended because many terminals support variable-size displays. This option provides the only means for automatically configuring the user environment in such a manner that applications can handle these terminals correctly. Note that **LINES** and **COLUMNS** are not set if the **-p** option is used and the user manually enters a terminal type.

The following steps are performed in the order indicated when identifying a terminal:

1. **ttytype** tries the Wyse 30/50/60 id request sequence.
2. **ttytype** tries the standard ANSI id request sequence. If a response is received, it is converted to a string according to an internal table.
3. **ttytype** tries the HP id request sequence.
4. If none of the above steps succeed, **ttytype** prompts interactively for the correct terminal type unless the **-a** option has been given.

ttytype may skip one or more of the first three steps, depending on the presence of **-t** options.

The HP ID-request sequence can switch some ANSI terminals into an unexpected operating mode. Recovery from such a condition sometimes requires cycling power on the terminal. To avoid this problem, **ttytype** always sends the HP identification sequence last.

WARNINGS

The terminal identification sequences sent by **ttytype** can cause unexpected behavior on terminals other than the Wyse 30/50/60, standard ANSI or HP terminals. If you have such terminals in your configuration, use the **-t** or **-p** options to prevent **ttytype** from sending sequences that cause unexpected behavior.

DEPENDENCIES

Series 300/400

When identifying the Series 300/400 internal terminal emulator (ITE), **ttytype** must determine the **TERM** value based upon the number of lines on the display. For graphics cards of identical resolution, **ttytype** returns a value for **TERM** that is appropriate for the ITE but may not exactly match the card's product number. For example, an HP 98550 is identified as an HP 98548 because the ITE behaves the same for both cards.

AUTHOR

ttytype was developed by HP.

SEE ALSO

csh(1), ksh(1), sh(1), stty(1), ttytype(4), environ(5).

NAME

ul - do underlining

SYNOPSIS

`ul` [`-t` *terminal*] [`-i`] [*name* ...]

DESCRIPTION

`ul` reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable **TERM**. The `-t` option overrides the terminal type specified in the environment. The *terminfo*(4) file corresponding to **TERM** is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode, the standout mode is used instead. If the terminal can overstrike, or handles underlining automatically, `ul` degenerates to `cat`. If the terminal cannot underline, underlining is ignored.

The `-i` option causes `ul` to indicate underlining onto by a separate line containing appropriate dashes `-`; this is useful when you want to look at the underlining present in an `nroff` output stream on a CRT terminal.

WARNINGS

`nroff` usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

FILES

`/usr/lib/terminfo/?/*`	terminal capability files

AUTHOR

`ul` was developed by the University of California, Berkeley>

SEE ALSO

col(1), man(1), nroff(1).

NAME

umask - get or set the file mode creation mask

SYNOPSIS

Get Mask:

`umask [-S]`

Set Mask:

`umask` [*mask*]

DESCRIPTION

`umask` is a built-in shell command (executed within the shell process) that can be used to print the current file mode creation mask value or to set a new one.

When a new file is created, each bit in the file mode creation mask that is *set* causes the corresponding permission bit in the the file mode to be cleared (disabled); hence the term *mask*. Conversely, bits that are *cleared* in the mask allow the corresponding file mode bits to be enabled in newly created files.

Getting Current Mask Value

To print the current file mode creation mask value, use one of the commands:

`umask`	Print the octal value of the current file mode creation mask.
`umask -S`	Print the current file mode creation mask value using a symbolic format that resembles:

```
u=rwx,g=rwx,o=rwx
```

where zero, one, two, or three of the letters `r`, `w`, and `x` are printed for each category: `u` (user (or owner)), `g` (group), and `o` (other), depending on the current mask setting. Each `r`, `w`, or `x` in the output indicates that the corresponding permissions are allowed in newly created files.

Both command-line forms produce output that is a valid value for use as the *mask* argument in a subsequent `umask` command when setting a new mask value.

Note that the `-S` option is implemented *only* in the Korn shell (see *ksh*(1)) and in the POSIX shell (see *sh-posix*(1)). Other shells such as Bourne shell (see *sh-bourne*(1)) and C shell (see *csh*(1)) do not support the symbolic output form.

Setting A New Mask Value

To set a new file mode creation mask value for the current shell execution environment, use the command:

`umask` *mask*

where *mask* can be a symbolic value or a non-negative octal integer (obsolescent). The resulting interpreted mask affects the initial value of the file permission bits for subsequently created files.

Symbolic *mask* Value:
The `umask` command built into the POSIX and Korn shells accepts symbolic *mask* values (as well as the obsolescent octal form). Symbolic *mask* values are constructed in the same manner as when using the `chmod` command (see *chmod*(1)) by concatenating comma-separated strings of the form:

who operator permissions

constructed as follows:

who One of the characters or character combinations `u`, `g`, `o`, `ug`, `uo`, `go`, `ugo`, or `a` (short form for `ugo`). If *who* is not specified, the *operator* and *permissions* apply to all categories. If two or more *who* categories have the same mask value, they can be placed together by using `ug`, `uo`, `go`, `ugo`, or `a` as appropriate.

If *who* is omitted, *operator* and *permissions* apply to all categories (same as `a` or `ugo`).

operator One of the characters `+`, `-`, or `=`. An operator is required for each *who*. Multiple operations can be performed on a given *who* in a complete *mask* argument, such as when permissions are deleted from all or some categories then new permissions are added by another *mask* substring (see EXAMPLES below).

If no *permissions* are specified for a given *operator*, no change is made to the current file creation mode mask. If *who* is not specified, the operation applies to all categories (u, g, and o).

*operator*s are interpreted as follows:

+	Clear the file mode bits represented by the accompanying *who* and *permissions* values in the mask, thus enabling corresponding permissions in newly created files.
-	Set the file mode bits represented by the specified *who* and *permissions* values in the mask, thus denying corresponding permissions in newly created files.
=	Clear the file mode bits specified by the corresponding *who* and *permissions* values and set all others.

permissions
One of the characters or character combinations **r**, **w**, **x**, **rx**, **wx**, **rw**, or **rwx**, specifying read, write, and/or execute (search) permission for the corresponding *who* and *operator*.

If *permissions* is not specified, no change is made to the existing file creation mode mask for the corresponding *who*.

Octal *mask* Value (obsolescent):
The **umask** command built into the POSIX, Korn, C, and Bourne shells accepts a non-negative integer value for *mask*. For an *octal* integer *mask* operand, the file mode bits are set absolutely.

If the user file-creation mode *mask* value is specified as three octal digits, they refer to read/write/execute permissions for owner, group, and others, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, the command **umask 022** removes group and others write permission (files normally created with mode 777 become mode 755; files created with mode 666 become mode 644).

The following table gives the octal integers and the corresponding mode-bits:

User			Group			Other		
Symbolic	**Octal**	**Mode bit**	**Symbolic**	**Octal**	**Mode bit**	**Symbolic**	**Octal**	**Mode bit**
u+r	0400	S_IRUSR	g+r	0040	S_IRGRP	o+r	0004	S_IROTH
u+w	0200	S_IWUSR	g+w	0020	S_IWGRP	o+w	0002	S_IWOTH
u+x	0100	S_IXUSR	g+x	0010	S_IXGRP	o+x	0001	S_IXOTH

If *mask* is omitted, the current value of the mask is printed in octal. The first digit, a zero, specifies that the output is expressed in octal. **umask** writes a message to standard output that can be later used as a umask mask operand. The use of an operand obtained in this way is not obsolescent, even if it is an octal number.

If a mask operand is specified, there is no output written to standard output.

umask is recognized and executed by the shell.

Note that the file creation mode mask does not affect the set-user-ID, set-group-ID, or "sticky" bits.

Since **umask** affects the current shell execution environment, it is generally provided as a shell regular built-in. Moreover, different shells provide different implementations of **umask** as a built-in. Features of **umask** as described here may not be supported by all the shells. Refer to the manual entry for specific shells for differences.

If **umask** is called in a subshell or separate utility execution environment such as one of the following:

```
(umask 002)
nohup umask ...
find . -exec umask ...
```

it does not affect the file mode creation mask of the caller's environment.

RETURN VALUE

umask exits with one of the following values:

0	The file mode creation mask was successfully changed or no *mask* operand was supplied.
>0	An error occurred.

EXAMPLES

Symbolic-Mode:

Set the `umask` value to produce read, write, and execute permissions for the file's owner and read-only permission for all others (`-rwxr--r--`) on newly created files:

```
umask u=rwx,g=r,o=r
```

Set the `umask` value to produce read, and write permissions for the file's owner, read-only for others users in the same group, and no access to others (`-rw-r-----`):

```
umask a-rwx,u+rw,g+r
```

Octal-Mode:

Set the `umask` value to deny read, write, or execute permissions to everyone (`----------`):

```
umask 0777
```

SEE ALSO

chmod(1), csh(1), ksh(1), sh-posix(1), sh(1), chmod(2), creat(2), umask(2).

STANDARDS CONFORMANCE

`umask`: SVID2, XPG2, XPG3, POSIX.2

NAME

umodem - XMODEM-protocol file transfer program

SYNOPSIS

`umodem` [*options*] *files* ...
`umodem -c`

DESCRIPTION

`umodem` is a file transfer program that incorporates the well-known XMODEM protocol used on CP/M systems and on the HP 110 portable computer.

Options

`umodem` recognizes the following options and command-line arguments:

`-1`	(one) Employ TERM II FTP 1.
`-3`	Enable TERM FTP 3 (CP/M UG).
`-7`	Enable 7-bit transfer mask.
`-a`	Turn on ARPA Net flag.
`-c`	Enter command mode.
`-d`	Do not delete `umodem.log` before starting.
`-l`	(ell) Turn on entry logging.
`-m`	Allow overwriting of files.
`-p`	Print all messages.
`-r[t|b]`	Receive file. Specify `t` for text, or `b` for binary.
`-s[t|b]`	Send file. Specify `t` for text, or `b` for binary.
`-y`	Display file status only.
file	Name of file or files to be transferred.

EXAMPLES

Receive a text file:

`umodem -rt7` *file*

Receive a binary file:

`umodem -rb` *file*

Send a text file:

`umodem -st7` *file*

Send a binary file:

`umodem -sb` *file*

AUTHOR

`umodem` is in the public domain.

SEE ALSO

kermit(1), cu(1), uucp(1).

NAME

uname - print name of current HP-UX version

SYNOPSIS

uname [**-amnrsvil**]

uname [**-S** *nodename*]

DESCRIPTION

When used in the first form above, **uname** calls **uname()** and prints selected information about the current system as requested by the accompanying option arguments (see *uname*(2)). The second form calls **setuname()** and sets the system name in the **utsname** structure as explained below (see *uname*(2)).

Options

uname recognizes the options listed below where each option selects contents of the corresponding field in the structure defined by header file **<sys/utsname.h>** as indicated:

none	**uname** with no option is equivalent to **uname -s**.
-s	Print the (usually trademarked) name of the operating system. On standard HP-UX systems, this option always prints **HP-UX**.
-n	Print the nodename field contents of the **utsname** structure, the name by which the system is usually known in a UUCP network (see WARNINGS).
-r	Print the current release level of the HP-UX operating system.
-v	Print the current version level of the HP-UX operating system.
-m	Print the machine hardware model name.
-i	Print the machine identification number (system name if the machine identification number cannot be ascertained).
-l	Print the license level for this system.
-a	Print all the above information.
-S *nodename*	Change the system name in the nodename field of the **utsname** structure to *nodename*. *nodename* is restricted to **UTSLEN**−1 characters; **UTSLEN** is defined in **<sys/utsname.h>**. Only users having appropriate privileges can use this option. See WARNINGS below.

EXAMPLES

Executing the command **uname -a** on a local system produces output resembling the following:

```
HP-UX attila A.08.07 A 9000/720 17927181
```

Printed fields are interpreted as follows:

HP-UX	An HP-UX operating system is installed on this machine.
attila	UUPC network system name by which the system is known.
A.08.07	HP-UX operating system release and version identifier.
A	License level for this system. As explained in *uname*(2) this system has a two-user license.
9000/720	Computer is an HP 9000 Model 720.
17927181	Machine identification number.

WARNINGS

Many types of networking services are supported on HP-UX, each of which uses a different assigned system name and naming convention. To ensure predictable system behavior, it is essential that system names (also called host names or node names) be assigned in such a manner that they do not create conflicts when the various networking facilities interact with each other.

The system does not rely on a single system name in a specific location, partly because different services use dissimilar name formats as explained below. System names are assigned by using the **uname -S**,

hostname, and **nodename** commands. In addition, the system name used in the HP Clustered Environment (called the **cnode name**) is assigned in the cluster configuration file **/etc/clusterconf**. System names are assigned as follows:

Nodename	Command/File	Format	Used By
NetIPC name	**nodename** *name*	*foo*[.*a*[.*b*]]	NS Services and NetIPC
Internet name	**hostname** *name*	*foo*[.*x*.*y*.*z*...]	ARPA and NFS Services
UUCP name	**uname -S** *name*	*foo*	UUCP and related programs
cnode name	**/etc/clusterconf**	*foo*	Cluster server and clients

where *foo* represents the assigned system name (it is *strongly* recommended that *foo* be identical for all commands and locations) and the optional .*x*.*y*.*z* or .*a*.*b* follow the specified notation for the particular ARPA/NFS or NS/NetIPC environment.

Internet names are also frequently called hostnames or domain names (not to be confused with NFS domain names). Refer to *hostname*(5) for more information about Internet naming conventions.

Whenever the system name is changed in any file or by use of any of the above commands, it should also be changed in all other locations as well. Other files or commands in addition to those above (such as **/usr/lib/uucp/Permissions** if used to circumvent **uname**, for example) may contain or alter system names. To ensure correct operation, they should also use the same system name.

System names are normally assigned by the **/etc/rc** script at start-up, and should not be altered elsewhere.

SEE ALSO

hostname(1), nodename(1), gethostname(2), sethostname(2), uname(2), clusterconf(4), hostname(5).

STANDARDS CONFORMANCE

uname: SVID2, XPG2, XPG3, POSIX.2

NAME

unget - undo a previous get of an SCCS file

SYNOPSIS

`unget` [`-r` *SID*] [`-s`] [`-n`] *file* ...

DESCRIPTION

`unget` undoes the effect of a `get -e` done prior to creating the intended new delta. If *file* is a directory name, `unget` treats each file in the directory as a file to be processed, except that non-SCCS files and unreadable files are silently ignored. If `-` is specified for *file*, the standard input is read with each line being taken as the name of an SCCS file to be processed. Refer to *sact*(1), which describes how to determine what deltas are currently binding for an s-file.

Options

`unget` recognizes the following options and command-line arguments. Options and arguments apply independently to each named *file*.

`-r`*SID*	Uniquely identifies which delta is no longer wanted (this would have been specified by `get` as the "new delta"). This option is necessary only if two or more outstanding `get`s for editing on the same SCCS file were done by the same person (login name). `unget` prints a diagnostic message if the specified *SID* is ambiguous, or if it is required but not present on the command line (see *sact*(1)).
`-s`	Silent option. Suppress printing the intended delta's *SID* on the standard output.
`-n`	Retain the file deposited in the current directory by the previous `get`. Normally this file is removed by `unget`.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

Only the user who did the corresponding `get -e` can execute `unget`. any other user must either use `su` to change user ID to that user (see *su*(1)), or edit the p-file directly (which can be done either by the s-file owner or a user who has appropriate privileges).

FILES

p-file	See *delta*(1).
g-file	See *delta*(1).

SEE ALSO

delta(1), get(1), help(1), sact(1).

SCCS User's Guide, in *Programming on HP-UX*.

STANDARDS CONFORMANCE

`unget`: SVID2, XPG2, XPG3

NAME

unifdef - remove preprocessor lines

SYNOPSIS

`unifdef [-clt] [[-D` *sym* `] [-U` *sym* `] [-iD` *sym* `] [-iU` *sym* `]] ...` [*file*]

DESCRIPTION

unifdef simulates some of the actions of **cpp** in interpreting C language preprocessor command lines (see *cpp*(1)). For **unifdef**, a valid preprocessor command line contains as its first character a **#** and one of the following keywords: **ifdef**, **ifndef**, **if**, **else**, or **endif**. The **#** character and its associated keyword must appear on the same line, but they can be separated by spaces, tabs, and commented text. When appropriate, the portions of code surrounded by and including the targeted preprocessor directives are removed, and the resultant text is written to the standard output.

Unlike **cpp**, **unifdef** does not insert included files, interpret macros, or strip comment lines. This means, among other things, that **#define** and **#undef** macros occurring within the input text are not interpreted.

Since **unifdef** is language-independent, it can be used for processing source files for languages other than the C language. For example, **unifdef** can be used on FORTRAN language source files, provided the C language preprocessor commands are used.

Options

unifdef recognizes the following command-line options:

-c	Complement the normal behavior by printing only the rejected lines.
-iD*sym*	Ignore text delimited by **#ifdef** *sym*. In other words, text that would otherwise be affected by some action is not touched when found within the context of a preprocessor command using *sym*.
-iU*sym*	Ignore text delimited by **#ifndef** *sym*.
-l	Replace rejected lines with blank lines in the text written to the standard output.
-t	Treat the input source as plain text. C-language comment and quoting constructs are not recognized.
-D*sym*	Define symbol *sym*.
-U*sym*	Cause symbol *sym* to be undefined.

EXAMPLES

Assume file **foo.f** contains the following:

```
      PROGRAM TEST1
      INTEGER I, J
#ifdef ANSI77
      DO I=1,10
#else
      DO 100 I=1,10
#endif
      J=J+1
#if defined (DEBUG) || defined (TEST)
      PRINT *,J
#endif
#ifdef ANSI77
      ENDDO
#else
          100   CONTINUE
#endif
      END
```

The command sequence:

```
unifdef -DANSI77 -UDEBUG -DTEST foo.f > /tmp/foo.f
```

produces the following result in file `/tmp/foo.f`:

```
PROGRAM TEST1
INTEGER I, J
DO I=1,10
J=J+1
PRINT *,J
ENDDO
END
```

AUTHOR

`unifdef` was developed in the public domain.

SEE ALSO

cpp(1).

NAME

uniq - report repeated lines in a file

SYNOPSIS

`uniq` [`-udc` [`-f` *fields*] [`-s` *chars*] [*input_file* [*output_file*]]

DESCRIPTION

`uniq` reads the input text file *input_file*, comparing adjacent lines, and copies the result to *output_file*. If *input_file* is not specified, the standard input and standard output are used. If *input_file* is specified, but *output_file* is not, results are printed to standard output. *input_file* and *output_file* must not be the same file.

Line-Comparison Options

`uniq` recognizes the following options when comparing adjacent lines:

- `-u` Print *only* those lines that are *not* repeated in the original file.
- `-d` Print *one* copy only of each repeated line in the input file.
- `-c` Generate an output report in default style except that each line is preceded by a count of the number of times it occurred. If this option is specified, the `-u` and `-d` options are ignored if either or both are also present.

If none of the options `u`, `d`, or `c` are present, `uniq` prints the results of the union of the `-u` and `-d` options, producing a copy of the original input file with the second and succeeding copies of any repeated lines removed. (Note that repeated lines must be adjacent in order to be found — see *sort*(1)).

Field-Skip Options

Two options are provided for skipping an initial portion of each line when making comparisons:

- `-f` *fields* Ignore the first *fields* fields, together with any blanks before each. *fields* is a positive decimal integer. A field is defined as a string of non-space, non-tab characters separated by tabs and/or spaces from its neighbors.
- `-s` *chars* Ignore the first *chars* characters. *chars* is a positive decimal integer. Fields are skipped before characters.

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` must be equal to the value it had when the input files were sorted.

`LC_CTYPE` defines a space character when the `-f` or `-s` option is used.

LANG determines the language in which messages are displayed.

If `LC_COLLATE` or `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `uniq` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occured.

SEE ALSO

comm(1), sort(1).

STANDARDS CONFORMANCE

`uniq`: SVID2, XPG2, XPG3, POSIX.2

NAME

units - conversion program

SYNOPSIS

`units` [- *file*]

DESCRIPTION

`units` converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively as follows:

System Prompt	User Response
`You have:`	`inch`
`You want:`	`cm`

The system responds with two factors; one used if multiplying (preceded by `*`), the other if dividing (preceded by `/`):

```
* 2.540000e+00
/ 3.937008e-01
```

After providing the conversion factors, `units` prompts for the next set of values. To terminate `units`, press the interrupt character as defined for your login.

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, and division by the usual sign:

System Prompt	User Response
`You have:`	`15 lbs force/in2`
`You want:`	`atm`

The system responds with:

```
* 1.020689e+00
/ 9.797299e-01
```

`units` only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

`pi`	ratio of circumference to diameter
`c`	speed of light
`e`	charge on an electron
`g`	acceleration of gravity
`force`	same as `g`,
`mole`	Avogadro's number,
`water`	pressure head per unit height of water,
`au`	astronomical unit.

Units must be provided in lowercase only. `pound` is not recognized as a unit of mass; `lb` is. Compound names are run together, (e.g., `lightyear`). British units that differ from their U.S. counterparts are prefixed thus: `brgallon`. For a complete list of units, examine the file:

```
/usr/lib/unittab
```

An alternate unit database file can be specified for use with the - *file* option. `units` looks in this file rather than the default `/usr/lib/unittab` for the table of conversions. This must be in the same format as `/usr/lib/unittab`. This is useful in defining your own units and conversions.

WARNINGS

The monetary exchange rates are out of date.

FILES

`/usr/lib/unittab`

NAME
uptime - show how long system has been up

SYNOPSIS
`uptime`

DESCRIPTION
`uptime` prints the current time, the length of time the system has been up, the number of users logged on to the system, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes.

EXAMPLES
The command:

```
uptime
```

produces text resembling the following:

```
2:30pm  up 14days, 2:39, 33 users,  load average: 1.71, 1.88, 1.80
```

depending upon the current status of the system.

AUTHOR
`uptime` was developed by the University of California, Berkeley.

NAME

users - compact list of users who are on the system

SYNOPSIS

`users [-c]`

DESCRIPTION

`users` lists the login names of the users currently on the system in a compact, one-line format. In the HP Clustered environment, the `-c` option can be used to list the login names of all users in the cluster (see *glossary*(9)).

The login names are sorted in ascending collation order (see Environment Variables below).

EXTERNAL INFLUENCES

Environment Variables

`LC_COLLATE` determines the order in which the output is sorted.

If `LC_COLLATE` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, *users* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

AUTHOR

`users` was developed by the University of California, Berkeley and HP.

FILES

`/etc/utmp`

SEE ALSO

who(1).

NAME

uucp, uulog, uuname - UNIX system to UNIX system copy

SYNOPSIS

uucp [*options*] *source_files destination_file*

uulog -f *system* [**-x**] [*-number*]

uulog [**-s** *system*] ... [**-x**] [*-number*]

uuname [**-l**]

DESCRIPTION

uucp

uucp copies files named by the *source_files* argument to the destination identified by the *destination_file* argument. When copying files to or from a remote system, *source_files* and *destination_file* can be a path name on the local system, or have the form:

system_name **!** *path_name*

where *system_name* is the name of a remote system in a list of system names known to **uucp**. When copying files to (but not from) a remote system, *system_name* can also be a chained list of remote system names such as:

system_name **!** *system_name* **!** ... **!** *system_name* **!** *path_name*

in which case an attempt is made to send the file, via the specified route, to the destination. Care should be taken to ensure that intermediate nodes in the route are configured to forward information (see WARNINGS below for restrictions).

The shell metacharacters **?**, ***** and **[. . .]** appearing in *path_name* are expanded on the appropriate system.

path_name can be one of:

- A full path name.
- A path name preceded by *~user* where *user* is a login name on the specified system and *~user* is replaced by that user's login directory. (If an invalid login is specified, the default public directory (**/usr/spool/uucppublic**) is used instead.
- A path name preceded by *~/destination* where *destination* is appended to **/usr/spool/uucppublic**.

  NOTE: This destination is treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that *destination* is a directory, append a **/** to the destination argument. For example, **~/dan/** as the destination argument causes directory **/usr/spool/uucppublic/dan** to be created if it does not already exist, and places the requested file or files in that directory.
- Anything else is prefixed by the current directory.

If an erroneous path name is specified for the remote system, the copy fails. If *destination_file* is a directory, the file-name part of the *source_file* argument is used.

uucp preserves execute permissions across the transmission and sets read and write permissions to 0666 (see *chmod*(2) and Access Control Lists below).

Options

uucp recognizes the following options:

-c	Do not copy local file to the spool directory for transfer to the remote machine (default).
-C	Force the copy of local files to the spool directory for transfer.
-d	Make all necessary directories for the file copy (default).
-f	Do not make intermediate directories for the file copy.

`-g`*grade*	*grade* is a single letter or number. A lower ASCII sequence value for *grade* causes the job to be transmitted earlier in a given conversation between systems.
`-j`	Output the ASCII job identification string on standard output. This job identification can be used by `uustat` to obtain the status or terminate a job (see *uustat*(1)).
`-m`*file*	Send mail to the requester when the copy is completed.
`-n`*user*	Notify *user* on the remote system that a file was sent.
`-r`	Do not start the file transfer; just queue the job.
`-s`*file*	Report status of the transfer to *file*. Note that *file* must be a full path name.
`-x`*debug_level*	Produce debugging on standard output. *debug_level* is a number between 0 and 9; higher numbers give more information.

uulog

`uulog` queries a log file of `uucp` transactions in a file `/usr/spool/uucp/.Log/uucico/`*system*.

The following options cause `uulog` to print logging information:

`-s` *system*	Print information about work involving *system*.
`-f` *system*	Do a `tail -f` (see *tail*(1)) of the file transfer log for *system*.

Other options used in conjunction with the `-s` and `-f` options above are:

`-x`	Search for the given system in the `/usr/spool/uucp/.Log/uuxqt/`*system* file instead of in the *uucico* log file.
`-`*number*	Do a *tail*(1) command of *number* lines.

uuname

`uuname` lists the `uucp` names of known systems. `uuname -l` returns the local system's default name.

Access Control Lists (ACLs)

A file's optional ACL entries are not preserved across `uucp` transmission. Instead, new files have a summary of the access modes (as returned in `st_mode` by `stat()`; see *stat*(2)).

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings displayed by `uucp` and `uulog` commands.

`LANG` determines the language in which messages are displayed by `uucp` and `uuname` commands.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `uucp`, `uulog`, and `uuname` behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. In most cases, you cannot fetch files by path name from a remote system. Ask a responsible person on the remote system to send them to you. For the same reasons, you probably cannot send files to arbitrary path names. As distributed, remotely accessible files are those whose names begin `/usr/spool/uucppublic` (equivalent to `~/`).

All files received by `uucp` are owned by `uucp`.

The `-m` option only works when sending files or when receiving a single file. Receiving multiple files specified by special shell characters `? * [...]` does not activate the `-m` option.

Protected files and files in protected directories owned by the requester can be sent by `uucp`. However, if the requester is root and the directory is not searchable by `other` or the file is not readable by `other`, the request fails.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system. For this reason, UUCP uses the cluster server cnode ID instead of the system name defined by `uname -S`, which makes the use of `uname -S` ineffective when changing system names for UUCP-based communications (see *clusterconf*(4)). See the discussion of the **Permissions** file **MYNAME** parameter in the *Remote Access User's Guide*.

DEPENDENCIES

Series 300/400

The following options to **uucp** are not currently supported: **-s**, **-x**.

The following options to *uulog* are not currently supported: **-f***system*, **-x**, and -*number*.

FILES

`/usr/lib/uucp/*`	other data and program files
`/usr/spool/uucp`	spool directories
`/usr/spool/uucppublic`	public directory for receiving and sending

SEE ALSO

mail(1), uux(1), chmod(2), stat(2), clusterconf(4), acl(5).

UUCP tutorial in *Remote Access Users Guide.*

STANDARDS CONFORMANCE

uucp: SVID2, XPG2, XPG3

uulog: SVID2, XPG2, XPG3

uuname: SVID2, XPG2, XPG3

NAME

uuencode, uudecode - encode/decode a binary file for transmission by mailer

SYNOPSIS

`uuencode` [*source*] *remotedest*
`uudecode` [*file*]

DESCRIPTION

`uuencode` and `uudecode` can be used to send a binary file to another machine by means of such services as `elm`(1), `mailx`(1), or `uucp`(1) (see *elm*(1), *mailx*(1), and *uucp*(1)).

`uuencode` takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, includes the original mode of the input file, and preserves the value of the remotedest argument which is the intended name for the file when it is restored later on the remote system.

`uudecode` reads an encoded file, ignores any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The encoded file is an ordinary ASCII text file and can be edited with any text editor to change the mode or remote name.

EXAMPLES

To encode and send a compiled program `foo` to user `friend`:

```
uuencode foo foo | mailx -s 'new program' friend
```

After receiving the mail message, user `friend` can decode the program by first saving the message in a file `foo.mail` and executing the command:

```
uudecode foo.mail
```

WARNINGS

The file is expanded by 35% (three bytes become four plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking `uudecode` (often `uucp`) must have write permission for the specified file.

If an encoded file has the same name as the destination name specified in it, `uudecode` starts overwriting the encoded file before decoding is completed.

SEE ALSO

elm(1), mail(1), mailx(1), shar(1), uucp(1), uux(1), uuencode(5)

NAME

uupath, mkuupath - access and manage the pathalias database

SYNOPSIS

`uupath` [`-f` *pathsfile*] *mailaddress*

`mkuupath` [`-v`] *pathsfile*

DESCRIPTION

`uupath` provides electronic message routing by expanding a simple UUCP address into a full UUCP path (see *uucp*(1)). For example, *host*`!`*user* could be expanded into *hostA*`!`*hostB* `!`*host*`!`*user*.

`uupath` expands an address by parsing *mailaddress* for the dominant host (see below) and looking up the host in the appropriate `pathalias` database (see *pathalias*(1)). If the host is found in the database, the expanded address is written to the standard output. If the host is not found, `uupath` writes the original address to the standard output and returns an exit status of 1. `uupath` expects *mailaddress* to be in UUCP format (*host*`!`...`!`*hostZ*`!`*user*) or ARPANET format (*user@host*).

The `-f` option opens the `pathalias` database based on *pathsfile* rather than the default database based on `/usr/lib/mail/paths`. This database must be a database created by `mkuupath`, consisting of the two files *pathsfile*`.dir` and *pathsfile*`.pag`.

The dominant host is the left-most UUCP host in *mailaddress*. If no UUCP host is found (no `!` is in the address), `uupath` assumes that the address is in the simple ARPANET format *user@host*. If the address does not match either format, `uupath` writes the original address to the standard output and returns an exit status of 1.

`mkuupath` constructs a mail routing database by using the *pathsfile* data file obtained from `pathalias` (see *pathalias*(1)). as input. The recommended *pathsfile* location is `/usr/lib/mail/paths`, because this is the default database used by `uupath`. The database files *pathsfile*`.dir` and *pathsfile*`.pag` are created by `mkuupath`. If these files already exist, they must be removed prior to running `mkuupath`.

The `-v` option specifies verbose mode, which writes a line to the standard output for each entry written to the database.

DIAGNOSTICS

`uupath` returns an exit status of 1 and writes the original *mailaddress* to the standard output if the address is not found or is incorrectly formatted. `uupath` returns an exit status of 2 and prints a diagnostic message if the database files are not accessible, or if improper parameters are given. Otherwise, `uupath` returns an exit status of 0.

If the database files *pathsfile*`.dir` and *pathsfile*`.pag` already exist prior to running `mkuupath`, the message `mkuupath:` *pathsfile*`.dir: File exists` is displayed. These files must be removed before running `mkuupath`.

AUTHOR

`uupath` was developed by University of California, Berkeley.

FILES

```
/usr/lib/mail/paths
/usr/lib/mail/paths.dir
/usr/lib/mail/paths.pag
```

SEE ALSO

pathalias(1), uucp(1).

NAME

uustat - uucp status inquiry and job control

SYNOPSIS

```
uustat -a
uustat -m
uustat -p
uustat -q
uustat -kjobid ]
uustat -rjobid ]
uustat [-ssys ] [-uuser ]
```

DESCRIPTION

`uustat` displays the status of, or cancels, previously specified `uucp` commands, or provide general status on `uucp` connections to other systems (see *uucp*(1)). Only one of the following options can be specified with `uustat` per command execution:

- `-a` Output all jobs in queue.
- `-m` Report the status of accessibility of all machines.
- `-p` Execute a `ps -flp` for all the process IDs that are in the lock files.
- `-q` List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in `( )` next to the number of `C` or `X` files it is the age in days of the oldest `C.` or `X.` file for that system. The Retry field is the number of hours until the next possible call. The Count field is the number of failure attempts. Note that for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real time to execute. As an example of the output produced by `uustat -q` :

```
eagle  3C 04/07-11:07 NO DEVICES AVAILABLE
mh3bs3 2C 07/07-10:42 SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file can have zero or more files to be sent (a command file with no files to be sent causes the `uucp` system to call the remote system and see if work is waiting). The date and time refer to the previous interaction with the system followed by the status of interaction.

`-k` *jobid*
Kill the `uucp` request whose job identification is *jobid*. The killed `uucp` request must belong to the person issuing the `uustat` command unless the command is executed by the super-user.

`-r` *jobid*
Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the cleanup daemon.

The following options can be used singly or together but cannot be used with the options listed above:

- `-s` *sys* Report the status of all `uucp` requests for remote system *sys*.
- `-u` *user* Report the status of all `uucp` requests issued by *user*.

Output for both the `-s` and `-u` options has the following format:

```
eaglen0000   4/07-11:01:03   (POLL)
eagleN1bd7   4/07-11:07      S        eagle dan 522 /usr/dan/A
eagleC1bd8   4/07-11:07      S        eagle dan 59 D.3b2a12cd4924
             4/07-11:07      S        eagle dan rmail mike
```

With the `-s` and `-u` options, the first field is the *jobid* of the job. This is followed by the date and time. The next field is either an `S` or `R`, depending on whether the job is to send or request a file. The next field is the destination system name. This is followed by the user ID of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution the name of the command (such as `rmail` which is the command used for remote mail). When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (such as `D.3b2alce4924`) that is created for data files associated with remote execution (`rmail` in this example).

When no options are given, `uustat` outputs the status of all `uucp` requests issued by the current user. The format used is the same as with the `-s` or `-u` options.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings.

LANG determines the language in which messages are displayed.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `uustat` behaves as if all internationalization variables are set to "C". See *environ*(5).

FILES

`/usr/spool/uucp/*`	spool directories

SEE ALSO

uucp(1).

UUCP tutorial in *Remote Access Users Guide*.

STANDARDS CONFORMANCE

`uustat`: SVID2, XPG2, XPG3

NAME

uuto, uupick - public UNIX system to UNIX system file copy

SYNOPSIS

`uuto` [*options*] *source-files destination*
`uupick` [`-s` *system*]

DESCRIPTION

`uuto` sends *source-files* to *destination*. `uuto` uses the `uucp` facility to send files (see *uucp*(1)), while allowing the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system ! *user*

where *system* is taken from a list of system names that `uucp` knows about (see `uuname` in *uucp*(1) manual entry). *user* is the login name of someone on the specified system.

`uuto` recognizes the following options:

`-p`	Copy the source file into the spool directory immediately, and send the copy.
`-m`	Send mail to the requester when the copy is complete.

The files (or sub-trees if directories are specified) are sent to *PUBDIR* on *system*, where *PUBDIR* is the UUCP public directory (`/usr/spool/uucppublic`). Specifically the files are sent to

PUBDIR`/receive/`*user*`/`*mysystem*`/files.`

The recipient is notified by electronic mail when the files arrive.

`uupick` accepts or rejects the files transmitted to the recipient. Specifically, `uupick` searches *PUBDIR* for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

`from` *system* `:` [`file` *file-name*] [`dir` *dirname*] `?`

`uupick` then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
`d`	Delete the entry.
`m` [*dir*]	Move the entry to named directory *dir* (current directory is default). Note that, if the current working directory is desired for *dir*, do *not* specify any parameter with `m`. A construction such as `m.` is erroneous, and results in loss of data.
`a` [*dir*]	Same as `m` except move all the files sent from *system*.
`p`	Print the contents of the file.
`q`	Stop.
EOT	(control-D) Same as `q`.
`!`*command*	Escape to the shell to execute *command*.
`*`	Print a command summary.

`uupick` invoked with the `-s`*system* option searches only the *PUBDIR* for files sent from *system*.

WARNINGS

To send files that begin with a dot (such as `.profile`) the filename must contain a corresponding dot. For example: `.profile`, `.prof*`, and `.profil?` are correct, whereas `*prof*` and `?profile` are incorrect.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

FILES

PUBDIR	`/usr/spool/uucppublic`	`public directory`

SEE ALSO

mail(1), uuclean(1M), uucp(1), uustat(1), uux(1).

UUCP, tutorial in *Remote Access Users Guide* .

STANDARDS CONFORMANCE

uuto: SVID2, XPG2, XPG3

uupick: SVID2, XPG2, XPG3

NAME

uux - UNIX system to UNIX system command execution

SYNOPSIS

uux [*options*] *command-string*

DESCRIPTION

uux gathers zero or more files from various systems, executes a command on a specified system, then sends standard output to a file on a specified system. Note that, for security reasons, many installations limit the list of commands executable on behalf of an incoming request from **uux**. Many sites will permit little more than the receipt of mail (see *mail*(1), *mailx*(1), and *elm*(1)) via **uux**.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name* **!**. A null *system-name* is interpreted as the local system.

File names can be one of the following:

- A full path name;
- A path name preceded by *~xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory. Note that if an invalid login is specified, the default will be to the public directory (**/usr/spool/uucppublic**);
- A path name preceded by *~/destination* where *destination* is appended to **/usr/spool/uucppublic**.
- A simple file name (which is prefixed by the current directory). See *uucp*(1) for details.

For example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

gets files **file1** and **file2** from machines **usg** and **pwba**, and executes a *diff*(1) command, placing the results in **file.diff** in the local directory **/usr/spool/uucppublic**.

Any special shell characters such as **<**, **>**, **;**, or **|** should be quoted, either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux attempts to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!cut -f1 b!/usr/file \(c!/usr/file\)
```

gets **/usr/file** from system **b** and sends it to system **a**, performs a **cut** command on the file, and sends the result of the cut command to system **c**.

uux notifies you if the requested command on the remote system was disallowed. The list of commands allowed is specified in the **Permissions** file in **/usr/lib/uucp**. The response comes by remote mail from the remote machine.

uux recognizes the following *options*:

-	The standard input to **uux** is made the standard input to the *command-string*.
-a*name*	Use *name* as the user identification replacing the initiator user-ID (notification is returned to the user).
-b	Return whatever standard input was provided to the **uux** command if the exit status is non-zero.
-c	Do not copy the local file to the spool directory for transfer to the remote machine (default).
-C	Force the copy of local files to the spool directory for transfer.
-g*grade*	*grade* is a single letter/number; lower ASCII sequence characters cause the job to be transmitted earlier during a particular conversation.
-j	Output the *jobid* (the job identification ASCII string) on the standard output. This job identification can be used by **uustat** to obtain the status or terminate a job

(see *uustat*(1)).

-n	Do not notify the user if the command fails.
-r	Do not start the file transfer, just queue the job.
-s*file*	Report status of the transfer in *file*.
-x*debug_level*	Produce debugging output on standard output. The *debug_level* is a number between 0 and 9. The higher the number, the more detailed the information returned.
-z	Send success notification to user.

WARNINGS

Only the first command of a shell pipeline can have a *system-name* **!**. All other commands are executed on the system of the first command.

The use of the shell metacharacter ***** will probably not do what you want it to do. The shell tokens **<<** and **>>** are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the UUCP subsystem. All files required for the execution are put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the **uux** request. The following command does *not* work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command:

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

works (if `diff` is a permitted command).

Protected files and files that are in protected directories that are owned by the requester can be sent in commands using **uux**. However, if the requester is `root`, and the directory is not searchable by `other`, the request fails.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system. See the discussion of the `Permissions` file **MYNAME** parameter in the *Remote Access User's Guide*.

FILES

`/usr/spool/uucp`	spool directory
`/usr/spool/uucppublic`	public directory
`/usr/lib/uucp/*`	other data and programs

SEE ALSO

mail(1), uuclean(1M), uucp(1).

UUCP, tutorial in *Remote Access Users Guide*.

STANDARDS CONFORMANCE

uux: SVID2, XPG2, XPG3

NAME

vacation - return "I am not here" indication

SYNOPSIS

`vacation -i`
`vacation` [[`-a` *alias*] ...] *login*

DESCRIPTION

`vacation` returns a message to the sender of a message telling them that you are currently not reading your mail. The intended use is in a `.forward` file in `$HOME`. For example, your `.forward` file might have:

```
\eric, ~|vacation -a allman eric~
```

which would send messages to you (assuming your login name was `eric`) and reply to any messages for `eric` or `allman`. The `\` preceding `eric` is required to force direct delivery to `eric`'s mailbox and prevent an infinite loop through the `.forward` file.

No message is sent unless `login` or an `alias` supplied using the `-a` option is a substring of either the `To:` or `Cc:` headers of the mail. No messages from `???-REQUEST`, `Postmaster`, `UUCP`, `MAILER`, or `MAILER-DAEMON` are replied to, nor is a notification sent if a `Precedence: bulk` or `Precedence: junk` line is included in the mail headers. Only one message per week is sent to each unique sender (at each unique host system). The people who have sent you messages are recorded in a database in the files `.vacation.pag` and `.vacation.dir` in your home directory.

`vacation` expects a file `.vacation.msg`, in your home directory, containing a message to be sent back to each sender. It should be an entire message (including headers). For example, it might say:

```
From: eric@ucbmonet.Berkeley.EDU (Eric Allman)
Subject: I am on vacation
X-Delivered-By-The-Graces-Of: The vacation program
Precedence: bulk

I am on vacation until July 22. If you have something urgent,
please contact Joe Kalash <kalash@ucbingres.Berkeley.EDU>.
        --eric
```

Header lines in this file must be left-justified and must not be preceded by any other (including blank) lines (see *sendmail*(1M)). If there is no `.vacation.msg` file, `vacation` uses the file `/usr/lib/vacation.def`, if it exists. Otherwise, it logs an error.

`vacation` reads the first line from the standard input (the incoming mail message in the example `.forward` file above) for a UNIX-style `From` line to determine the sender. *sendmail*(1M) includes this `From` line automatically, and its absence indicates non-mail input.

Options

`-i` Initializes the vacation database files. This option should be used before modifying a `.forward` file.

`-a`*alias* Identifies another name that can legitimately appear in the `To:` line of the mail header instead of your login name. More than one `-a` option can be specified.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which error messages are printed.

DIAGNOSTICS

On error, `vacation` exits with a value from `<sysexits.h>` and causes `sendmail` to report an error back to the sender of the original message. Errors such as the absence of `.vacation.msg` or calling `vacation` with incorrect arguments, are logged using `syslogd` on the system where `vacation` actually runs (see *syslogd*(1M)). The `syslog` file (`/usr/adm/syslog` by default – see */etc/syslog.conf* and *syslogd*(1M) for customizations) should be inspected when `vacation` generates mailer error messages.

Remember that, in an HP Clustered Environment, mail is handled at the cluster server rather than on client cnodes. This means that `syslog` diagnostics appear in the cluster server's `syslog`; not the client's.

WARNINGS

Errors in the `.forward` file can lead to loss of mail and infinite mail loops.

Always send test mail to yourself after configuring `vacation` just to be sure that it is working properly. This is akin to checking telephone forwarding before leaving for an extended period, and can prevent loss of messages.

FILES

`$HOME/.vacation.dir`	database file
`$HOME/.vacation.msg`	message to send
`$HOME/.vacation.pag`	database file
`/usr/lib/vacation.def`	system-wide default header and message
`/etc/syslog.conf`	dictates where error messages are recorded

AUTHOR

`vacation` was developed by Eric Allman and the University of California, Berkeley.

SEE ALSO

sendmail(1M), syslog(1M), ndbm(3)

NAME

val - validate SCCS file

SYNOPSIS

val -

val [**-s**] [**-r** *SID*] [**-m** *name*] [**-y** *type*] [**-v**] *files*

DESCRIPTION

val reads one or more *files* to determine whether each file read is an SCCS file meeting the characteristics specified by the optional argument list. Command-line options can appear in any order, and are described below.

Options

val recognizes the following options and command-line arguments. The effects of each option apply independently to each specified *file*.

-s	Silent option. Suppress diagnostic messages normally generated on the standard output when an error is encountered while processing any specifed *file*.
-r*SID*	Check existence of revision *SID* in *file* where *SID* (*S*CCS *ID*entification string) is an SCCS delta number. *SID* is first checked to ensure that it is unambiguous and valid before checking *file*. For example, ***-r1** is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist; ***-r1.0** and ***-r1.1.0** are invalid because they have a zero suffix which never appears in a valid delta number.
-m*name*	*name* is compared with the SCCS **%M%** keyword in *file*.
-y*type*	*type* is compared with the SCCS **%Y%** keyword in *file*.
-v	Verbose option. Prints additional detailed diagnostic messages on the standard output for any corruption detected while processing each named *file*. The messages are intended for use with the information contained in *sccsfile*(4) when fixing the file.
file	One or more SCCS files to be processed. If **-** is used as a *file* argument, **val** reads the standard input until an end-of-file condition is encountered. Each line read is independently processed as if it were a command-line argument list.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

RETURN VALUE

The 8-bit code returned by **val** is a disjunction of the possible errors; i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

Bit	**Interpretation**
0	Missing *file* argument;
1	Unknown or duplicate option argument;
2	Corrupt SCCS file;
3	Cannot open *file* or *file* is not an SCCS file;
4	*SID* is invalid or ambiguous;
5	*SID* does not exist;
6	*%Y%* does not match **-y** *type* argument;
7	*%M%* does not match **-m** *name* argument;

Note that **val** can process two or more files on a given command line, and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned; a logical OR of the codes generated for each command line and file processed.

DIAGNOSTICS

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described earlier under RETURN VALUE. Use the **help** command for explanations (see *help*(1)).

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).

BUGS

`val` can process up to 50 files on a single command line. Any number above 50 produces a fatal error.

STANDARDS CONFORMANCE

`val`: SVID2, XPG2, XPG3

NAME

vc - version control

SYNOPSIS

`vc` [`-a`] [`-t`] [`-c` *char*] [`-s`] [*keyword*=*value* ... *keyword*=*value*]

DESCRIPTION

`vc` copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* can be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as `vc` command arguments.

A control statement is a single line beginning with a control character, except as modified by the `-t` option (see below). The default control character is colon (`:`), except as modified by the `-c` option (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines, and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or fewer alphanumeric characters; the first character must be alphabetic. A value is any ASCII string that can be created using `ed` (see *ed*(1)); a numeric value is an unsigned string of digits. Keyword values must not contain spaces or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The `-a` option (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character can be included in a value by preceding it with \. If a literal \ is desired, it too must be preceded by \.

Options

`vc` recognizes the following options and arguments:

`-a`	Replace keywords surrounded by control characters with their assigned value in *all* text lines and not just in `vc` statements.
`-t`	Ignore all characters from the beginning of a line up to and including the first *tab* character for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.
`-c`*char*	Specify a control character to be used in place of `:`.
`-s`	Silence warning messages (not errors) that are normally printed on the diagnostic output.

Version Control Statements

Version control statements occur in the following forms:

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An `asg` statement overrides the assignment for the corresponding keyword on the `vc` command line and all previous `asg`s for that keyword. Keywords declared, but not assigned values have null values.

```
:if condition
     .
     .
     .
:end
```

Used to skip lines of the standard input. If the condition is true, all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

```
<cond>    ::= [ "not" ] <or>
<or>      ::= <and> | <and> "|" <or>
<and>     ::= <exp> | <exp> "&" <and>
<exp>     ::= "(" <or> ")" | <value> <op> <value>
<op>      ::= "=" | "!=" | "<" | ">"
<value>   ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
`not`	allowed only immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (such as `: 012 > 12` is false). All other operators take strings as arguments (for example, `: 012 != 12` is true). The precedence of the operators (from highest to lowest) is:

```
= != > <      all of equal precedence
&
|
```

Parentheses can be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one space or tab.

::text
Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **-a** option.

:on
:off Turn on or off keyword replacement on all lines.

:ctl**char**
Change the control character to char.

:msg**message**
Prints the given message on the diagnostic output.

:err**message**
Prints the given message followed by:

```
ERROR: err statement on line... (915)
```

on the diagnostic output. `vc` halts execution and returns an exit code of 1.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of keywords, values, the control character assigned through `ctl` and within text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `vc` behaves as if all internationalization variables are set to "C". See *environ*(5).

RETURN VALUE

vc returns 0 on normal completion; 1 if an error occurs.

DIAGNOSTICS

Use *help*(1) for explanations.

SEE ALSO

ed(1), help(1).

NAME
vi - screen-oriented (visual) display editor

SYNOPSIS
vi [**-**] [**-v**] [**-r**] [**-R**] [**+***command*] [**-l**] [**-t** *tag*] [**-V**] [**-w***size*] [**-x**] [*file* ...]

view [**-**] [**-v**] [**-r**] [**-R**] [**+***command*] [**-l**] [**-t** *tag*] [**-V**] [**-w***size*] [**-x**] [*file* ...]

vedit [**-**] [**-v**] [**-r**] [**-R**] [**+***command*] [**-l**] [**-t** *tag*] [**-V**] [**-w***size*] [**-x**] [*file* ...]

DESCRIPTION
vi (visual) is a display-oriented text editor that is based on the underlying **ex** line editor (see **ex**(1)). It is possible to switch back and forth between the two, and to execute **ex** commands from within **vi**.

When using **vi**, the terminal screen acts as a window into the file being edited. Changes made to the file are reflected in the screen display; the position of the cursor on the screen indicates the position within the file.

The environment variable **TERM** must give the terminal type, and the terminal must be defined in the *terminfo*(4) database. As with **ex**, editor initialization scripts can be placed in the environment variable **EXINIT**, or in the file **.exrc** in the current or home directory.

The **view** invocation is identical to **vi** except that the **readonly** editor option is set.

vedit is intended for beginners. It sets the **report** editor option to **1**, and sets the **showmode** and **novice** editor options. Otherwise it is the same as **vi**. These settings help make it easier for some beginners to learn the editor.

Options
vi recognizes the following command-line options:

-	Suppress all interactive user feedback. This is useful in processing editor scripts.
-v	Invoke **vi** (this option is intended for use with **ex**, and has no effect on **vi**).
-r	Recover *file*(s) after an editor or system crash. If no *file* is specified, a list of all saved files is printed. You must be owner of the saved file in order to recover it (super-user cannot recover files owned by other users).
-R	Set "read-only" mode to prevent overwriting the file inadvertently.
+*command*	Begin editing by executing the specified **ex** search or positioning *command*.
-l	*lisp* mode; indents appropriately for *lisp* code; the **( )**, **{ }**, **[[**, and **]]** commands in **vi** are modified to have meaning for *lisp*.
-t *tag*	Edit the file containing *tag*, and position the editor at its definition (see the **tag** command in *ex*(1) and *ctags*(1)).
-V	Verbose mode; editor commands are displayed as they are executed when input from a **.exrc** file or a source file (see the **source** command in *ex*(1)).
-w*size*	Set the value of the **window** editor option to *size*.
-x	Encryption mode; the user is prompted for a key to allow for the creation or editing of an encrypted file (see the **crypt** command in *ex (1)).*

See *ex*(1) for the complete description of **ex**. Only the visual mode of the editor is described here.

When invoked, **vi** is in **command mode**; input mode is initiated by several commands used to insert or change text. In input mode, ESC (escape) is used to leave input mode; however, two consecutive ESC characters are required to leave input mode if the **doubleescape** editor option is set (see *ex*(1)). In command mode, ESC is used to cancel a partial command; the terminal bell sounds if the editor is not in input mode and there is no partially entered command.

The last (bottom) line of the screen is used to echo the input for search commands (**/** and **?**), **ex** commands (**:**), and system commands (**!**). It is also used to report errors or print other messages.

Receipt of **SIGINT** during text input or during the input of a command on the bottom line terminates the input (or cancels the command) and returns the editor to command mode. During command mode, **SIGINT** causes the bell to be sounded; in general the bell indicates an error (such as an unrecognized key).

Lines displayed on the screen containing only a ~ indicate that the last line above them is the last line of the file (the ~ lines are past the end of the file). Terminals with limited local intelligence might display lines on the screen marked with an @: these indicate space on the screen not corresponding to lines in the file. (These lines can be removed by entering a **^R**, forcing the editor to retype the screen without these holes.)

If the system crashes or `vi` aborts due to an internal error or unexpected signal, `vi` attempts to preserve the buffer if any unwritten changes were made. Use the `-r` command line option to retrieve the saved changes.

`vi` supports the **SIGWINCH** signal, and redraws the screen in response to window-size changes.

Command Summary

Most commands accept a preceding number as an argument, either to give a size or position (for display or movement commands), or as a repeat count (for commands that change text). For simplicity, this optional argument is referred to as *count* when its effect is described.

The following operators can be followed by a movement command to specify an extent of text to be affected: **c**, **d**, **y**, **<**, **>**, **!**, and **=**. The region specified begins at the current cursor position and ends just prior to the cursor position indicated by the move. If the command operates on lines only, all the lines that fall partly or wholly within this region are affected. Otherwise the exact marked region is affected.

In the following listing, control characters are indicated in the form **^X**, which represents `Ctrl-X`. White space is defined to be the characters space, tab, and alternative space. Alternative space is the first character of the *langinfo*(3C) **ALT_PUNCT** item for the language specified by the **LANG** environment variable (see *environ*(5)).

Unless otherwise specified, the commands are interpreted in command mode and have no special effect in input mode.

^B	Scroll backward to display the previous window of text. A *count* specifies the number of windows to go back. Two lines of overlap are kept if possible.
^D	Scroll forward a half-window of text. A *count* gives the number of (logical) lines to scroll, and is remembered for future **^D** and **^U** commands.
	In input mode, **^D** backs `shiftwidth` spaces over the indentation provided by `autoindent` or **^T**. White space inserted by **^T** at other than the beginning of a line cannot be backed over using **^D**. A **^** followed by a **^D** removes all indentation (for the current and subsequent input lines of the current input mode) until new indentation is established by inserting leading white space, either by direct input or by using **^T**.
^E	Scroll forward one line, leaving the cursor where it is if possible.
^F	Scroll forward to display the window of text following the current one. A *count* specifies the number of windows to advance. Two lines of overlap are kept if possible.
^G	Print the current file name and other information, including the number of lines and the current position (equivalent to the `ex` command `f`).
^H	Move one space to the left (stops at the left margin). A *count* specifies the number of spaces to back up (same as **h**).
	In input mode, **^H** returns the cursor to the last input character without erasing it.
^J	Move the cursor down one line in the same column. A *count* specifies the number of lines to move down (same as **^N** and `j`).
^L	Clear and redraw the screen (use when the screen is scrambled for any reason).
^M	Move to the first non-whitespace character in the next line. A *count* specifies the number of lines to advance.
^N	Same as **^J** and `j`.
^P	Move the cursor up one line in the same column. A *count* specifies the number of lines to move up (same as **k**).

^R	Redraw the current screen, eliminating the false lines marked with @ (which do not correspond to actual lines in the file).
^T	In input mode, `^T` inserts `shiftwidth` white space. If at the beginning of the line, this inserted space can only be backed over using `^D`.
^U	Scroll up a half-window of text. A *count* gives the number of (logical) lines to scroll, and is remembered for future `^D` and `^U` commands.
^V	In input mode, `^V` quotes the next character to permit the insertion of special characters (including ESC) into the file.
^W	In input mode, `^W` backs up one word; the deleted characters remain on the display.
^Y	Scroll backward one line, leaving the cursor where it is, if possible.
^ [	Cancel a partially formed command; `^ [` sounds the bell if there is no partially formed command.
	In input mode, `^ [` terminates input mode. However, two consecutive ESC characters are required to terminate input mode if the `doubleescape` editor option is set (see *ex*(1)).
	When entering a command on the bottom line of the screen (**ex** command line or search pattern with `\` or **?**), terminate input and execute command.
	On many terminals, `^ [` can be entered by pressing the ESC or ESCAPE key.
^\	Exit `vi` and enter *ex* command mode. If in input mode, terminate the input first.
^]	Take the word after the cursor as a tag and execute the `tag` editor command (see *ex*(1)).
^ ^	Return to the previous file (equivalent to `:ex #`).
space	Move one space to the right (stops at the end of the line). A *count* specifies the number of spaces to go forward (same as `l`).
erase	Erase, where *erase* is the user-designated erase character (see *stty*(1)). Same as ^H.
kill	Kill, where *kill* is the user-designated kill character (see *stty*(1)). In input mode, *kill* backs up to the beginning of the current input line without erasing the line from the screen display.
susp	Suspend the editor session and return to the calling shell, where *susp* is the user-designated process-control suspend character (see *stty*(1)). See *ex*(1) for more information on the `suspend` editor command.
!	An operator that passes specified lines from the buffer as standard input to the specified system command, and replaces those lines with the standard output from the command. The `!` is followed by a movement command specifying the lines to be passed (lines from the current position to the end of the movement) and then the command (terminated as usual by a return). A *count* preceding the `!` is passed on to the movement command after `!`.
	Doubling `!` and preceding it by *count* causes that many lines, starting with the current line, to be passed.
"	Use to precede a named buffer specification. There are named buffers `1` through `9` in which the editor places deleted text. The named buffers `a` through `z` are available to the user for saving deleted or yanked text; see also **y**, below.
$	Move to the end of the current line. A *count* specifies the number of lines to advance (for example, **2$** causes the cursor to advance to the end of the next line).
%	Move to the parenthesis or brace that matches the parenthesis or brace at the current cursor position.
&	Same as the *ex* command `&` (that is, `&` repeats the previous `substitute` command).
´	When followed by a `´`, `vi` returns to the previous context, placing the cursor at the beginning of the line. (The previous context is set whenever a non-relative move is made.) When followed by a letter **a-z**, returns to the line marked with that letter (see

the `m` command), at the first non-whitespace character in the line.

When used with an operator such as `d` to specify an extent of text, the operation takes place over complete lines (see also `` ` ``).

`` ` `` When followed by a `` ` ``, **vi** returns to the previous context, placing the cursor at the character position marked (the previous context is set whenever a non-relative move is made). When followed by a letter `a` `z`, returns to the line marked with that letter (see the `m` command), at the character position marked.

When used with an operator such as `d` to specify an extent of text, the operation takes place from the exact marked place to the current position within the line (see also `´`).

`[[` Back up to the previous section boundary. A section is defined by the value of the **sections** option. Lines that start with a formfeed (`^L` character) or `{` also stop `[[`.

If the option **lisp** is set, the cursor stops at each `(` at the beginning of a line.

`]]` Move forward to a section boundary (see `[[`).

`^` Move to the first non-whitespace position on the current line.

`(` Move backward to the beginning of a sentence. A sentence ends at a `.`, `!`, or `?` followed by either the end of a line or by two spaces. Any number of closing `)`, `]`, `"` and `'` characters can appear between the `.`, `!`, or `?` and the spaces or end of line. If a *count* is specified, the cursor moves back the specified number of sentences.

If the **lisp** option is set, the cursor moves to the beginning of a **lisp** *s*-expression. Sentences also begin at paragraph and section boundaries (see `{` and `[[`).

`)` Move forward to the beginning of a sentence. If a *count* is specified, the cursor advances the specified number of sentences (see `(`).

`{` Move back to the beginning of the preceding paragraph. A paragraph is defined by the value of the **paragraphs** option. A completely empty line and a section boundary (see `[[` above) are also interpreted as the beginning of a paragraph. If a *count* is specified, the cursor moves backward the specified number of paragraphs.

`}` Move forward to the beginning of the next paragraph. If a *count* is specified, the cursor advances the specified number of paragraphs (see `{`).

`|` Requires a preceding *count*; the cursor moves to the specified column of the current line (if possible).

`+` Move to the first non-whitespace character in the next line. If a *count* is specified, the cursor advances the specified number of lines (same as `^M`).

`,` The comma (`,`) performs the reverse action of the last `f`, `F`, `t`, or `T` command issued, by searching in the opposite direction on the current line. If a *count* is specified, the cursor repeats the search the specified number of times.

`-` The hyphen character (`-`) moves the cursor to the first non-whitespace character in the previous line. If a *count* is specified, the cursor moves back the specified number of times.

`_` The underscore character (`_`) moves the cursor to the first non-whitespace character in the current line. If a *count* is specified, the cursor advances the specified number of lines, with the current line being counted as the first line; no *count* or a *count* of 1 specifies the current line.

`.` Repeat the last command that changed the buffer. If a *count* is specified, the command is repeated the specified number of times.

`/` Read a string from the last line on the screen, interpret it as a regular expression, and scan forward for the next occurrence of a matching string. The search begins when the user types a carriage return to terminate the pattern; the search can be terminated by sending **SIGINT** (or the user-designated interrupt character).

When used with an operator to specify an extent of text, the defined region begins with the current cursor position and ends at the beginning of the matched string. Entire lines can be specified by giving an offset from the matched line (by using a closing `/` followed by a *+n* or *-n*).

`0`	Move to the first character on the current line (the `0` is not interpreted as a command when preceded by a non-zero digit).
`:`	The colon character (`:`) begins an **`ex`** command. The `:` and the entered command are echoed on the bottom line; the **`ex`** command is executed when the user types a carriage return.
`;`	Repeat the last single character find using `f`, `F`, `t`, or `T`. If a *count* is specified, the search is repeated the specified number of times.
`<`	An operator that shifts lines to the left by one `shiftwidth`. The `<` can be followed by a move to specify lines. A *count* is passed through to the move command. When repeated (`<<`), shifts the current line (or *count* lines starting at the current one).
`>`	An operator that shifts lines right one `shiftwidth` (see `<`).
`=`	If the `lisp` option is set, `=` reindents the specified lines, as if they were typed in with `lisp` and `autoindent` set. Can be preceded by a *count* to indicate how many lines to process, or followed by a move command for the same purpose.
`?`	Scan backwards, the reverse of `/` (see `/`).
@buffer	Execute the commands stored in the named *buffer*. Be careful not to include a <return> character at the end of the buffer contents unless the <return> is part of the command stream. Commands to be executed in *ex* mode should be preceded by a colon (`:`).
`~`	The tilde (`~`) switches the case of the character under the cursor (if it is a letter), then moves one character to the right, stopping at the end of the line). A *count* specifies how many characters from the current line are switched.
`A`	Append at the end of line (same as `$a`).
`B`	Back up one word, where a word is any non-blank sequence, placing the cursor at the beginning of the word. If a *count* is specified, the cursor moves back the specified number of words.
`C`	Change the rest of the text on the current line (same as `c$`).
`D`	Delete the rest of the text on the current line (same as `d$`).
`E`	Move forward to the end of a word, where a word is any non-blank sequence. If a *count* is specified, the cursor advances the specified number of words.
`F`	Must be followed by a single character; scans backwards in the current line, searching for that character and moving the cursor to it, if found. If a *count* is specified, the search is repeated the specified number of times.
`G`	Go to the line number given as preceding argument, or the end of the file if no preceding *count* is given.
`H`	Move the cursor to the top line on the screen. If a *count* is given, the cursor moves to *count* number of lines from the top of the screen. The cursor is placed on the first non-whitespace character on the line. If used as the target of an operator, entire lines are affected.
`I`	Insert at the beginning of a line (same as `^` followed by `i`).
`J`	Join the current line with the next one, supplying appropriate white space: one space between words, two spaces after a period, and no spaces at all if the first character of the next line is a closing parenthesis (`)`). A *count* causes the specified number of lines to be joined, instead of two.
`L`	Move the cursor to the first non-whitespace character of the last line on the screen. If a *count* is given, the cursor moves to *count* number of lines from the bottom of the screen.

	When used with an operator, entire lines are affected.
`M`	Move the cursor to the middle line on the screen, at the first non-whitespace position on the line.
`N`	Scan for the next match of the last pattern given to `/` or `?`, but in the opposite direction; this is the reverse of `n`.
`O`	Open a new line above the current line and enter input mode.
`P`	Put back (replace) the last deleted or yanked text before/above the cursor. Entire lines of text are returned above the cursor if entire lines were deleted or yanked. Otherwise, the text is inserted just before the cursor.
	The `P` can be preceded by a named buffer specification ("*x*), to retrieve the contents of the buffer.
`Q`	Exit **vi** and enter **ex** command mode.
`R`	Replace characters on the screen with characters entered, until the input is terminated with ESC.
`S`	Change entire lines (same as `cc`). A *count* changes the specified number of lines.
`T`	Must be followed by a single character; scan backwards in the current line for that character, and, if found, place the cursor just after that character. A *count* is equivalent to repeating the search the specified number of times.
`U`	Restore the current line to its state before the cursor was last moved to it.
`W`	Move forward to the beginning of a word in the current line, where a word is a sequence of non-blank characters. A *count* specifies the number of words to advance.
`X`	Delete the character before the cursor. A *count* repeats the effect, but only characters on the current line are deleted.
`Y`	Place (yank) a copy of the current line into the unnamed buffer (same as `yy`). If a *count* is specified, *count* lines are copied to the buffer. If the `Y` is preceded by a buffer name, the lines are copied to the named buffer.
`ZZ`	Exit the editor, writing out the buffer if it was changed since the last write (same as the **ex** command **x**). Note that if the last write was to a different file and no changes have occurred since, the editor exits without writing out the buffer.
`a`	Enter input mode, appending the entered text after the current cursor position. A *count* causes the inserted text to be replicated the specified number of times, but only if the inserted text is all on one line.
`b`	Back up to the previous beginning of a word in the current line. A word is a sequence of alphanumerics or a sequence of special characters. A *count* repeats the effect.
`c`	Must be followed by a movement command. Delete the specified region of text, and enter input mode to replace deleted text with new text. If more than part of a single line is affected, the deleted text is saved in the numeric buffers. If only part of the current line is affected, the last character deleted is marked with a `$`. A *count* is passed through to the move command. If the command is `cc`, the entire current line is changed.
`d`	Must be followed by a movement command. Delete the specified region of text. If more than part of a line is affected, the text is saved in the numeric buffers. A *count* is passed through to the move command. If the command is `dd`, the entire current line is deleted.
`e`	Move forward to the end of the next word, defined as for `b`. A *count* repeats the effect.
`f`	Must be followed by a single character; scan the rest of the current line for that character, and moves the cursor to it if found. A *count* repeats the action that many times.
`h`	Move the cursor one character to the left (same as `^H`). A *count* repeats the effect.
`i`	Enter input mode, inserting the entered text before the cursor (see `a`).

`j`	Move the cursor one line down in the same column (same as `^J` and `^N`).
`k`	Move the cursor one line up (same as `^P`).
`l`	Move the cursor one character to the right (same as `<space>`).
`m`	Must be followed by a single lowercase ASCII letter **x**; mark the current position of the cursor with that letter. The exact position on the marked line is referred to by `` ` ``*x*; the marked line is referred to by ´*x*.
`n`	Repeat the last `/` or `?` scanning commands.
`o`	Open a line below the current line and enter input mode; otherwise like `O`.
`p`	Put text after/below the cursor; otherwise like `P`.
`r`	Must be followed by a single character; the character under the cursor is replaced by the specified one. (The new character can be a new-line.) If `r` is preceded by a *count*, *count* characters are replaced by the specified character.
`s`	Delete the single character under the cursor and enter input mode; the entered text replaces the deleted character. A preceding *count* specifies how many characters on the current line are changed. The last character being changed is marked with a `$`, as for `c`.
`t`	Must be followed by a single character; scan the remainder of the line for that character. The cursor moves to the column prior to the character if the character is found. A *count* is equivalent to repeating the search *count* times.
`u`	Reverse the last change made to the current buffer. If repeated, `u` alternates between these two states; thus is its own inverse. When used after an insertion of text on more than one line, the lines are saved in the numerically named buffers.
`w`	Move forward to the beginning of the next word (where word is defined as in `b`). A *count* specifies how many words the cursor advances.
`x`	Delete the single character under the cursor. When `x` is preceded by a *count*, `x` deletes the specified number of characters forward from the cursor position, but only on the current line.
`y`	Must be followed by a movement command; the specified text is copied (yanked) into the unnamed temporary buffer. If preceded by a named buffer specification, "*x*, the text is placed in that buffer also. If the command is `yy`, the entire current line is yanked.
`z`	Redraw the screen with the current line placed as specified by the following options: `z`<return> specifies the top of the screen, `z.` the center of the screen, and `z-` the bottom of the screen. The commands `z^` and `z+` are similar to `^B` and `^F`, respectively. However, `z^` and `z+` do not attempt to maintain two lines of overlap. A *count* can be given after the `z` and before the following character to specify the number of lines displayed in the redrawn screen. A *count* before the `z` gives the number of the line to use as the reference line instead of the default current line.

Keyboard Editing Keys

At initialization, the editor automatically maps some terminal keyboard editing keys to equivalent visual mode commands. These mappings are only established for keys that are listed in the following table and defined in the *terminfo*(4) database as valid for the current terminal (as specified by the `TERM` environment variable).

Both command and input mode mappings are created (see the `map` command in *ex*(1)). With the exception of the `insert char` keys, which simply toggle input mode on and off, the input mode mappings exit input mode, perform the same action as the command mode mapping, and then reenter input mode.

On certain terminals, the character sequence sent by a keyboard editing key, which is then mapped to a visual mode command, can be the same character sequence a user might enter to perform another command or set of commands. This is most likely to happen with the input mode mappings; therefore, on these terminals, the input mode mappings are disabled by default. Users can override the disabling and enabling of both the command and input mode keyboard editing key mappings by setting the `keyboardedit` and `keyboardedit!` editor options as appropriate (see *ex*(1)). The `timeout`, `timeoutlen`, and `doubleescape` editor options are alternative methods of addressing this problem.

terminfo entry	**command mode map**	**input mode map**	**map name**	**description**
key_ic	i	^[	inschar	insert char
key_eic	i	^[	inschar	end insert char
key_up	k	^[ka	up	arrow up
key_down	j	^[ja	down	arrow down
key_left	h	^[ha	left	arrow left
key_right	l	^[la	right	arrow right
key_home	H	^[Ha	home	arrow home
key_il	o^[	^[o^[a	insline	insert line
key_dl	dd	^[dda	delline	delete line
key_clear	^L	^[^La	clear	clear screen
key_eol	d$	^[d$a	clreol	clear line
key_sf	^E	^[^Ea	scrollf	scroll down
key_dc	x	^[xa	delchar	delete char
key_npage	^F	^[^Fa	npage	next page
key_ppage	^B	^[^Ba	ppage	previous page
key_sr	^Y	^[^Ya	sr	scroll up
key_eos	dG	^[dGa	clreos	clear to end of screen

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions and in processing the *tags* file.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as uppercase or lowercase letters, the shifting of letters between uppercase and lowercase, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

LANGOPTS specifies options determining how text for right-to-left languages is stored in input and output files. See *environ*(5).

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, the editor behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

See warnings in *ex*(1).

Program Limits

vi places the following limits on files being edited:

Maximum Line Length

LINE_MAX characters (defined in `<limits.h>`), including 2-3 bytes for overhead. Thus, if LINE_MAX is 2048, containing up to 2044 characters should cause no problem.

If loading files containing lines longer than the stated limit, lines are truncated to the stated maximum length. If the file is stored back in the original file, truncated information is lost.

Attempting to create lines longer than allowable maximum from the editor produces a `line too long` error message.

Maximum File Size

Maximum file length of 250 000 lines is silently enforced.

Other limits:

- 256 characters per global command list.

- 128 characters in a filename in `vi` or `ex open` mode. On short-filename HP-UX systems, maximum filename length is 14 characters.
- 128 characters in previous insert/delete buffer.
- 100 characters in a shell-escape command.
- 63 characters in a string-valued option (`:set` command).
- 30 characters in a program tag name.
- 32 or fewer macros defined by `map` command.
- 512 or fewer characters total in combined `map` macros.

AUTHOR

`vi` was developed by the University of California, Berkeley. The 16-bit extensions to `vi` are based in part on software of the Toshiba Corporation.

SEE ALSO

ex(1), edit(1), expreserve(1), terminfo(4), environ(5), lang(5), regexp(5).

The Ultimate Guide to the vi and ex Text Editors, Benjamin Cummings Publishing Co., Inc., ISBN 0-8053-4460-8, HP stock number 97005-90015.

STANDARDS CONFORMANCE

`vi`: SVID2, XPG2, XPG3

NAME

vis, inv - make unprintable characters in a file visible or invisible

SYNOPSIS

vis [-n] [-s] [-t] [-u] [-x] *file* ...
inv [-n] [-s] [-t] [-u] [-x] *file* ...

DESCRIPTION

vis reads characters from each *file* in sequence and writes them to the standard output, converting those that are not printable into a visible form. *inv* performs the inverse function, reading printable characters from each *file*, returning them to non-printable form, if appropriate, then writing them to standard output;

Non-printable characters are represented using C-like escape conventions:

`\\`	backslash
`\b`	backspace
`\e`	escape
`\f`	form-feed
`\n`	new-line
`\r`	carriage return
`\s`	space
`\t`	horizontal tab
`\v`	vertical tab
`\`*n*	the 8-bit character whose ASCII code is the 3-digit octal number *n*.
`\x`*n*	the 8-bit character whose ASCII code is the 2-digit hexadecimal number *n*.

Space, horizontal-tab, and new-line characters can be treated as printable (and therefore passed unaltered to the output) or non-printable depending on the options selected. Backslash, although printable, is expanded by *vis*, to a pair of backslashes so that when they are passed back through *inv*, they convert back to a single backslash.

If no input file is given, or if the argument **-** is encountered, **vis** and *inv* read from the standard input.

Options

vis and **inv** recognize the following options:

-n Treat new-line, space, and horizontal tab as non-printable characters. **vis** expands them visibly as **\n**, **\s**, and **\t**, rather than passing them directly to the output. **inv** discards these characters, expecting only the printable expansions. New-line characters are inserted by **vis** every 16 characters so that the output will be in a form that is usable by most editors.

-s Make **vis** and **inv** silent about non-existent files, identical input and output, and write errors. Normally, no input file can be the same as the output file unless it is a special file.

-t Treat horizontal-tab and space characters as non-printable in the same manner that **-n** treats them.

-u Cause output to be unbuffered (character-by-character); normally, output is buffered.

-x Cause **vis** output to be in hexadecimal form rather than the default octal form. Either form is accepted to **inv** as input.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

AUTHOR

vis was developed by HP.

SEE ALSO

cat(1), echo(1), od(1).

WARNINGS

Redirecting output to an input file destroys the original data. Therefore, command forms such as

```
    vis file1 file2 >file1
```

should be avoided unless the source file can be safely discarded.

NAME

vmstat - report virtual memory statistics

SYNOPSIS

`vmstat` [[`-dnS`][*interval*[*count*]] | `-f` | `-s` | `-z`]

DESCRIPTION

`vmstat` normally reports certain statistics kept about process, virtual memory, trap, and CPU activity.

Options

`vmstat` recognizes the following options:

`-d`	Report disk transfer information in the form of transfers per second.
`-n`	Provide an output format that is more easily viewed on an 80-column display device. This format separates the report into two groups: virtual memory information and CPU data. Each group is displayed as a separate line of output. On multiprocessor systems, this display format also provides CPU utilization on a per CPU basis.
`-S`	Report the number of processes swapped in and out instead of page reclaims and address translation faults.
interval	Display successive lines which are summaries over the last *interval* seconds. The command `vmstat 5` prints what the system is doing every five seconds. This is a good choice of printing interval since this is how often some of the statistics are sampled in the system; others vary every second.
count	Repeat the summary statistics *count* times.
`-f`	Report on the number of `fork`s and the number of pages of virtual memory involved since boot-up.
`-s`	Print the contents of the `sum` structure, giving the total number of several kinds of paging-related events that have occurred since boot-up or since `vmstat -z` was last executed.
`-z`	Clear all accumulators in the `sum` structure. This requires write file access permission on `/dev/kmem`. This is normally restricted to users with appropriate privileges.

If none of these options are given, `vmstat` reports in the first line a summary of the virtual memory activity since boot-up or since `vmstat -z` was last executed. The column headings and the meanings of each column are:

`Procs:` Information about numbers of processes in various states.

`r`	in run queue
`b`	blocked for resources (i/o, paging, etc.)
`w`	runnable or short sleeper (< 20 secs) but swapped

`Memory:`

Information about the usage of virtual and real memory. Virtual pages are considered active if they belong to processes that are running or have run in the last 20 seconds.

`avm`	active virtual pages
`free`	size of the free list

`Page:`

Information about page faults and paging activity. These are averaged each five seconds, and given in units per second.

`re`	page reclaims
`at`	address translation faults
`pi`	pages paged in
`po`	pages paged out

`fr`	pages freed per second
`de`	anticipated short term memory shortfall
`sr`	pages scanned by clock algorithm, per-second

`Faults:`
Trap/interrupt rate averages per second over last 5 seconds.

`in`	(non clock) device interrupts per second
`sy`	system calls per second
`cs`	cpu context switch rate (switches/sec)

`Cpu:`
Breakdown of percentage usage of CPU time

`us`	user time for normal and low priority processes
`sy`	system time
`id`	CPU idle

DEPENDENCIES

Series 300/400

Disk transfer information is not yet reported for SCSI disks.

AUTHOR

`vmstat` was developed by the University of California, Berkeley, and HP.

FILES

`/dev/kmem`

SEE ALSO

iostat(1), privilege(5).

NAME

vt - log in on another system over lan

SYNOPSIS

`vt` *nodename* [*lan_device*]
`vt -p` [*lan_device*]

DESCRIPTION

`vt` enables a user to log in on another HP 9000 system (*nodename*) over an HP local area network. The `-p` option causes `vt` to send a poll request over the local area network to find out what systems currently have `vtdaemon` running (see *vtdaemon*(1M)). An asterisk (*) following a nodename in the response indicates that the system is a vt gateway. Plus signs (+) following the nodename indicate how many vt gateways must be traversed to reach that system.

The optional argument *lan_device* specifies a character special device name to use instead of the default device name to send and receive data to and from the local area network. The major number for this device must correspond to a CIO IEEE802.3 local area network device.

Once a connection has been established, `vt` enters input mode. In this mode, text typed is sent to the remote host. To issue `vt` commands when in input mode, precede them with the `vt` escape character. When in command mode, normal terminal editing conventions are available.

The connection should terminate automatically upon termination of the login shell on the remote machine. If the connection is not terminated upon exit, it is likely that the `ptydaemon` on the remote system has either been terminated or restarted. To terminate a vt connection, enter command mode and use the `quit` command to terminate the connection.

Commands

`vt` recognizes the following commands. Commands can be abbreviated by typing enough of the command to uniquely identify it.

`cd` *remote-directory*
: Change the working directory on the remote machine to *remote-directory*. This command is applicable for file transfer only.

`escape` [*escape-char*]
: Set the `vt` escape character. If a character is not specified `vt` prompts for one. To print or display the current `vt` escape character simply press Return in response to this prompt.

`help` or `?` Print a `vt` command summary.

`lcd` [*directory*]
: Change the local working directory. If no *directory* is specified, use the user's home directory. This command is applicable for file transfer and shell escapes only.

`get` *remote-file local-file*
`receive` *remote-file local-file*
: Copy *remote-file* to local machine and store as *local-file*. `vt` prompts for the file names if they are not specified. The file transfer can be aborted by typing the interrupt character or pressing the break key.

`put` *local-file remote-file*
`send` *local-file remote-file*
: Copy *local-file* to the remote machine and store as *remote-file*. `vt` prompts for the file names if they are not specified. The file transfer can be aborted by typing the interrupt character or pressing the break key.

`quit` Terminate the connection and exit *vt*.

`user`*user-name*[`:`[*password*]]
: Identify yourself to the remote `vt` server. `vt` prompts for a password (after disabling local echo) if a colon (`:`) is appended to *user-name*. This command must be executed before any file transfer command can be used.

`!` [*shellcommand*]
: Shell escape. The given command is passed to a sub-shell for execution. If no command is given, a shell is started and connected to the user's terminal.

Access Control Lists (ACLs)

When sending or receiving files using *vt*, optional ACL entries are removed. New files have a summary of the access modes (as returned in `st_mode` by `stat()` of the file being transferred (see *stat*(2)).

DIAGNOSTICS

The diagnostics produced by `vt` are intended to be self-explanatory.

WARNINGS

`vt` uses the Hewlett-Packard LLA (Link Level Access) direct interface to the HP network drivers. `vt` uses the multicast address **`0x01AABBCCBBAA`**. It should not be used or deleted by other applications accessing the network. `vt` uses the following IEEE 802.3 *sap* (service access point) values: **`0x90`**, **`0x94`**, **`0x98`**, `0x9C`, `0xA0`, `0xA4`, `0xA8`, `0xAC`, `0xB0`, `0xB4`, `0xB8`, `0xBC`, `0xC0`, `0xC4`, `0xC8`, `0xCC`, `0xD0`, and `0xD4`. They should not be used by other applications accessing the network.

When using `vt` on a system that has multiple LAN cards installed, the optional command-line argument *lan_device* may be required if the remote system is not accessible through the default LAN device. The appropriate *lan_device* is the one connected (either directly or by way of other gateways) to the remote system.

FILES

`/dev/ieee`	Default lan device name.

SEE ALSO

vtdaemon(1M), stat(2), lan(4), acl(5).

`vt` tutorial in *Remote Access Users Guide*.

NAME

vt3k - virtual terminal facility from HP-UX (HP 9000) to MPE (HP 3000).

SYNOPSIS

`vt3k` [[`-a`|`-I`] input_file] [`-l`[`i`][`o`] `-f` [*logfile*]] [`-t`] *hostname*

DESCRIPTION

`vt3k` provides direct login capability (via LAN) from HP-UX to MPE V or MPE XL. Invoking `vt3k` from HP-UX establishes a connection to an HP 3000 running NS Virtual Terminal Services. Upon successful connection, the HP-UX user receives an MPE logon prompt. When the user logs off the MPE system, the connection is automatically terminated and the user is returned to HP-UX.

Options

`vt3k` supports the following command-line options and arguments:

`-a` *input_file*	Read command input from *input_file* instead of standard input, and restore input back to standard input (keyboard) at the end of the script. The first line in *input_file* must be an MPE logon statement. `vt3k` returns input control to the keyboard after the last command in the input file. This is especially useful for an "automated" HP 3000 login and application invocation. The last line in *input_file* can also be an MPE `bye` command. This option cannot be used with the `-I` option.
`-I` *input_file*	Read command input from *input_file* instead of standard input. Equivalent to the `-a` option, except that the connection to the HP 3000 is terminated when the end of *input_file* is encountered, whether or not an MPE `bye` command exists in the last line of *input_file*. This option cannot be used with the `-a` option.
`-li`	Log input to file specified by the `-f` option.
`-lo`	Log output to file specified by the `-f` option.
`-lio`	Log input and output to file specified by the `-f` option.
`-f` *logfile*	Log input and/or output as specified by the `-l` option to *logfile*. If *logfile* is not specified, send logging information to standard output.
`-t`	Enable standard HP-UX type-ahead capability while connected to MPE. If this option is not specified, `vt3k` defaults to type-ahead “off”. This option enables type-ahead between the user’s keyboard and `vt3k` only; it does not apply to the connection between `vt3k` and the application on the MPE system. This option allows the user to take advantage of type-ahead features provided by HP-UX, but is not sufficient for MPE-iX applications that require **SETVAR HPTYPEAHEAD TRUE**. This also means that screen-oriented applications using character reads fail when the `-t` option is used with `vt3k`.

Supported Configurations

`vt3k` requires at least MPE V V-Delta-5, or MPE XL 1.2. equipped with NS Services and NS LAN Link products.

`vt3k` is supported on the following configurations:

- HP 2392 or HP 700/92 terminal connected via RS-232 to an HP 9000 Series 800 (connected via LAN to an HP 3000).
- HP 9000 Series 300/400/700 workstation running `hpterm`.

`vt3k` is **not** supported from a `telnet` or `rlogin` session.

`hpterm` offers HP Block-Mode terminal emulation when using X-Windows on a Series 300/400 or Series 700 workstation.

`vt3k` can cross gateways, but this requires a proxy server machine on the local network with routing information for systems not on the local network. Contact the network administrator for information about proxy servers.

Since HP 3000s use only IEEE protocol, `ifconfig` on HP-UX must be configured for IEEE in addition to Ethernet.

The target HP 3000 must be running NS Virtual Terminal Services.

To test whether an HP 9000 system and an HP 3000 are communicating, try a remote loop back (see *rlb*(1)) from HP-UX to the HP 3000 and ensure that `dscopy` (see *dscopy*(1)) is working between the two systems.

DIAGNOSTICS

Errors at the transport level are reported using NetIPC error codes. The most common error is:

```
NSR_NO_NODE (40) node does not exist.
```

Some possible causes are: the remote node is running; is on a different network; is running pre-UB-Delta-2 MPE; the node name is incorrect; or the remote node is not listed on the local proxy server. Refer to the *NS/HP 9000 User/Programmer Reference Manual* for a complete list of error codes and corrective actions.

The preferred way to kill `vt3k` is by using `kill -15` which allows cleanup before the session is terminated.

NOTES

To disconnect from a `vt3k` to the wrong HP 3000 by mistake, terminate the connection to the HP 3000 at the MPE logon prompt by typing `:eof:` in response to the MPE V logon prompt or `bye` in response to the MPE/XL logon prompt. The connection should close with the message `Connection Terminated (0)`.

If an HP 3000 session is hung and the HP-UX system supports Job Control (C shell, Korn shell, and POSIX shell), first suspend the `vt3k` session, then run *ps*(1) to obtain the PID of the `vt3k` process. Use a `kill -15` *PID* to terminate the session. A message stating that the process has been stopped should be echoed. Put the suspended shell process back into the foreground, and the HP 3000 session should have terminated with the message `Received signal 15`. (If X-Windows are being used, a second window can be used instead of suspending the foreground process.)

Limitations

`vt3k` only supports line-oriented and VPLUS blockmode applications on the HP 3000. VPLUS applications must use a FORMSPEC file to specify screen layout. `vt3k` does not support non-VPLUS blockmode or screen-oriented applications. `vt3k` does not support hybrid applications that mix VPLUS and MPE intrinsic calls for terminal communications.

The **Break** key may not work if *rlogin* was used to get to the HP-UX system where *vt3k* was invoked.

Character editors such as HPEDIT may run very slowly. When using these editors, the `vt3k -t` option is recommended.

`vt3k` may not transmit some control sequences such as XON/XOFF.

EXAMPLES

To `vt3k` to a remote host, type:

`vt3k` *remote_host*

To `vt3k` to a remote host and enable typeahead, type:

`vt3k -t` *remote_host*

To `vt3k` to a remote host and log output to the file *logfile*, type:

`vt3k -lo -f` *logfile remote_host*

AUTHOR

`vt3k` was developed by HP.

FILES

```
/usr/bin/vt3k
/usr/man/man1/vt3k.1
```

NAME

wait - await process completion

SYNOPSIS

`wait` [*pid*]

DESCRIPTION

If no argument is specified, `wait` waits until all processes (started with `&`) of the current shell have completed, and reports on abnormal terminations. If a numeric argument *pid* is given and is the process ID of a background process, `wait` waits until that process has completed. Otherwise, if *pid* is not a background process, `wait` exits without waiting for any processes to complete.

Because the `wait()` system call must be executed in the parent process, the shell itself executes `wait` without creating a new process (see *wait*(2)).

Command-Line Arguments

`wait` supports the following command line arguments:

`pid`	The unsigned decimal integer process ID of a command, whose termination `wait` is to wait for.

WARNINGS

Some processes in a 2-or-more-stage pipeline may not be children of the shell, and thus cannot be waited for.

SEE ALSO

csh(1), ksh(1), sh-posix(1), sh(1), wait(2).

SEE ALSO

sh(1), wait(2).

STANDARDS CONFORMANCE

`wait`: SVID2, XPG2, XPG3, POSIX.2

NAME

wc - word, line, and character count

SYNOPSIS

`wc [-lwc] [names]`

DESCRIPTION

`wc` counts lines, words, and characters in the named files, or in the standard input if no *names* are specified. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The `l`, `w`, and `c` options can be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is `-lwc`.

When *names* are specified on the command line, they are printed along with the counts.

EXAMPLES

The command:

```
wc-wfile1
```

prints the number of words in `file1`.

The following is printed when the above command is executed:

```
n file1
```

where *n* is the number of words in `file1`.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the range of graphics and space characters.

`LANG` determines the language in which messages are displayed.

If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of `LANG` is used as a default for each unspecified or empty variable. If `LANG` is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of `LANG`. If any internationalization variable contains an invalid setting, `wc` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

WARNINGS

`wc` counts the number of new-lines to determine the line count. If an ASCII text file has a final line that is not terminated with a new-line character, the count will be off by one.

If an input file contains a large number of characters, words, and/or lines, the output may be hard to read. This is because `wc` reserves a fixed column width for each count.

STANDARDS CONFORMANCE

`wc`: SVID2, XPG2, XPG3, POSIX.2

NAME

what - get SCCS identification information

SYNOPSIS

`what [-s]` *file* ...

DESCRIPTION

`what` searches the given files for all occurrences of the pattern that *get*(1) substitutes for **%Z%** (currently **@(#)** at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file `f.c` contains

```
char ident[] = "@(#)identification information";
```

and `f.c` is compiled to yield `f.o` and `a.out`, the command

```
what f.c f.o a.out
```

prints

```
f.c:        identification information
f.o:        identification information
a.out:      identification information
```

`what` is intended to be used in conjunction with the SCCS `get` command (see *get*(1)) which automatically inserts identifying information, but it can also be used where the information is inserted manually.

Options

`what` recognizes the following option:

`-s`	Quit after finding the first occurrence of pattern in each file.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of the pattern substituted for **%Z%** as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If **LC_CTYPE** is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `what` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use `help` for explanations (see *help*(1)).

WARNINGS

The pattern **@(#)** may occasionally appear unintentionally in random files, but this causes no harm in nearly all cases.

SEE ALSO

get(1), help(1).

SCCS User's Guide, in *Programming on HP-UX*.

STANDARDS CONFORMANCE

`what`: SVID2, XPG2, XPG3

NAME

whereis - locate source, binary, and/or manual for program

SYNOPSIS

`whereis` [`-bsm`] [`-u`] [`-BMS` *dir* ... `-f`] *name* ...

DESCRIPTION

`whereis` locates source, binary, and manuals sections for specified files. The supplied names are first stripped of leading path name components and any (single) trailing extension of the form `.`*ext* (such as `.c`). Prefixes of `s.` resulting from use of SCCS are also dealt with. `whereis` then attempts to locate the desired program in a list of standard places.

Options

`whereis` recognizes the following command-line options:

`-b`	Limit search to binary files. Can be used in conjunction with `-s` or `-m`.
`-s`	Limit search to source-code files. Can be used in conjunction with `-b` or `-m`.
`-m`	Limit search to manual entry files. Can be used in conjunction with `-b` or `-s`.
`-u`	Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus, `whereis -m -u *` searches for those files in the current directory that have no corresponding manual entry.
`-B` *dir* [*dir* ...]	Limit search to binaries located in one or more specified directories. Can be used in conjunction with `-S` or `-M`.
`-S` *dir* [*dir* ...]	Limit search to source files located in one or more specified directories. Can be used in conjunction with `-B` or `-M`.
`-M` *dir* [*dir* ...]	Limit search to manual entry files located in one or more specified directories. Can be used in conjunction with `-B` or `-S`.
`-f`	Terminates the last directory list (`-B`, `-S`, or `-M` options) and identifies the start of file names.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Find all the files in `/usr/bin` that are not documented in `/usr/man/man1` with source files in `/usr/src/cmd`:

```
cd /usr/bin
whereis -u -M /usr/man/man1 -S /usr/src/cmd -f *
```

WARNINGS

`whereis` uses the `chdir()` system call (see *chdir*(2)) to run faster. Therefore, path names given with the `-B`, `-M`, and `-S` options must be absolute path names.

AUTHOR

`whereis` was developed by the University of California, Berkeley.

FILES

```
/usr/src/*
/bin, /etc, /lib, /usr/{bin, games, lib}
/usr/man/*
/usr/local/{man/*, bin, games, include, lib}
/usr/contrib/{man/*, bin, games, include, lib}
/usr/man/$LANG/*
/usr/local/man/$LANG/*
/usr/contrib/man/$LANG/*
```

NAME

which - locate a program file including aliases and paths

SYNOPSIS

`which` [*name* ...]

DESCRIPTION

For each *name* given, `which` searches for the file that would be executed if *name* were given as a command, and displays the absolute path of that file. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are determined by sourcing (executing) the user's `.cshrc` file.

DIAGNOSTICS

A diagnostic is given for names that are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

EXAMPLES

The command:

```
which sh
```

specifies where the executable program of the *sh*(1) command is found. For example, the response might be:

```
/bin/sh
```

if the *sh*(1) being used is located in `/bin`.

WARNINGS

`which` reports `.cshrc` aliases even when not invoked from *csh*.

`which` cannot find `csh` built-in commands (e.g. jobs).

`which`'s information may be incorrect because it is unaware of any path or alias changes that have occurred in the current shell session.

AUTHOR

`which` was developed by the University of California, Berkeley.

FILES

`~/.cshrc`	source of aliases and path values

NAME

who - who is on the system

SYNOPSIS

`who [-muTlHqpdbrtasAcR] [`*file*`]`

`who am i`

`who am I`

DESCRIPTION

`who` can list the user's name, terminal line, login time, elapsed time since input activity occurred on the line, the user's host name, and the process-ID of the command interpreter (shell) for each current system user. It examines the `/etc/utmp` file to obtain its information. If *file* is given, that file is examined. Usually, *file* is `/etc/wtmp`, which contains a history of all the logins since the file was last created.

`who` with the `am i` or `am I` option identifies the invoking user.

Except for the default `-s` option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, `who` can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the `init` process.

Options

`-m` Output only information about the current terminal. This option is equivalent to the `am i` and `am I` options described above.

`-u` Lists only those users who are currently logged in. *name* is the user's login name. *line* is the name of the line as found in directory `/dev`. The *time* field indicates when the user logged in.

activity is the number of hours and minutes since input activity last occurred on that particular line. A dot (`.`) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked `old`. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's login process. The *comment* is the comment field associated with this line as found in `/etc/inittab` (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc. If no such information is found, then `who` prints, as the *comment*, the user's host name as it was stored in the `/etc/utmp` or named *file*. Note that the user's host name is printed instead of comments from the `/etc/inittab` file if the `-u` option is used in conjunction with the `-R` option.

`-T` Same as the `-u` option, except that the *state* of the terminal line is printed. *state* describes whether someone else can write to that terminal. A `+` appears if the terminal is writable by anyone; a `-` appears if it is not. `root` can write to all lines having a `+` or a `-` in the *state* field. If a bad line is encountered, a `?` is printed.

`-l` Lists only those lines on which the system is waiting for someone to login. The *name* field is `LOGIN` in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

`-H` Prints column headings above the regular output.

`-q` A quick `who`, displaying only the names and the number of users currently logged in. When this option is used, all other options are ignored.

`-p` Lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in `/etc/inittab`. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from `/etc/inittab` that spawned this process. See *inittab*(4).

`-d` This option displays all processes that have expired and not been respawned by `init`. The *exit* field appears for dead processes and contains the termination and exit values of the dead process (as returned by `wait()` — see *wait*(2)). This can be useful in determining why a process terminated.

`-b`	Indicates the time and date of the last reboot.
`-r`	Indicates the current *run-level* of the `init` process. The last three fields contain the current state of `init` , the number of times that state has been previously entered, and the previous state. These fields are updated each time `init` changes to a different run state.
`-t`	Indicates the last change to the system clock (via the `date` command) by `root`. See *su*(1).
`-a`	Processes `/etc/utmp` or the named *file* with all options turned on.
`-s`	(default) Lists only the *name*, *line*, and *time* fields.
`-A`	When the `/etc/wtmp` file is specified, this option indicates when the accounting system was turned on or off using the `startup` or `shutacct` commands (see *acctsh*(1M)). The *name* field is `.`. The *line* field is `acctg on`, `acctg off`, or a reason that was given as an option to the `shutacct` command. The *time* is the time that the on/off activity occurred.
`-c`	Displays information about an entire SM HP Cluster. If *file* is given and is context dependent (see *cdf*(4)), data from all elements of the CDF are displayed. If *file* is given and is not a CDF, the `-c` option has no effect.
`-R`	Displays the user's host name. If the user is logged in on a tty, `who` displays the string returned from `gethostname()` (see *gethostname*(2)). If the user is not logged in on a tty and the host name stored in the `/etc/utmp` or named file has not been truncated when stored (meaning that the entire host name was stored with no loss of information), it is displayed as it was stored. Otherwise, the `gethostbyaddr()` function is called with the internet address of the host (see *gethostent*(3N)). The host name returned by `gethostbyaddr()` is displayed unless it returns an error, in which case the truncated host name is displayed.

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `who` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

EXAMPLES

Check who is logged in on the system:

```
who
```

Check whether or not you can write to the terminal that another user is using:

```
who -T
```

and look for a plus (+) after the user ID.

AUTHOR

`who` was developed by AT&T and HP.

FILES

```
/etc/inittab
/etc/utmp
/etc/wtmp
```

SEE ALSO

date(1), login(1), init(1), mesg(1), su(1), gethostname(2), wait(2), gethostent(3N), cdf(4), inittab(4), utmp(4).

STANDARDS CONFORMANCE

who: SVID2, XPG2, XPG3

NAME

whoami - print effective current user id

SYNOPSIS

whoami

DESCRIPTION

whoami prints your *current* user name, even if you have used **su** to change it since your initial login (see *su*(1)). The command **who am i** reports your initial login name because it uses `/etc/utmp`.

FILES

`/etc/passwd`
name data base

AUTHOR

whoami was developed by the University of California, Berkeley.

SEE ALSO

who (1).

NAME

write - interactively write (talk) to another user

SYNOPSIS

`write` *user* [*line*]

DESCRIPTION

`write` copies lines from your terminal to that of another user. When first called, it sends the message:

`Message from` *yourname* `(tty`*??*`)` [*date*] ...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should `write` back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed `mesgn`. At that point `write` writes `EOT` on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument can be used to indicate which line or terminal to send to (e.g., `tty00`); otherwise, the first writable instance of the user found in `/etc/utmp` is assumed and the following message posted:

```
user is logged on more than one place.
You are connected to "terminal" .
Other locations are:
terminal
```

Permission to write may be denied or granted by use of the `mesg` command (see *mesg*(1)). Writing to others is normally allowed by default. Certain commands, in particular `nroff` and `pr` disallow messages in order to prevent interference with their output. However, if the user has the appropriate privileges, messages can be forced onto a write-inhibited terminal.

If the character `!` is found at the beginning of a line, `write` calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using `write`: when you first `write` to another user, wait for them to `write` back before starting to send. Each person should end a message with a distinctive signal (such as `(o)` for "over") so that the other person knows when to reply. The signal `(oo)` (for "over and out") could be used when conversation is to be terminated.

EXTERNAL INFLUENCES

Environment Variables

`LC_TIME` determines the format and contents of date and time strings.

If `LC_TIME` is not specified in the environment or is set to the empty string, the value of **LANG** is used as a default for each unspecified or empty variable. If **LANG** is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of **LANG**. If any internationalization variable contains an invalid setting, `write` behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

DIAGNOSTICS

`user is not logged on`
: The person you are trying to `write` to is not logged on.

`Permission denied`
: The person you are trying to `write` to denies that permission (with `mesg`).

`Warning: cannot respond, set mesg -y`
: Your terminal is set to `mesg n` and the recipient cannot respond to you.

`Can no longer write to user`
: The recipient has denied permission (`mesg n`) after you had started writing.

EXAMPLES

By issuing the command:

```
    write matthew
```

`linda` sends a message to `matthew`'s screen. If `matthew` types `write linda`, two-way communication between `matthew` and `linda` is established.

FILES

/etc/utmp	to find user
/bin/sh	to execute `!`

SEE ALSO

elm(1), mail(1), mailx(1), mesg(1), nroff(1), pr(1), sh(1), who(1).

STANDARDS CONFORMANCE

`write`: SVID2, XPG2, XPG3

(Requires Optional LAN/X.25 Software)

NAME

x25stat - display interface status, configuration information, and virtual circuit statistics.

SYNOPSIS

x25stat [-d *dev_file*] [-t] [-p] [-g] [-f] [-c] [-a] [-x] [-s] [-e]

DESCRIPTION

x25stat displays X.25 interface status information, configuration information, and virtual circuit statistics. If no options are provided, **x25stat** displays the virtual circuit types, the status of each virtual circuit, and the connections (local X.121 address and the foreign X.121 address) for all virtual circuits.

Default Operation

If no options are specified, the following information is produced:

LCI — Logical Channel Index number for each virtual circuit. Assignment of LCI values to virtual circuits is based on an agreement between the local node subscriber to the network and the network administration at subscription time. Values are only significant for the local node and network provider interface.

Virtual circuit types — Supported virtual circuit types are:

- SVC-inb (switched virtual circuits inbound),
- SVC-outb (switched virtual circuits outbound),
- PVC (permanent virtual circuits), and
- SVC-2way (switched virtual circuit – two way).

Local address
The X.121 address of the local interface card.

Foreign address
The X.121 address of the foreign host.

VC open time
The length of time the VC has been open in *hours* **.** *minutes* **.** *seconds*.

VC state
Possible VC states are:

- `disconnecting` (in the process of disconnecting)
- `connecting` (a connect call has been issued but no acknowledgement has been received),
- `connected` (the VC is connected),
- `inactive` (the PVC has not been used, although the interface card has been initialized), or
- `unknown` (an error condition occurred on the VC).

Options

-d dev_file — Obtain information for X.25 driver interface *dev_file*. If this option is not specified, the default value of **/dev/x25_0** is assumed. X.25 subsystem device file are located in directory **/dev**.

-t — Display virtual circuit data counters. Virtual circuit statistics for this option are cleared at the start of each connection. Counters are provided for the following categories:

LCI — Logical Channel index number for each virtual circuit. Assignment of LCI values to virtual circuits is based on an agreement between the local node subscriber to the network and the network administration at subscription time. Values are only significant for the local node and network provider interface.

VC state — possible VC states are:

- `disconnecting` (in the process of disconnecting)
- `connecting` (a connect call has been issued but no acknowledgement has been received),
- `connected` (the VC is connected),
- `inactive` (the PVC has not been used, although the interface card has been initialized), or

- `unknown` (an error condition occurred on the VC).

Imsgs
number of inbound messages for this virtual circuit

Omsgs
number of outbound messages for this virtual circuit

Ipackets
number of inbound packets received by this virtual circuit (not displayed on Series 300/400)

Opackets
number of outbound packets sent by this virtual circuit (not displayed on Series 300/400)

Ioctets
number of inbound octets received by this virtual circuit

Ooctets
number of outbound octets sent by this virtual circuit

-p
Display non-data packet counts for each virtual circuit. Virtual circuit statistics for this option are cleared at the start of each connection.

LCI Logical Channel index number for each virtual circuit. Assignment of LCI values to virtual circuits is based on an agreement between the local node subscriber to the network and the network administration at subscription time. Values are only significant for the local node and network provider interface.

VC state Possible VC states are:

- `disconnecting` (in the process of disconnecting)
- `connecting` (a connect call has been issued but no acknowledgement has been received),
- `connected` (the VC is connected),
- `inactive` (the PVC has not been used, although the interface card has been initialized), or
- `unknown` (an error condition occurred on the VC).

VC User
Indicates the user of the VC. This can be level-3, internet protocol, or no current user (PVC not being used).

Interrupt Msgs
Number of inbound interrupt messages received by this VC (not displayed on Series 300/400).

Inb. Resets
Number of inbound reset messages received by this VC (not displayed on Series 300/400).

-g Display a global summary of interface statistics for the specified device file. Statistics are cleared each time the interface card is initialized. Output reports a list of global statistics, level 3 access specific statistics, and IP specific statistics.

Global statistics shows activity since the card was initialized:

- Number of incoming calls cleared by the subsystem.
- Number of outgoing calls cleared by the network or the remote peer.
- Total number of packets sent (not displayed on Series 300/400).
- Total number of packets received (not displayed on Series 300/400).
- Total number of octets sent.
- Total number of octets received.
- Total number of resets received (not displayed on Series 300/400).
- Total number of restarts received (not displayed on Series 300/400).

- Total number of diagnostic packets received (not displayed on Series 300/400).

Level 3 Access Specific Statistics report Level 3 activity:

- number of outgoing calls: X.25 direct access
- number of incoming calls answered: X.25 direct access
- number of outgoing calls cleared by X.25 direct access users
- number of outgoing calls cleared due to user process abort

IP-specific statistics report activity related to IP:

- number of outgoing calls: IP access
- number of incoming calls answered: IP access
- number of VCs cleared due to IP idle timer expiration
- number of VCs cleared by IP due to lack of free VCs
- number of outbound messages dropped: Queue overflow
- number of outbound messages dropped: no VC available
- number of outbound messages dropped: no VC to next hop destination
- number of outbound messages dropped: no IP-to-X.121 address map entry

-f

Display X.25 Level 2 and IP for *dev_file*.

Level 2 statistics describing Level 2 activity:

- number of RRs sent/received RR (Receive Ready) is a function of the S (link set up) frame.

number of RNRs sent/recv
: RNR (Receive Not Ready) is a function of the S (link set up) frame.

number of REJs sent/recv
: REJ (Reject) is a function of the S (link set up) frame.

number of frames received that were too long
: The count reports how many frames were transmitted that contained a number of octets which exceeded the maximum frame size. The value of frame size is set when the subsystem is configured.

number of I-frames sent/recv
: I (information) frames carry packets and therefore user's data.

number of Data Carrier Detect changes
: contains the number of DCD state changes.

number of Clear-To-Send changes
: contains the number of CTS state changes.

number of CRC errors
: In the event of packet retransmission, the Cyclic Redundancy Check reviews the data in duplicated packets. This count shows how many packets of which the data was corrupted.

number of T1 timer expirations
: The T1 timer is used to specify the amount of time to wait for acknowledgement of a frame.

# of Aborted frames sent/recv
: contains the number of incomplete packets that were sent and received.

Transmitter underruns
: contains the number of times that a transmission was aborted because the data to be transmitted was not provided soon enough.

Receiver overruns
: contains the number of times that a reception was aborted because the data was not processed by the interrrupt routine fast enough.

-c
Display current X.25 configuration. Parameter values specified in the configuration file or through the **x25init** user interface are displayed. Refer to *x25init_smpl*(4) for a description of configuration file parameters.

-a Display information present in the IP mapping table:

IP Address Internet Protocol address

X.121 Address
: X.121 address corresponding to this IP address

PVC
: Logical Channel Identifier number of the permanent virtual circuit of this interface card

Use Reverse Charges
: Shows whether reverse charge calls can be issued across this X.121 address

Accept Reverse Charges
: Shows whether reverse charge calls are accepted from across this X.121 address

-e
Display inbound/outbound effective size and class information (not displayed on Series 300/400):

LCI
: Logical Channel Index number for each virtual circuit. Assignment of **LCI** values to virtual circuits is based on an agreement between the local node subscriber to the network and the network administration at subscription time. Values are only significant for the local node and network provider interface.

Eff Ipkt/Opkt sizes
: is the effective size of the inbound and outbound packet in octets. This value is the result of packet size negotiation on the VC.

Eff Iwin/Owin Sizes
: is the effective size of the inbound and outbound window in octets. This value is the result of packet size negotiation on the VC.

Eff Ithr/Othr Sizes
: is the effective inbound and outbound throughput class number. It indicates to the network the amount of resources to allocate to a virtual circuit. This value is the result of packet size negotiation on the VC.

-s
Show status information for all virtual circuits (not displayed on Series 300/400).

LCI
: Logical Channel Index number for each virtual circuit. Assignment of **LCI** values to virtual circuits is based on an agreement between the local node subscriber to the network and the network administration at subscription time. Values are only significant for the local node and network provider interface.

Virtual Port #
: mapping between the **LCI** and the connection-id.

Destination
: Is 1 if the virtual circuit is being used for PAD support or by a user.

State
: The current state of the virtual circuit. Possible VC states are: disconnected, connecting, connected, inactive, unknown, reset, ready, reset indication, DTE waiting, DCE waiting, reset request, clear request, collision, PVC out of order.

Trace State
: Shows whether VC path tracing is set for the virtual ciruit designated by the **LCI**

number.

Fast Select

Value shown is one of the following:

`with restriction`
indicates that user data cannot be sent with CALL ACCEPTED packets. However, the called DTE (destination node) may send user data with a CLEAR REQUEST packet.

`without restriction`
indicates that full Fast select facilties may be used.

`not set.`
indicates that Fast select is not being used.

-x

Display Level 3 X.25 protocol information for the interface specified by the `-d`*dev_file* parameter (not displayed on Series 300/400).

Interface state
This value reflects Level 2 information. The value shown is one of the following: `UNINITIALIZED`, `UNINITIALIZED - Initialization failed`, `INITIALIZED`, `Lock-out is set`, or `DOWN`.

Protocol state
Reflects Level 3 information. Value shown is one of the following: `STOPPED`, `DISCONNECTED`, `RESTART REQUESTED`, or `READY`.

Tracing state
Reflects Level 3 information. Value shown is one of the following: `Not tracing`, `Tracing all VCs`, `Tracing reserved for DTC`, or `Tracing a` *specified* `VC`.

Number of Hosts Connected Through This Interface
Total number of hosts communicating through the designated interface

Number of Switching Elements
Total number of switching elements through which all virtual circuits using this interface must pass

Number of Open SVCs
Number of currently open switched virtual circuits

Max. Number of Simultaneously Opened VCs
Maximum number of virtual circuits that may be open concurrently
Value is hardware dependent.

Number of VCs Opened
Total number of virtual vircuits that are currently opened

DIAGNOSTICS

`x25stat: Could not get machine type.`
Attempt to get the machine type through a system call resulted in an error.

`x25stat: Could not open default subsystem device.`
Attempt to use `x25stat` without the `-d` option failed because the default subsystem management device (`x25_0`) was not initialized, or the device file was not present.

`x25stat: Could not read interface state.`
Interface status information was not received. Check to ensure that the designated interface was initialized.

`x25stat: NOTICE: X.25 subsystem is not currently active on device.`
The `x25init` command was not executed for this device file. An attempt was unsuccessfully made to use `x25stat` on an interface name that was not initialized using `x25init`.

`x25stat: Error on subsystem device read.`
System returned an error while attempting to get information from the subsystem management device driver.

`x25stat: No data returned from subsystem subsystem device.`
No information was received from subsystem management device driver.

`x25stat: NOTICE: Interface driver type is unknown.`
The driver type specified was not an X.25 interface driver.

`x25stat: IO error on access to card. (errno = EIO)`
`x25init` may not have been executed for this interface.

`x25stat: Cannot read interface configuration information.`
System error occured while attempting to get information from the kernel.

`Bad self test on interface card.`
Hardware problem. If this messages is displayed continuously, the interface card may need to be replaced.

DEPENDENCIES

Series 300/400:

`-e`, `-s`, and `-x` options are not supported.

AUTHOR

`x25stat` was developed by HP.

FILES

`x25stat.cat`	native language catalog file

SEE ALSO

x25init(1M), x25stop(1M),

Installing and Administering X.25/9000.

NAME

xargs - construct argument list(s) and execute command

SYNOPSIS

xargs [*options*] [*command* [*initial-arguments*]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the options specified.

command, which can be a shell file, is searched for, using the **$PATH** environment variable. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Spaces and tabs can be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings, a backslash (\) escapes the next character.

The amount of memory available for the execution of *command* is limited by the system parameter **ARG_MAX**. By default, the size of the argument list is limited to **LINE_MAX** bytes. See *limits*(5) and *sysconf*(2) for a description of these system parameters and how their values can be determined. To increase the available argument list space, use the **-s** option.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (exception: see **-i** option). The **-i**, **-l**, and **-n** options determine how arguments are selected for each command invocation. When none of these options is specified, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are option conflicts (such as **-l** versus **-n**), the last option has precedence. *option* values are:

-l *number* — *command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. **1** is assumed if *number* is omitted or is given as the empty string (Option **-x** is forced.

-i *replstr* — Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* can each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are discarded. Constructed arguments must not grow larger than 255 characters, and option **-x** is also forced. **{ }** is assumed if *replstr* is omitted or is given as the empty string (

-n *number* — Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments are used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **-x** is also coded, each *number* arguments must fit in the *size* limitation or **xargs** terminates execution.

-s *size* — The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than **LINE_MAX** (see *limits*(5), *sysconf*(2)). If **-s** is not coded, **LINE_MAX** is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-t — Trace mode: The *command* and each constructed argument list are echoed to standard error just prior to their execution.

-p — Prompt mode: The user is asked whether to execute *command* prior to each invocation. Trace mode (**-t**) is turned on to print the command instance to be executed, followed by a `?...` prompt. A reply of `y` (optionally followed by anything) executes the command; anything else, including pressing Return, skips that particular invocation of *command*.

-x Causes *xargs* to terminate if any argument list would be greater than *size* characters. **-x** is forced by the options **-i** and **-l**. When none of the options **-i**, **-l**, or **-n** is coded, the total length of all arguments must be within the *size* limit.

-e *eofstr* *eofstr* is taken as the logical end-of-file string. Underscore (_) is assumed for the logical **EOF** string if **-e** is not coded. The value **-e** with *eofstr* given as the empty string (turns off the logical **EOF** string capability (underscore is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs terminates if it receives a return code of −1 from *command* or if it cannot execute, *command*. When *command* is a shell program, it should explicitly **exit** (see *sh*(1)) with an appropriate value to avoid accidentally returning with −1.

RETURN VALUE

xargs exits with one of the following values:

- **0** All invocations of *command* completed successfully.
- **>0** One or more invocations of *command* did not complete successfully.

EXAMPLES

Move all files from directory **$1** to directory **$2**, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

Combine the output of the parenthesized commands onto one line, then echo to the end of file **log**:

```
(logname; date; echo $0 $*) | xargs >>log
```

Ask the user which files in the current directory are to be archived then archive them into **arch** one at a time:

```
ls | xargs -p -l ar r arch
```

or many at a time:

```
ls | xargs -p -l | xargs ar r arch
```

Execute **diff** (see *diff*(1)) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO

sh(1).

STANDARDS CONFORMANCE

xargs: SVID2, XPG2, XPG3, POSIX.2

NAME

xdb - C, FORTRAN, Pascal, and C++ Symbolic Debugger

SYNOPSIS

xdb [**-d** *dir*] [**-r** *file*] [**-R** *file*] [**-p** *file*] [**-P** *process_ID*] [**-L**] [**-l** *library*]
[**-i** *file*] [**-o** *file*] [**-e** *file*] [**-S** *num*] [**-s**] [*objectfile* [*corefile*]]

DESCRIPTION

xdb is a source level debugger for C, HP FORTRAN, HP Pascal, and C++ programs. It provides a controlled environment for their execution. See the *HP-UX Symbolic Debugger User's Guide* for a comprehensive description of *xdb* .

objectfile is an executable program file having zero or more of its component modules compiled with the debug option turned on (enabled by the **-g** option of the **cc**, **f77**, **pc**, and **CC** compilers). The support module **/usr/lib/end.o** must be included as the last object file linked, except for libraries included with the **-l** option to **ld** (see *ld*(1)). The support module is included automatically when **ld** is invoked as part of a compile command that uses the **-g** option. The default for *objectfile* is **a.out**. Note that by default **ld** links in shared libraries instead of archive libraries.

corefile is a core image from a failed execution of *objectfile*. The default for *corefile* is **core**.

Options

xdb recognizes the following options:

-d *dir*	Specify *dir* as an alternate directory where source files are located.
-r *file*	Specify a record *file*, which is invoked immediately (for overwrite, not for append).
-R *file*	Specify a restore state *file*, which is processed before the **-p** option (if any) and after the **-r** option (if any).
-p *file*	Specify a playback *file* which is invoked immediately.
-P *process_ID*	Specify the process ID of an existing process the user wants to debug.
-L	Use the line-oriented interface.
-l *library*	Pre-load information about this shared *library,* **-l ALL** means always pre-load shared library information.
-i *file*	Redirect standard input to the child process from the designated file or character device.
-o *file*	Redirect standard output from the child process to the designated file or character device.
-e *file*	Redirect standard error from the child process to the designated file or character device.
-S *num*	Set the size of the string cache to *num* bytes (default is 1024, which is also the minimum).
-s	Enable debugging of shared-libraries.

At start-up, the debugger executes commands from the file **$HOME/.xdbrc**, if it exists.

ENVIRONMENT VARIABLES

Display

TERM	This variable specifies the terminal type. There is no default for the terminal type.
LINES	This variable specifies the window height in lines of text. The default for this variable is 24 if not otherwise determinable.
COLUMNS	This variable specifies the window width in text columns. The default for this variable is 80 if not otherwise determinable.

Command Line Editing

XDBHIST	This variable specifies the history file. The default for this variable is **$HOME/.xdbhist**.
HISTSIZE	This variable specifies the actual number of commands allowed in the history file. The default for this variable is **128**.

`XDBEDIT` This variable specifies the editing mode (`vi`, `emacs`, or `gmacs`). The default for this variable is to match the environment variable `VISUAL` or `EDITOR`; otherwise, there is no default.

Native Language Support

LANG This variable determines the local language equivalent of `y` (for yes/no queries). LANG also determines the locale in which messages are displayed. The default value for this variable is `C`.

`LC_CTYPE` This variable determines the interpretation of text as single- and/or multi-byte characters and their printability when reading or writing character and string data. If `LC_CTYPE` is not specified in the environment or is set to the empty string, the value of LANG is used as the default.

International Code Set Support

Single- and multi-byte character code sets are supported.

LOCATION SYNTAX

line A *number* that refers to a particular line in a file.

location A particular *line* in a file and its corresponding address in the user's program. *location* has the following general forms:

line
#label
file[**:***line*]
[*file***:**]*proc*[**:***proc*[...]][**:***line* | **#***label*]
[*class*]**: :***proc*[**:***line* | **#** *label*]

To reference code addresses symbolically, use:

*proc***#***line*
[[*class*]**: :**]*proc***#***line*

Variable Identifiers

Variables are referenced exactly as they are named in the source file or files. Uppercase/lowercase sensitivity is controlled by the `tc` command.

Several methods can be used to obtain the value of some variable *var*, depending on where and what it is:

var Search for *var* first as a local in the current procedure (or the most recent instance of the current procedure), next as a member of that procedure's class, or finally as a global.

class **: :** *var* Search *class* for variable.

proc **:** *var*
[[*class*]**: :**]*proc* **:** [*class* **: :**]*var*
Search for *var* in the current or most recent instance of *proc*. A leading :: indicates a global.

proc **:** *depth* **:** *var*
[[*class*]**: :**]*proc* **:** *depth* **:** [*class* **: :**]*var*
Use the instance of *proc* that is at depth *depth* (exactly), instead of the current or most recent instance.

: *var*
: : *var* Search for a global (not local) variable named *var*.

. *dot* is shorthand for the last thing viewed.

Special Variables

Special variables are names for things that are not normally directly accessible. Special variables include:

$var The debugger has room in its own address space for several user-created special variables of type `long` .

`$pc`, `$sp`, `$r7`, etc.
These are the names of the program counter, the stack pointer, the CPU general registers, etc.

`$result`	This is used to reference the return value from the last command-line procedure call. It can also be referenced as `$short` and `$long`.
`$signal`	This lets you see and modify the current child process signal number.
`$lang`	This lets you see and modify the current language. Possible values are: `C`, `C++`, **FORTRAN**, `Pascal`, `default`.
`$depth`	This is the default stack level for viewing local variables.
`$print`	Alters the behavior of the "print" command when printing character data. Values that can be assigned are `ascii`, `native`, and `raw`.
`$line`	This lets you see and modify the current source line number.
`$malloc`	This lets you see the current amount of memory (bytes) allocated at run-time for use by the debugger itself.
`$step`	This lets you see and modify the number of machine instructions the debugger steps while in a non-debuggable procedure before setting an up-level breakpoint and free-running to it.
`$cplusplus`	This is interpreted as a set of flags to control behavior of certain C++ capabilities.

`bit 0`	Set means print full base class information at each occurrence.
`bit 1`	Set means `bpc` sets breakpoints on member functions of base classes, also.
`bit 2`	Set means `bi` sets breakpoints on member functions of base classes, also.

The default for all bits is clear. Some commands take a `-c` or `-C` argument which causes the action to be as if the appropriate bit of `$cplusplus` was clear (`-c`) or set (`-C`).

COMMANDS

The debugger has a large number of commands for viewing and manipulating the program being debugged. They are explained below and are grouped according to functional similarity.

Window Mode Commands

These commands control what is displayed in the source window. The source window by default comes up in source mode for viewing source code. If assembly language instructions are needed, the disassembly mode can be selected. Registers are also shown in this mode. If both assembly instructions and source code are needed, the split-screen mode can be selected. Commands are as follows:

`td`	Toggle disassembly mode.
`ts`	Toggle split-screen mode.
`gr`	Display the general registers when the debugger is in disassembly (non-split-screen) mode.
`fr`	Display the floating-point registers when the debugger is in disassembly (non-split-screen) mode.
`+r`	Scroll the floating-point register display forward four lines.
`-r`	Scroll the floating-point register display back four lines.
`w` [*size*]	Set the size of the source viewing window.
`u`	Update the screen to reflect the current location.
`U`	Clear and redraw the screen.

Path Map Commands

Path maps can be used to redirect portions of a source tree to different directories. Commands are as follows:

`apm` *old_path* [*new_path*]
Add a path map to the list of path maps.

`lpm`	List path maps. The list is numbered for use with the `dpm` command.
`dpm` [*number* | `*`]	Delete path map. Default *number* is 1 (most recent path map). A `*` deletes all path maps.
`D` *directory*	Adds *directory* to the list of alternate directory search paths for source files.

File Viewing Commands

These commands may change the current viewing position, but they do not affect the next statement to be executed in the child process, if any. Commands are as follows:

`v`	View the source one window forward from the *current* source window.
`v` *location*	View the source at the specified *location*, placing it in the center of the window.
`V` [*depth*]	View the source at the current procedure at *depth* on the stack.
`va` [*address*]	View the assembly code at *address* in the source window (see the `td` command).
`L`	Display the file name, procedure name, line number, and the current source statement corresponding to the object code being executed or examined.
`+` [*lines*]	Move to *lines* (default one) lines after the current line.
`-` [*lines*]	Move to *lines* (default one) lines before the current line.
`/` [*string*]	Search forward through the current file for *string*, starting at the line after the current line.
`?` [*string*]	Search backward for *string*, starting with the line before the current line.
`n`	Repeat the previous `/` or `?` command using the same *string* as the last search, starting at the current location being viewed.
`N`	The same as `n`, but the search goes in the opposite direction from that specified by the previous `/` or `?` command.

Display Formats

The display formats tell the debugger's data viewing commands how output should be formatted. A *format* is of the form [*count*]*formchar*[*size*]. For example, `p abc\4x2` prints, starting at the location of `abc`, four two-byte numbers in hexadecimal.

Formats that print numbers use lowercase characters to represent `integer` data types and uppercase to represent `long` data types. For example, `O` prints in `long` octal.

The following formats are available:

`n`	Print in the "normal" format, based on the type. Arrays of `char` and pointers to `char` are interpreted as strings, and structures are fully dumped.
(`d`|`D`)	Print in decimal (as `integer` or `long`).
(`u`|`U`)	Print in unsigned decimal (as `integer` or `long`).
(`o`|`O`)	Print in octal (as `integer` or `long`).
(`x`|`X`)	Print in hexadecimal (as `integer` or `long`).
(`z`|`Z`)	Print in binary (as `integer` or `long`).
(`b`|`B`)	Print a byte in decimal (either way).
`c`	Print a character.
`C`	Print a wide-character.
(`e`|`E`)	Print in "e" floating-point notation (as `float`, `double`, or `long`).
(`f`|`F`)	Print in "f" floating-point notation (as `float`, `double`, or `long double`).
(`g`|`G`)	Print in "g" floating-point notation (as `float`, `double`, or `long double`).
`i`	Print a disassembled machine instruction.

`a`	Print a string using *expr* as the address of the first byte.
`w`	Print a wide-character string using *expr* as the address of the first element.
`W`	Print a wide-character string using *expr* as the address of a pointer to the first element.
`r`	Print the template of an object (C++).
`R`	Print the template of an object with base classes displayed (C++).
`s`	Print a string using *expr* as the address of a pointer to the first byte.
`t`	Show the type of *expr* (usually a variable or procedure name).
`T`	This is identical to the `t` format except for C++ classes and struct objects where base class and struct type information are also displayed.
`p`	Print the name of the procedure containing address *expr*.
`S`	Do a formatted dump of a structure.
`k`	Identical to the `S` format.
`K`	Same as the `S` format, except that for C++ class and struct objects the base class and struct data are also displayed.

There are some short hand notations for *size*:

`b`	1 byte (`char`).
`s`	2 bytes (`short`).
`l`	4 bytes (`long`).
`D`	8 bytes (`double`). Can only be used with floating-point formats.
`L`	16 bytes (`long double`). Can only be used with floating-point formats.

Data Viewing and Modification Commands

`p` *expr*	If *expr* does not resemble anything else (such as a command), it is handled as if you had typed `p` *expr*`\n` (print expression in normal format).
`p` *expr*`\`*format*	Print the contents (value) of *expr* using *format*.
`p` *expr*`?`*format*	Print the address of *expr* using *format*.
`p -`[[`\`]*format*]	Back up to the preceding memory location (based on the size of the last thing displayed) and use *format* if supplied, or the previous *format* if not.
`p +`[[`\`]*format*]	Go forward to the following memory location (based on the size of the last thing displayed) and use *format* if supplied, or the previous *format* if not.
`p` *class*`::`	Print the values of all static data members of *class*.
`pq` *argument*	Print quietly (evaluate but do not print). *argument* can be anything allowed after `p`.
`l` [*proc*[`:`*depth*]] `l` [[[*class*]`::`][*proc*[`:`*depth*]]]	List all parameters and local variables of the current procedure (or of *proc*, if given, at the specified *depth*, if any).
`la`	List all **a**ssertions.
`lb`	List all **b**reakpoints.
`ld`	List all **d**irectories (where to search for files).
`lsl`	List all shared libraries known to the debugger.
`lz`	List all **z**ignals (signal actions).
`lc` [*string*]	List all (or matching) **c**ommon blocks in the current procedure (FORTRAN).
`lf` [*string*]	List all (or matching) **f**iles (source files used to build *objectfile*).
`lg` [*string*]	List all (or matching) **g**lobal variables.
`ll` [*string*][`@`*library*]	List all (or matching) **l**abels.
`lm` [*string*]	List all (or matching) **m**acros.

`lp` [[*class*]`::`][*string*]	List all (or matching) procedure names.
`lr` [*string*]	List all (or matching) registers.
`ls` [*string*]	List all (or matching) special variables (except registers).
`lx`	List exception stop-on-throw and -catch state (C++).
`lcl` [*string*]	List all (or matching) classes (C++).
`lct` [*string*]	List all (or matching) class templates (C++).
`ltf` [*string*]	List all (or matching) template functions (C++).
`lft` [*string*]	List all (or matching) function templates (C++).
`lo` [[*class*]`::`][*string*]	List all (or matching) overloaded functions (C++).
`mm` [*string*]	Show a memory-map of all currently loaded shared-libraries and the main program, or of the one specified.

Stack Viewing Commands

`t` [*depth*]	Trace the stack for the first *depth* (default 20) levels.
`T` [*depth*]	Same as `t`, but local variables are also displayed, using `\n` format (except that all arrays and pointers are shown simply as addresses, and structures as first word only).
`up` [*offset*]	Move up (decreasing depth) *offset* levels in the stack. The default value of *offset* is 1.
`down` [*offset*]	Move down (increasing depth) *offset* levels in the stack. The default value of *offset* is 1.
`top`	Move to the top of the stack (this is the same as `V 0`).
`tst`	Toggle the visibility of inter-procedural stubs in stack traces (PA-RISC only).

Job Control Commands

These commands let you control execution of the program. The parent (HP Symbolic Debugger) and child (*objectfile*) processes take turns running. The debugger is only active and able to execute commands while the child process is stopped due to a signal or a breakpoint, or by terminating.

`r` [*arguments*]	Run a new child process with the given or previous argument list, if any.
`R`	Run a new child process with no argument list.
`k`	Terminate (kill) the current child process, if any.
`c` [*location*]	Continue after a breakpoint or a signal, ignoring the signal, if any. If a *location* is specified, a tempory breakpoint is set at that *location*.
`C` [*location*]	Continue just like `c`, but allow the signal (if any) to be received. If a *location* is specified, a tempory breakpoint is set at that *location*.
`s` [*count*]	Single step 1 (or *count*) statements (or instructions in disassembly mode).
`S` [*count*]	Similar to `s`, but treat procedure calls as single statements (do not step "into" them). The `s` and `S` commands pass the current signal (like `C`). Set `$signal = 0` if necessary, to prevent this.

Breakpoint Commands

The debugger provides a number of commands for setting and deleting breakpoints. Associated with any breakpoint are three attributes:

location	A particular *line* in a file and its corresponding address in the user's program, if executable code exists for that line.
count	The number of times the breakpoint is encountered prior to recognition. This can be suffixed with `p` (permanent, which is the default) or `t` (temporary).
commands	Actions to be taken upon recognition of a breakpoint before waiting for command input. This is a list of debugger commands separated by `;` and enclosed by `{ }`.

Each breakpoint can be individually activated or suspended, and there is an overall breakpoint mode which can be toggled. If any breakpoint is added or activated, or if all breakpoints are suspended, the global mode is toggled automatically.

Here are the breakpoint commands:

`lb`	List all breakpoints.
`b` [*location*] [`\`*count*] [*commands*]	Set a permanent breakpoint at the current location (or at *location*).
`db` [*number*]	Delete breakpoint number *number*, or at the current *location*.
`db *`	Delete all breakpoints (including "procedure" breakpoints).
`bi` *expr.proc* [`\`*count*] [*commands*]	After evaluating *expr* to what must be a class instance, set an "instance" breakpoint at the first executable line of *proc* for the instance's class.

bi [-c | -C] *expr* [*commands*]
: After evaluating *expr* to what must be a class instance, set "instance" breakpoints at the first executable line of all member functions of the instance's class. The **-c** option indicates only members of the designated class. The **-C** option indicates members of base classes as well as members of the designated class.

bpc [-c | -C] *class* [*commands*]
: Set "class" breakpoints at the first executable line of all member functions of *class*. See the previous command for information on **-c** and **-C**.

bpo [[*class*]**::**]*proc* [*commands*]
: Set "overload" breakpoints at the first executable line of all overloaded functions with name *proc* (which can be qualified by a *class*).

bp [*commands*] Set permanent breakpoints at the beginning (first executable line) of every debuggable procedure.

bpx [*commands*] Set permanent breakpoints at the exit (final executable statement) of every debuggable procedure.

bpt [*commands*] Set permanent breakpoints at the entry and exit (first and final executable statement) of every debuggable procedure. The given *commands* are associated with the entry breakpoint and default to `Q;t 2;c`

dp Delete all procedure breakpoints.

Dpx Delete all "procedure exit" breakpoints.

Dpt Delete all "procedure trace" breakpoints.

abc *commands* Define a global breakpoint command list to be executed whenever any breakpoint is hit (normal, instance, class, overload, procedure, procedure exit, or procedure trace).

dbc Delete the global breakpoint command.

bb [*depth*] [\count] [*commands*]
: Set a breakpoint at the beginning (first executable line) of the procedure at the given stack *depth* (if *depth* is not given use the current procedure).

bx [*depth*] [\count] [*commands*]
: Set a breakpoint at the exit (last executable line) of the procedure at the given stack *depth* (if *depth* is not given use the current procedure).

bu [*depth*] [\count] [*commands*]
: Set an up-level breakpoint.

bt [*depth* | *proc*] [\count] [*commands*]
: Trace the current procedure (or procedure at *depth*, or *proc*). By default, the entry breakpoint *commands* are `Q;t 2;c` , which shows the top two procedures on the stack and continues.

ba *address* [\count] [*commands*]
: Set a breakpoint at the given code address.

txc Toggle the exception stop-on-catch state.

txt Toggle the exception stop-on-throw state.

xcc [*commands*] Define the stop-on-catch command-list.

xtc [*commands*] Define the stop-on-throw command-list.

bc *number count* Set the count of breakpoint *number* to *count*.

sb [*num*] Suspend the breakpoint having the *num* specified, or if a *num* is not entered, the breakpoint at the current line is suspended if one exists.

sb * Suspend all breakpoints.

ab [*num*] Activate breakpoint number *num* or if a *num* is not entered, the breakpoint at the current line is activated if one exists.

`ab *`	Activate all breakpoints.
`tb`	Toggle the overall breakpoint mode between *active* and *suspended*.

Auxiliary Breakpoint Commands

`if` *expr* {*commands*} [{*commands*}]	If *expr* evaluates to a non-zero value, the first group of commands (the first `{}` block) is executed; otherwise it (and the following `{`, if any) is skipped.
`Q`	If the `Q`uiet command appears as the first command in a breakpoint command list, the normal announcement of `breakpoint at` *address* is not made.
"*any string you like*"	Print the given string.

Assertion Control Commands

Assertions are command lists that are executed before every instruction. If there is an active assertion, the program is single-stepped at the machine-instruction level and runs very slowly.

Each assertion can be individually activated or suspended, and there is an overall assertions mode which can be toggled. If any assertion is added or activated or if all assertions become suspended, the global mode is toggled automatically.

Here are the assertion commands:

`a` *commands*	Create a new assertion with the given command list which is not parsed until it is executed.
`aa` *number*	Activate assertion *number*.
`aa *`	Activate all assertions.
`da` *number*	Delete assertion *number*.
`da *`	Delete all assertions.
`sa` *number*	Suspend assertion *number*.
`sa *`	Suspend all assertions.
`ta`	Toggle the overall assertions mode between *active* and *suspended*.
`x` [*mode*]	Force an exit from assertions *mode* immediately (default or *mode* is non-zero) or at the end of the command list (*mode* non-zero).

Signal Control Commands

These commands are used to modify and list the contents of the "zignal" (signal) handling table. Here are the signal control commands:

`z` [*signal*] `[ i ] [ r ] [ s ] [ Q ]`	Toggles flags (`i`gnore, `r`eport, or `s`top) for signals (`Q`uietly).
`lz`	Lists the current handling of all signals.

Record and Playback Commands

These commands allow the recording of debugger sessions in a recordfile and the playing back of those sessions. Here are the record and playback commands:

>*file*	Set or change the recordfile to *file* and turn recording on.
>>*file*	Same as >*file*, but appends to *file* instead of overwriting.
>@*file* >>@*file*	Set or change the record-all file to *file*, for overwriting or appending.
`>t`	Turn recording on.
`>f`	Turn recording off.
`>c`	Close the recording file When recording is resumed, new commands are appended to the file. In this context, >> is equivalent to >.

`>@t`	Turn record-all on
`>@f`	Turn record-all off
@u-3p `>@c`	Close the record-all file In this context, `>>@` is equivalent to `>@`.
`tr` [@]	Toggle recording [record-all]; if on, turn it off; if off, turn it on.
`>`	Tell the current recording status (same as `>>`).
`>@`	Tell the current record-all status (same as `>>@`).
<*file*	Start playback from *file*.
<<*file*	Start playback from *file*, using the single-step feature of playback.

Save State Command

`ss` *file*	Save the current set of breakpoints, macros and assertions in *file* for later use with the `-R` command-line option.

Macro Definition Commands

`def` *name* [*replacement-text*]	Define *name* as a macro whose value is *replacement-text*.
`undef` *name*	Remove the macro definition from *name* so that *name* no longer exists as a replacement string macro.
`tm`	Toggle the state of the macro substitution mechanism between active and suspended.

Miscellaneous Commands

`sm`	Suspend the built-in *more* pagination facility of the debugger output.
`am`	Activate the built-in *more* pagination facility to paginate the debugger output.
<carriage-return> `~`	Repeat the last command, if possible, with an appropriate increment, if any.
`!` [*command-line*]	Invoke a shell program.
`#` [*text*]	Flag *text* as a comment to be echoed to the command window.
`f`[*printf-style-format*]	Set the address printing format using *printf*(3S) format specifications (*not* debugger format styles). If no argument is provided, the format is set to the default, `%10.8lx` .
`g` (*line* | *#label* | +[*lines*] | -[*lines*])	Go to an address in the procedure on the stack at *depth* zero (not necessarily the current procedure).
`h` [*topic*] `help` [*topic*]	Print commands/syntaxes related to this *topic* using *more*(1). Use `h help` for a list of topics.
`I`	Print information (inquire) about the state of the debugger and various toggles.
`M`	Print the current text (*objectfile*) and core (*corefile*) address maps.
`tM`	Toggle the address mapping of *corefile* between the initial map and the modifiable mapping pair which the user can set with the `Mc` command.
`M(t | c)` [*expr* [`;` *expr* [...]]]	Set the `t`ext (*objectfile*) or the modifiable `c`ore (*corefile*) address map.
`q`	Quit the debugger.
`tc`	Toggle case sensitivity in searches.

ADOPTING AN EXISTING PROCESS

The symbolic debugger (**xdb**) command line option `-P` *process_ID* allows for the debugging of a free-running process. To adopt a process, the effective user IDs of the debugger and the process to be adopted must match, or the effective user ID of the debugger must be *root*. When a process is adopted, it halts, and the debugger displays where the program is halted, at which point the program can be debugged. If the user quits the debugger without killing the process, the debugger removes all breakpoints from the process and allows it to continue running. If a program is designed to be adopted by the debugger when in a certain state (such as an error condition), it is important that the program do something such as enter an infinite loop, rather than calling the system routine `sleep()` (see *sleep*(3C)). A sleeping program cannot be adopted correctly by the debugger, although a suspended process (i.e., blocked on a read) can be.

When using the `-s` command-line option with **xdb** to debug shared libraries in an adopted process, prepare the *executable_file* by executing:

```
pxdb -s on  executable_file
```

Once the file is prepared for debugging, run *executable_file* in the background and adopt it using:

```
xdb -s -P  process_ID executable_file
```

The syntax for this use of the **pxdb** command is:

`pxdb -s [ on | enable ]` *file*
: Enables shared library debugging of the adopted process by setting private data switches within the *file*.

`pxdb -s [ off | disable ]` *file*
: Disables shared library debugging of the adopted process by clearing private data switches within the *file*.

`pxdb -s [ status ]` *file*
: This command reports whether: shared-library debugging is enabled or disabled, symbolic-debugging information is present, or symbolic-debug information has already been preprocessed. *file* is not changed when the `status` option is given. If all three conditions are true, an exit value of 0 is returned; otherwise 1.

Note that for the `on` or `off` options, *file* must be writable by the user.

WARNINGS

The debugger does not terminate on an interrupt (`SIGINT`); but jumps instead to its main loop and awaits another command. However, this does not imply that sending the debugger an interrupt is harmless. It can result in internal tables being left in an inconsistent state that could produce incorrect behavior.

Code that is not compiled debuggable or does not have a corresponding source file is dealt with in a half-hearted manner. The debugger shows `unknown` for unknown file and procedure names, cannot show code locations or interpret parameter lists, etc. However, the linker symbol table provides procedure names for most procedures, even if they are not debuggable.

On some systems, if the debugger is run on a shared *objectfile* you cannot set breakpoints. (This may only apply if someone else is also executing the program.) This may be indicated by the error "Bad access" when you attempt to start a child process. If another person starts running *objectfile* while you are debugging, they and you may have some interesting interactions.

The debugger will probably be unusable on systems that have been booted from something other than `/hp-ux` (such as if `SYSBCKUP` was booted instead on a Series 300/400 system).

The debugger has no knowledge about or control over child processes forked in turn by the process being debugged. Programs being debugged should not execute a different program via `exec()` without a `fork()` (see *exec*(2) and *fork*(2)).

Child process output may be (and usually is) buffered. Hence it may not appear immediately after you step through an output statement such as `printf()` (see *printf*(3S)). It may not appear at all if you kill the process.

If the *address* given to a `ba` command is not a code address in the child process, meaningless results or errors may ensue.

Single stepping floating-point instructions may show delayed results for operations that are actually emulated via exception traps (e.g. `fsin` on the Series 300/400 MC68040 processor). Actual results will not be apparent until the next floating-point operation is performed.

Debugging dynamically loaded code is inherently difficult, since no symbols within it are known to the debugger.

If you set the address printing format to something *printf*(3S) does not like, you may get an error (usually memory fault) each time you try to print an address, until you fix the format with another `f` command.

Do not use the `z` command to manipulate the `SIGTRAP` signal. This signal is used by the debugger to synchronize with and control the traced process, and unpredictable results may occur if it is otherwise manipulated. A corrolary to this is that applications that make use of the `SIGTRAP` signal are at best difficult to debug.

If you single step or run with assertions through a call to `longjmp()` (see *setjmp*(3C)), the child process will probably take off free-running because the debugger sets, but never hits, an up-level breakpoint.

Do not modify any file while the debugger has it open. If you do, the debugger gets confused and may display garbage.

Although the debugger tries to do things reasonably, it is possible to confuse the recording mechanism. Be careful about trying to play back from a file currently open for recording, or vice versa; strange things can happen.

The output of some program generators such as `yacc` have compiler-line-number directives in them that can confuse the debugger (see *yacc*(1)). It expects source line entries in the symbol table to appear in sorted order.

DEPENDENCIES

Series 300/400

The `lc` command is not supported on the Series 300 or 400.

Series 700/800 (PA-RISC)

All programs are shared executables. This implies three limitations. You cannot set breakpoints or single step a program if another process is running it; the error message `Bad access to child process` results. If debugging a program, and another process starts to run the same program, either through your process executing a `fork()`, or another process, such as a shell, executing an `exec()`, this second process can hit one of your breakpoints and generate a SIGTRAP. You cannot single step through a call to `fork()`.

AUTHOR

`xdb` was developed by HP and Third Eye Software.

FILES

`a.out`	Default *objectfile* to debug.
`core`	Default *corefile* to debug.
`/usr/lib/xdb.help`	Text file listed by the `help` command.
`/usr/lib/xdb.help.nro`	Unformatted text file used to generate xdb.help.
`/usr/lib/end.o`	Auxiliary object file (support module) to link with all debuggable programs.
`/usr/lib/nls/$LANG/xdb.cat`	The xdb message catalog.
`/usr/lib/nls/$LANG/pxdb.cat`	The pxdb message catalog.
`/usr/lib/xdb_demos/*`	Demo files.
`$HOME/.xdbrc`	The xdb startup command file.

SEE ALSO

adb(1), cc(1), echo(1), fc(1), ksh(1), ld(1), more(1), pc(1), creat(2), exec(2), fork(2), open(2), ptrace(2), ecvt(3C), multibyte(3C), printf(3S), setjmp(3C), shl_load(3X), system(3S), a.out(4), core(4), user(4), lang(5), signal(5).

HP-UX Symbolic Debugger User's Guide
HP-UX Symbolic Debugger Quick Reference.

NAME

xstr - extract strings from C programs to implement shared strings

SYNOPSIS

`xstr` [-c] [-] [*file*]

DESCRIPTION

`xstr` maintains a file `strings` into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, which are most useful if they are also read-only.

The command:

`xstr -c` *name*

extracts the strings from the C source in *name*, replacing string references with expressions of the form (`&xstr[`*number*`]`) for some *number*. An appropriate declaration of `xstr` is placed at the beginning of the file. The resulting C text is placed in the file `x.c`, for subsequent compiling. The strings from this file are placed in the `strings` database if they are not there already. Repeated strings and strings that are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled, a file `xs.c` declaring the common `xstr` space, can be created by the command:

```
xstr
```

This `xs.c` file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared), saving space and swap overhead.

`xstr` can also be used on a single file. A command:

`xstr` *name*

creates files `x.c` and `xs.c` as before, without using or affecting any `strings` file in the same directory.

It may be useful to run `xstr` after the C preprocessor if any macro definitions yield strings or if there is conditional code containing strings that are not, in fact, needed. `xstr` reads from its standard input when the argument - is given. An appropriate command sequence for running `xstr` after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

`xstr` does not touch the file `strings` unless new items are added, thus `make` can avoid remaking `xs.o` unless truly necessary (see *make*(1)).

AUTHOR

`xstr` was developed by the University of California, Berkeley.

FILES

`strings`	Data base of strings
`x.c`	Massaged C source
`xs.c`	C source for definition of array `xstr`
`/tmp/xs*`	Temp file when 'xstr name' does not touch `strings`

WARNINGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by `xstr`, both strings are placed in the data base, when placing only the longer one there would be sufficient.

SEE ALSO

mkstr(1).

NAME

yacc - yet another compiler-compiler

SYNOPSIS

yacc [**-vdlt**] [**-N**<*secondary*><*n*> ...] [**-p** *sym_prefix*] [**-b** *file_prefix*] *grammar*

DESCRIPTION

yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LALR(1) parsing algorithm as described in popular compiler-construction literature. Ambiguous grammar is allowed; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program **yyparse**. This program must be loaded with the lexical analyzer program, **yylex**, as well as **main** and **yyerror**, an error handling routine. These routines must be supplied by the user; **lex** is useful for creating lexical analyzers usable by **yacc** (see *lex*(1)).

Options

yacc recognizes the following options:

-v Prepare file **y.output** containing a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

-d Generate file **y.tab.h** with the **#define** statements that associate the **yacc**-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

-l Do not place any **#line** constructs in code produced in **y.tab.c**. Generally, this option should only be used after **y.tab.c** has compiled successfully, since the **#line** directives allow the C compiler to give error messages that refer to the **yacc** source file rather than the **y.tab.c** file. This option is useful, however, for symbolic debugging, since some symbolic debuggers may be confused by line numbers that are not in order.

-t Compile runtime debugging code (see Debugging below for details).

-N<*secondary*><*n*>
Allow the sizes of certain internal **yacc** tables to be reset. *secondary* is one of the letters from the set {**B a m s p n e c l w**} and specifies the table; *n* is the new size. Tables that can be reset by using secondary letters are as follows:

a	a-array size; default is 12 000.
m	mem array size; default is 12 000.
s	number of states; default is 1000.
p	number of productions; default is 800.
n	number of non-terminals; default is 600.
e	temp-space size; default is 1250.
c	name-space size; default is 5000.
l	look-ahead set table size; default is 650.
w	working set table size; default is 650.

-p *sym_prefix*
Use *sym_prefix* instead of **yy** as the prefix for externally scoped variable and function names produced by **yacc**. Names affected are: **yyparse**, **yylex**, **yyerror**, **yylval**, **yychar**, **yydebug**, and **yynerrs**.

-b *file_prefix*
Use *file_prefix* instead of **y** as the prefix for output filenames. Names affected are: **y.tab.c**, **y.tab.h**, and **y.output**.

If an array overflows, **yacc** issues a fatal error message including a suggestion of which table to reset. For example:

```
too many states, try -Ns option
```

Debugging

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when **yacc**'s **-t** option is used, this debugging code is compiled by default. Independent of whether the `-t` option was used, the runtime debugging code is under the control of `YYDEBUG`, a pre-processor symbol. If `YYDEBUG` has a non-zero value, the debugging code is included. If its value is zero, the code is not included. The size and execution time of a program produced without the runtime debugging code is smaller and slightly faster.

EXTERNAL INFLUENCES

Environment Variables

`LC_CTYPE` determines the classification of characters as letters and digits for name fields. If `LC_CTYPE` indicates a multi-byte character set is in use, intermixed ASCII and 16-bit characters can be used in non-reserved token names, terminal symbols, non-terminal symbols, strings and comments.

`LC_MESSAGES` determines the language in which messages are displayed.

`LANG` is used as a default if `LC_CTYPE` or `LC_MESSAGES` is not set.

International Code Set Support

Single- and multi-byte character code sets are supported.

ERRORS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the `y.output` file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

FILES

`y.output`	
`y.tab.c`	
`y.tab.h`	defines for token names
`yacc.tmp,`	
`yacc.acts, yacc.debug`	temporary files
`/usr/lib/yaccpar`	parser prototype for C programs

WARNINGS

File names are fixed. Therefore, only one `yacc` process can be active in a given directory at any given time.

The maximum number of terminal symbols is fixed at 2000 and cannot be reset using the `-N` option.

Even though `yacc` is able to accept and process 16-bit characters, C compilers that are used to compile the generated `y.tab.c` and `y.tab.h` files may not. Using the `-d` option to generate a `y.tab.h` file with %token definitions may result in an un-compilable file if multi-byte characters are used in token names.

SEE ALSO

lex(1), setlocale(3C), malloc(3C).

YACC - Yet Another Compiler Compiler tutorial in *C Programming Tools* manual.

Compilers - Principles, Techniques, and Tools by Aho, Sethi, and Ullman; Addison-Wesley ISBN 0-201-10088-6.

STANDARDS CONFORMANCE

`yacc`: SVID2, XPG2, XPG3, POSIX.2

NAME

yes - be repetitively affirmative

SYNOPSIS

yes [*expletive*]

DESCRIPTION

yes repeatedly outputs **y**, or if *expletive* is given, the *expletive* is output repeatedly. Termination is by interrupt.

AUTHOR

yes was developed by the University of California, Berkeley.

NAME

ypcat - print all values in Network Information Service map

SYNOPSIS

`ypcat` [`-k`][`-t`][`-d` *domain*] *mname*
`ypcat -x`

DESCRIPTION

`ypcat` prints all values in a Network Information Service (NIS) map specified by *mname*, which can be either a *mapname* or a map *nickname*. A map *nickname* is a synonym by which a NIS map can be referenced. Values are listed, one per line.

Options

`ypcat` recognizes the following options:

`-k`	Print the associated key preceding each value. This option is useful for examining maps in which the values are null or the keys are not part of the value, such as the *ypservers* map. The maps derived from files that have an ASCII version in `/etc` (such as `passwd` and `hosts`) are not in this category.
`-t`	Inhibit the translation of a map's *nickname* to its corresponding *mapname*. For example, `ypcat -t passwd` fails because there is no map named `passwd`, whereas `ypcat passwd` translates to `ypcat passwd.byname`.
`-d`	Specify a *domain* other than the one returned by `domainname` (see *domainname*(1)).
`-x`	Display the table that lists the *nickname* for each NIS map.

AUTHOR

`ypcat` was developed by Sun Microsystems, Inc.

EXAMPLES

Display the network-wide password database whose *mapname* is `passwd.byname` and *nickname* is `passwd`:

```
ypcat passwd
```

SEE ALSO

domainname(1), ypmatch(1), ypserv(1M), ypfiles(4).

INTERNATIONAL SUPPORT

8-bit data, messages.

NAME

ypmatch - print values of selected keys in Network Information Service map

SYNOPSIS

`ypmatch` [`-k`] [`-t`] [`-d` *domain*] *key* ... *mname*
`ypmatch -x`

DESCRIPTION

`ypmatch` prints the values associated with one or more keys in a Network Information Service (NIS) map specified by *mname*. The *mname* can be either a *mapname* or a map *nickname*. A map *nickname* is a synonym by which a NIS map can be referenced.

If multiple keys are specified, the same map is searched for an occurrence of each key. A match is made only when the case and length of a key is the same as that stored in the database. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

Options

`ypmatch` recognizes the following command-line options:

`-k` Before printing the value associated with a key, print the key followed by a colon (`:`). This option is useful if the keys are not part of the values (as in a `ypservers` map), or so many keys were specified that the output could be confusing.

`-t` Inhibit the translation of a map's *nickname* to its corresponding *mapname*. For example, `ypmatch -t zippy passwd` fails because there is no map named `passwd`, while `ypmatch zippy passwd` is translated to `ypmatch zippy passwd.byname`.

`-d` Specify a *domain* other than the one returned by `domainname` (see *domainname*(1)).

`-x` Display the table that lists the *nickname* for each NIS map.

AUTHOR

`ypmatch` was developed by Sun Microsystems, Inc.

SEE ALSO

domainname(1), ypcat(1), ypserv(1M), ypfiles(4).

INTERNATIONAL SUPPORT

8-bit data, messages.

NAME

yppasswd - change login password in Network Information System

SYNOPSIS

yppasswd [*name*]

DESCRIPTION

yppasswd changes or installs a password associated with the login *name* in the Network Information System (NIS). The NIS password can be different from the one on your own machine. If *name* is omitted, it defaults to the name returned by **getlogin()** (see *getlogin*(3C)).

yppasswd prompts for the old NIS password (even if it does not exist), then twice for the new one. The old password must be entered correctly for the change to take effect. Checks occur to ensure that the new password meets the following construction requirements.

- Only the first eight characters are significant.
- A password can be as few as four characters long if it contains
    - at least one special character or
    - a mixture of numeric, uppercase and lowercase letters.
- A password can be as few as five characters long if it contains a mixture of
    - uppercase and lowercase letters or
    - numeric and either uppercase or lowercase letters.
- A password must contain at least six characters if it contains only monocase letters.

All these rules except the first are relaxed if you try three times to enter an unacceptable new password. You cannot, however, enter a null password.

Only the owner of the *name* or the super-user can change a password.

The Network Information System password daemon, *yppasswdd*(1M), must be running on the master NIS passwd server to change NIS passwords.

WARNINGS

The password update protocol passes the old and new passwords to the master NIS server at once. Thus, if the old NIS password is incorrect, no notification is given until the new NIS password is successfully entered.

The **yppasswd** password construction rules are different from those of the HP-UX **passwd** command (see *passwd*(1)).

Password aging, as described in *passwd*(1) and *passwd*(4), is not implemented in the Network Information System password database.

User applications that call this routine must be linked with **/usr/include/librpcsvc.a**. For example,

```
    cc my_source.c -lrpcsvc
```

AUTHOR

yppasswd was developed by Sun Microsystems, Inc.

SEE ALSO

id(1), passwd(1), su(1), yppasswdd(1M), getlogin(3C), yppasswdd(3N), ypfiles(4).

INTERNATIONAL SUPPORT

8-bit data, messages

NAME

ypwhich - list which host is Network Information System server or map master

SYNOPSIS

`ypwhich`
`ypwhich` [`-d` *domain*] [`-V1` | `-V2`] [*hostname*]
`ypwhich` [`-d` *domain*] [`-t`] [`-m` [*mname*]]
`ypwhich -x`

DESCRIPTION

`ypwhich` lists the host name of the Network Information System (NIS) server that supplies NIS services to a NIS client. It can also print the NIS server that is the master for *mname*. The *mname* can be either a *mapname* or a map *nickname*. A map *nickname* is a synonym by which a NIS map can be referenced.

If invoked without arguments, `ypwhich` prints the host name of the NIS server serving the local machine. If *hostname* is specified, that machine is queried to determine which NIS server it is using.

Options

`ypwhich` recognizes the following command-line options and arguments:

`-d` Specify a *domain* other than the one returned by *domainname*(1).

`-V1` List the server that is serving Version 1 NIS protocol-speaking client processes.

`-V2` List the server that is serving Version 2 NIS protocol-speaking client processes.

If neither version is specified, `ypwhich` locates the server supplying the Version 2 (current) services. However, if no Version 2 server is found, `ypwhich` attempts to locate the server supplying the Version 1 services. Since NIS servers and NIS clients are both backward compatible, the user seldom needs to know which version is being used.

`-t` Inhibit the translation of a map's *nickname* to its corresponding *mapname*. For example, `ypwhich -t -m passwd` fails because there is no map named `passwd`, whereas `ypwhich -m passwd` translates to `ypwhich -m passwd.byname`. This option is useful if a *mapname* is identical to a *nickname* (which is not true of any HP map).

`-m` [*mname*]
List the master NIS server for a map. No *hostname* can be specified with `-m`. The *mname* can be a *mapname* or a map *nickname*. If *mname* is omitted, a complete list of available maps and the corresponding host names of the master NIS servers is produced.

`-x` Display the table that lists the *nickname* for each NIS map.

AUTHOR

`ypwhich` was developed by Sun Microsystems, Inc.

SEE ALSO

domainname(1), ypserv(1M), ypset(1M), ypfiles(4).

INTERNATIONAL SUPPORT

8-bit data, messages.

Section 9: Glossary

NAME

intro - introduction to glossary section

DESCRIPTION

This section contains a glossary of common HP-UX terms. References to other HP-UX documentation are included as appropriate. References to entities such as *wait*(2), *sh*(1), or *fopen*(3S) refer to entries in the other sections of this manual. References to items in italics but having no parenthetical suffixes refer to other entries in this glossary. Any italicized manual names refer to separate manuals that are either included with your system or available separately.

The definitions specifically reflect the HP-UX operating system, although some terms and definitions are also derived from those in the emerging IEEE POSIX standards and the *X/Open Portability Guide.* Differences in wording exist to more specifically reflect characteristics of the HP-UX system.

SEE ALSO

The introduction to this manual.

HP-UX and other UNIX-like systems use a specialized vocabulary in which certain words and terms have very specific meanings. This glossary is intended as an aid in promoting exactness in use of these specialized terms whose meanings sometimes differ from those that might be encountered in other environments.

.o ("dot-oh") The suffix customarily given to a relocatable object file. The term, ".o file," is sometimes used to refer to a relocatable object file; the format of such files is sometimes called ".o format". See *a.out*(4).

a.out The name customarily given to an executable object code file on HP-UX. The format is machine-dependent, and is described in *a.out*(4) for each implementation. Object code that is not yet linked has the same format, but is referred to as a *.o* ("dot-oh") file. *a.out* is also the default output file name used by the linker, *ld*(1).

absolute path name
A path name beginning with a slash (/). It indicates that the file's location is given relative to the root directory (/), and that the search begins there.

access Access concerns the process of obtaining data from or placing data in storage, or the right to use system resources. Accessibility is governed by three process characteristics: the effective user ID, the effective group ID, and the group access list. The *access*(2) system call determines accessibility of a file according to the bit pattern contained in its *amode* parameter, which is constructed to read, write, execute or check the existence of a file. The *access*(2) system call uses the *real user ID* instead of the *effective user ID* and the *real group ID* instead of the *effective group ID*.

access groups The group access list is a set of *supplementary group IDs* used in determining resource accessibility. Access checks are performed as described below in *file access permissions*.

access mode An access mode is a form of access permitted to a file. Each implementation provides separate read, write, and execute/search access modes.

address A number used in information storage or retrieval to specify and identify memory location. An *address* is used to mark, direct, indicate destination, instruct or otherwise communicate with computer elements.

In mail, *address* is a data structure whose format can be recognized by all elements involved in transmitting information. On a local system, this might be as simple as the user's *login* name, while in a networked system, *address* specifies the location of the resource to the network software.

In a text editor (such as *vi*, *ex*, *ed*, or *sed*), an *address* locates the line in a file on which a given instruction is intended.

For *adb*, the *address* specifies at what assembly-language instruction to execute a given command.

In disk utilities such as *fsdb*, *address* might refer to a raw or *block special file,the inode* number, *volume header*, or other file attribute.

In the context of peripheral devices, *address* refers to a set of values that specify the location of an I/O device to the computer. The exact details of the formation of an address differ between systems. On Series 300/400 systems, the address is composed of up to four elements: the *select code*, *bus address*, *unit number (ID)*, and *volume number (ID)*. On Series 700 systems, the address consists of up to two elements: the *select code*, and the *function number*.

address space The range of memory locations to which a process can refer.

affiliation See *terminal affiliation*.

appropriate privileges
Each implementation provides a means of associating privileges with a process, for function calls and function call options requiring special privileges. In HP-UX, *appropriate privileges* refers either to superuser status or to a privilege associated with privilege groups (see *setprivgrp*(1M)).

archive A file comprised of the contents of other files, such as a group of object files (that is, **.o**) used by the linker, *ld*(1)). An archive file is created and maintained by *ar*(1) or similar programs, such as *tar*(1) or *cpio*(1). An *archive* is often called a *library*.

ASCII An acronym for American Standard Code for Information Interchange. ASCII is the traditional System V coded character set and defines 128 characters, including both control characters and graphic characters, each of which is represented by 7-bit binary values ranging from 0 through 127 decimal.

background process group
Any process group that is a member of a session which has established a connection with a controlling terminal that is not in the foreground process group.

backup The process of making a copy of all or part of the file system in order to preserve it, in case a system crash occurs (usually due to a power failure, hardware error, etc.). This is a highly recommended practice.

block (1) The fundamental unit of information HP-UX uses for access and storage allocation on a mass storage medium. The size of a block varies between implementations and between file systems. In order to present a more uniform interface to the user, most system calls and utilities use *block* to mean 512 bytes, independent of the actual block size of the medium. This is the meaning of *block* unless otherwise specified in the manual entry.

(2) On media such as 9-track tape that write variable length strings of data, the size of those strings. *Block* is often used to distinguish from *record*; a block contains several records, whereas the number of records denotes the blocking factor.

block special file
A special file associated with a mass storage device (such as a hard disk or tape cartridge drive) that transfers data in multiple-byte blocks, rather than by series of individual bytes (see *character special file*). *Block special files* can be mounted. A *block special file* provides access to the device where hardware characteristics of the device are not visible.

boot or *boot-up* The process of loading, initializing, and running an operating system.

boot area A portion of a mass storage medium on which the volume header and a "bootstrap" program used in booting the operating system reside. The *boot area* is reserved exclusively for use by HP-UX.

boot ROM A program residing in ROM (Read-Only Memory) that executes each time the computer is powered up and is designed to bring the computer to a desired state by means of its own action. The first few instructions of a bootstrap program are sufficient to bring the remainder of the program into the computer from an input device and initiate functions necessary for computation. The function of the boot ROM is to run tests on the computer's hardware, find all devices accessible through the computer, and then load either a specified operating system or the first operating system found according to a specific search algorithm.

bus address A number which makes up part of the address HP-UX uses to locate a particular device. The *bus address* is determined by a switch setting on a peripheral device which allows the computer to distinguish between two devices connected to the same interface. A *bus address* is sometimes called a "device address".

character An element used for the organization, control, or representation of text. Characters include *graphic characters* and *control characters.*

character set A set of characters used to communicate in a native or computer language.

character special file
A special file associated with I/O devices that transfer data byte-by-byte. Other byte-mode I/O devices include printers, nine-track magnetic tape drives, and disk drives when accessed in "raw" mode (see *raw disk*). A *character special file* has no predefined structure.

child process A new process created by a pre-existing process via the *fork*(2) system call. The new process is thereafter known to the pre-existing process as its *child process*. The pre-existing process is the *parent process* of the new process. See *parent process* and *fork*.

clock tick

A rate used within the system for scheduling and accounting. It consists of the number of intervals per second as defined by {CLK_TCK} that is used to express the value in type **clock_t**. {CLK_TCK} was previously known as the defined constant HZ.

cluster

A set of one or more systems, connected by a network, that share a single *file hierarchy*.

This term applies to implementations that include the HP-UX Clustered Environment.

cluster node (cnode)

One of the nodes, or systems, in a *cluster*.

This term applies to implementations that include the HP-UX Clustered Environment.

cluster server

The node in a *cluster* to which the storage device containing the root file system of the *cluster* is physically attached, and which acts as a server to client systems in the cluster. Also referred to as the *root server* (defined below).

cluster server process (CSP)

A special kernel process that runs on a machine in a *cluster* to satisfy requests from other nodes in the cluster. There are two kinds of CSP, the "limited CSP" (LCSP) and the "general CSP" (GCSP). Each node in a cluster runs exactly one LCSP that is necessary to maintain the cluster itself. In addition, the *cluster server* runs some number of GCSPs, which satisfy user requests for remotely available resources such as files and swap space.

This term applies to implementations that include the HP-UX Clustered Environment.

cnode ID

A numeric value used to identify each node in a *cluster*. Each *cnode ID* is unique within the cluster.

This term applies to implementations that include the HP-UX Clusted Environment.

cnode name

A character string used to identify each node in a *cluster*. Each *cnode name* is unique within the cluster.

This term applies to implementations that include the HP-UX Clusted Environment.

coded character set

A set of unambiguous rules that establishes a character set and the one-to-one relationship between each character of the set and its corresponding bit representation. *ASCII* is a *coded character set.*

collating element

The smallest entity used in collation to determine the logical ordering of strings (that is, the *collation sequence*). To accommodate native languages, a collating element consists of either a single character, or two or more characters collating as a single entity. The current value of the *LANG* environment variable determines the current set of collating elements.

collation

The logical ordering of strings in a predefined sequence according to rules established by precedence. These rules identify a collation sequence among the collating elements and also govern the ordering of strings consisting of multiple collating elements, to accommodate native languages.

collation sequence

The ordering sequence applied to *collating elements* when they are sorted. To accommodate native languages, *collation sequence* can be thought of as the relative order of *collating elements* as set by the current value of the *LANG* environment variable. Characters can be omitted from the collation sequence, or two or more collating elements can be given the same relative order (see *string*(3C)).

command

A directive to perform a particular task. HP-UX commands are executed through a *command interpreter* called a *shell*. HP-UX supports several shells, including *sh*(1), *csh*(1), *ksh*(1), and *pam*(1). Most commands are carried out by an executable file, called a *utility*, which might take the form of a stand-alone unit of executable object code (a program) or a file containing a list of other programs to execute in a given order (a shell script). Scripts can contain references to other scripts, as well as to object-code programs. A typical *command* consists of the utility name followed by arguments that are passed to the utility. For example, in the command, "ls mydirectory," "ls" is the utility name and "mydirectory" is an argument passed to the "ls" utility.

command interpreter
: A program which reads lines of text from standard input (typed at the keyboard or read from a file), and interprets them as requests to execute other programs. A command interpreter for HP-UX is called a *shell*. See *sh*(1), *csh*(1), *ksh*(1), and *pam*(1).

composite graphic symbol
: A graphic symbol consisting of a combination of two or more other graphic symbols in a single character position, such as a diacritical mark and a basic letter.

context
: A set of character strings associated with each process, that ordinarily determines which element (if any) from any given *context-dependent file* is accessed by that process. The *context* contains elements such as *cnode*name, the hardware attributes of the *cnode*(see *cluster node*), and possibly others. See *context*(5) for a complete list of items that can appear in the context.

  This term applies to implementations that include the HP-UX Clustered Environment.

context-dependent file (CDF)
: A set of files, all associated with the same path name. The *context* of a process ordinarily determines which element (if any) from any given CDF will be accessed by that process. (See *cdf*(4)).

  This term applies to implementations that include the HP-UX Clustered Environment.

control character
: A character other than a graphic character that affects the recording, processing, transmission, or interpretation of text. In the *ASCII* character set, *control characters* are those in the range 0 through 31, and 127. Control characters can be generated by holding down [CTRL], [CONTROL], or [CNTL] (depending on what the control key is labeled on your terminal) and pressing a character key (as you would use SHIFT). These two-key sequences are often written as ctrl-d, for example, or ^D, where ^ stands for the control key. Both representations assume that the control key is held down while the second key is pressed.

controlling process
: The session leader that establishes the connection to the *controlling terminal*. Should the terminal subsequently cease to be a controlling terminal for this session, the session leader ceases to be the controlling process.

controlling terminal
: A terminal that is associated with a session. Each session can have at most one controlling terminal associated with it and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal cause signals to be sent to all processes in the foreground process group associated with the controlling terminal.

CS/80 or *CS-80*
: A family of mass storage devices that communicate with the controlling computer by means of a series of commands and data transfer protocol referred to as the *CS/80* (Command Set 1980) command set. This command set was implemented in order to provide better forward/backward compatibility between models and generations of mass storage devices as technological advances develop. Some mass storage devices support only a subset of the full *CS/80* command set, and are usually referred to as *SS/80* (SubSet 1980) devices.

crash
: The unexpected shutdown of a program or system. If the operating system crashes, this is a "system crash", and requires the system to be re-booted.

current directory
: See *working directory*.

current working directory
: See *working directory*.

daemon
: A process which runs in the background, and which is usually immune to termination instructions from a terminal. Its purpose is to perform various scheduling, clean-up, and maintenance jobs. *lpsched*(1M) is an example of a *daemon*. It exists to perform these

functions for line printer jobs queued by *lp*(1). An example of a permanent *daemon* (that is, one that should never die) is *cron*(1M).

data encryption
A method for encoding information in order to protect sensitive or proprietary data. For example, HP-UX automatically encrypts all users' passwords. The encryption method used by HP-UX converts ASCII text into a base-64 representation using the alphabet **.**, **/**, **0-9**, **A-Z**, **a-z**. See *passwd*(4) for the numerical equivalents associated with this alphabet.

default search path
The sequence of directory prefixes that *sh*(1), *time*(1), and other HP-UX commands apply in searching for a file known by an relative path name (that is, a path name not beginning with a *slash* (/)). It is defined by the environment variable **PATH** (see *environ*(5)). *login*(1) sets **PATH** equal to **:/bin:/usr/bin**, which means that your working directory is the first directory searched, followed by **/bin,followed**by */usr/bin*. You can redefine the search path by modifying the value of **PATH**. This is usually done in **/etc/profile**, and/or in the **.profile** file found in the home directory.

delta A term used in the Source Code Control System (SCCS) to describe a unit of one or more textual changes to an SCCS file. Each time you edit an SCCS file, the changes you make to the file are stored separately as a *delta*. Then, using the *get*(1) command, you can specify which deltas are to be applied to or excluded from the SCCS file, thus yielding a particular version of the file. Contrast this with the *vi* or *ed* editor, which incorporates your changes into the file immediately, prohibiting you from obtaining a previous version of that file. See *SCCS SCCS file*.

demon See *daemon*.

device A computer peripheral or an object that appears to an application as such.

device address See *bus address*.

device file See *special file*.

directory A file that provides the mapping between the names of files and their contents, and is manipulated by the operating system alone. For every file name contained in a directory, that directory contains a pointer to the file's *inode*; The pointer is called a *link*. A file can have several links appearing anywhere on the same file system. Each user is free to create as many directories as needed (using *mkdir*(1)), provided that the *parent directory* of the new directory gives the permission to do so. Once a directory has been created, it is ready to contain ordinary files and other directories. An HP-UX directory is named and behaves exactly like an ordinary file, with one exception: no user (including the super-user) is allowed to write data on the directory itself; this privilege is reserved for the HP-UX operating system.

By convention, a directory contains at least two links, **.** and **..**, referred to as *dot* and *dot-dot* respectively. *Dot* refers to the directory itself and *dot-dot* refers to its *parent directory*. A directory containing only **.** and **..** is considered empty.

dot In any directory, the special file name, *dot*, consisting of a single dot character ("."), refers to the current directory. It can be used alone or at the beginning of a directory path name. (Also see *path name resolution*.) The *dot* also functions as a special command in the Bourne and Korn shells, and has special meaning in text editors and formatters, in parsing regular expressions and in designating file names.

dot-dot The special file name, *dot-dot* (".."), refers to the *parent directory*. If it begins a *path name*, *dot-dot* refers to the parent of the current directory. If it occurs in a path name, *dot-dot* refers to the parent directory of the directory preceding *dot-dot* in the path name string. As a special case, *dot-dot* refers to the current directory in any directory that has no parent (most often, the root directory). (Also see *path name resolution*.)

downshifting The conversion of an uppercase character to its lowercase representation.

dynamic loader A routine invoked at process startup time that loads shared libraries into a process' address space. The dynamic loader also resolves symbolic references between a program and the shared libraries, and initializes the shared libraries' linkage tables. See *dld.sl*(5) for details.

effective group ID
Every process has an *effective group ID* that is used to determine *file access permissions*. A process's *effective group ID* is determined by the file (command) that process is executing. If that file's set-group-ID bit is set (located in the mode of the file, see *mode*), the process's *effective group ID* is set equal to the file's group ID. This makes the process appear to belong to the file's group, perhaps enabling the process to access files that must be accessed in order for the program to execute successfully. If the file's set-group-ID bit is not set, the process's *effective group ID* is inherited from the process's parent. The setting of the process's *effective group ID* lasts only as long as the program is being executed, after which the process's effective group ID is set equal to its real group ID. See *group*, *real group ID*, and *set-group-ID bit*.

effective user ID
A process has an *effective user ID* that is used to determine *file access permissions* (and other permissions with respect to system calls, if the effective user ID is 0, which means super-user). A process's effective user ID is determined by the file (command) that process is executing. If that file's set-user-ID bit is set (located in the mode of the file, see *mode*), the process's effective user ID is set equal to the file's user ID. This makes the process appear to be the file's owner, enabling the process to access files which must be accessed in order for the program to execute successfully. (Many HP-UX commands which are owned by *root*, such as *mkdir* and *mail*, have their set-user-ID bit set so other users can execute these commands.) If the file's set-user-ID bit is not set, the process's effective user ID is inherited from that process's parent. See *real user ID* and *set-user-ID bit*.

end-of-file (EOF)
(1) The data returned when attempting to read past the logical end of a file via *stdio*(3S) routines. In this case end-of-file is not properly a character.

(2) The character [CTRL]-[D].

(3) A character defined by *stty*(1) or *ioctl*(2) (see *termio*(7)) to act as end-of-file on your terminal. Usually this is [CTRL]-[D].

(4) The return value from *read*(2) that indicates end of data.

environment
The set of defined shell variables (some of which are PATH, TERM, SHELL, EXINIT, HOME) that define the conditions under which your commands run. These conditions can include your terminal characteristics, home directory, and default search path. Each shell variable setting in the current process is passed on to all *child processes* that are created, provided that each shell variable setting has been exported via the *export* command (see *sh*(1)). Unexported shell variable settings are meaningful only to the current process, and any child processes created get the default settings of certain shell variables by executing **/etc/profile**, **$HOME/.profile**, or **$HOME/.login**.

Epoch
The Epoch refers to the time at 0 hours, 0 minutes, 0 seconds, Coordinated Universal Time on January 1, 1970. Increments quantify the amount of time elapsed from the Epoch to the referenced time.

Leap seconds, which occur at irregular intervals, are not reflected in the count of seconds between the Epoch and the referenced time. (Fourteen leap seconds occurred in the years 1970 through 1988.)

FIFO special file (FIFO)
A type of *file*. Data written to a *FIFO* is read on a first-in-first-out basis. Other characteristics are described under *open*(2), *read*(2), *write*(2) and *lseek*(2).

file
A stream of bytes that can be written to and/or read from. A *file* has certain attributes, including permissions and type. File types include *regular file*, *character special file*, *block special file*, *FIFO special file*, *context-dependent file*, network special file, *directory*, and *symbolic link*. Every file must have a *file name* that enables the user (and many of the HP-UX commands) to refer to the contents of the file. The system imposes no particular structure on the contents of a file, although some programs do. Files can be accessed serially or randomly (indexed by byte offset). The interpretation of file contents and structure is up to

the programs that access the file.

file access mode

A characteristic of an *open file description* that determines whether the described file is open for reading, writing, or both. (See *open*(2).)

file access permissions

Every file in the *file hierarchy* has a set of access permissions. These permissions are used in determining whether a process can perform a requested operation on the file (such as opening a file for writing). Access permissions are established when a file is created via the *open*(2) or *creat*(2) system calls, and can be changed subsequently through the *chmod*(2) call. These permissions are read by *stat*(2) or *fstat*(2).

File access controls whether a file can be read, written, or executed. Directory files use the execute permission to control whether or not the directory can be searched.

File access permissions are interpreted by the system as they apply to three different classes of users: the *owner* of the file, the users in the file's *group*, and anyone else ("other"). Every file has an independent set of access permissions for each of these classes. When an access check is made, the system decides if permission should be granted by checking the access information applicable to the caller.

Read, write, and execute/search permissions on a file are granted to a process if any of the following conditions are met:

> The process's *effective user ID* is super-user.
>
> The process's *effective user ID* matches the user ID of the owner of the file and the appropriate access bit of the *owner* portion (0700) of the file mode is set.
>
> The process's *effective user ID* does not match the user ID of the owner of the file, and either the process's *effective group ID* matches the group ID of the file, or the group ID of the file is in the process's group access list, and the appropriate access bit of the *group* portion (070) of the file mode is set.
>
> The process's *effective user ID* does not match the user ID of the owner of the file, and the process's *effective group ID* does not match the group ID of the file, and the group ID of the file is not in the process's group access list, and the appropriate access bit of the "other" portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied.

file descriptor

A small unique, per-process, non-negative integer identifier that is used to refer to a file opened for reading and/or writing. Each *file descriptor* refers to exactly one *open file description*.

A *file descriptor* is obtained through system calls such as *open*(2), *creat*(2), *rdup (2), fcntl*(2) or *pipe*(2). The *file descriptor* is used as an argument by calls such as *read*(2), *write*(2), *ioctl*(2), and *close*(2).

The value of a *file descriptor* has a range from 0 to one less than the system-defined maximum. The system-defined maximum is the value NOFILE in **<sys/param.h>**.

file group class

A process is in the *file group class* of a file if the process is not the *file owner class* and if the *effective group ID* or one of the *supplementary group IDs* of the process matches the group ID associated with the file.

file hierarchy

The collection of one or more *file systems* available on a system. All *files* in these *file systems* are organized in a single hierarchical structure in which all of the non-terminal nodes are *directories*. Because multiple *links* can refer to the same *file*, the directory is properly described as a directed graph.

file name

A string of up to 14 bytes (or 255 bytes on file systems configured to support long file names) used to refer to an ordinary file, special file, or directory. The byte values zero (null) and slash (/) cannot be used as characters in a file name. Note that it is generally unwise to use *, ?, , !, or] as part of file names because the shell attaches special meaning to these characters (see *sh*(1), *csh*(1), or *ksh*(1)). Avoid beginning a file name with **-**, **+**, or **=**,

because to some programs, these characters signify that a command argument follows. The last byte of a file name should not be a **+**
due to interactions with context-dependent files on HP Clustered systems. A file name is sometimes called a path name component. Although permitted, it is inadvisable to use characters that do not have a printable graphic on the hardware you commonly use, or that are likely to confuse your terminal.

file name portability
File names should be constructed from the *portable file name character set* because the use of other characters can be confusing or ambiguous in certain contexts.

file offset The file offset specifies the position in the file where the next I/O operation begins. Each *open file description* associated with either a regular file or special file has a *file offset*. There is no file offset specified for a *pipe* or *FIFO*.

file other class A process is in the *file other class* if the process is not in the *file owner class* or *file group class*.

file owner class A process is in the *file owner class* if the *effective user ID* of the process matches the user ID of the file.

file permission bits
See *permission bits*.

file pointer A data element obtained through any of the *fopen*(3S) standard I/O library routines that "points to" (refers to) a file opened for reading and/or writing, and which keeps track of where the next I/O operation will take place in the file (in the form of a byte offset relative to the beginning of the file). After obtaining the file pointer, it must thereafter be used to refer to the open file when using any of the standard I/O library routines. (See *stdio*(3S) for a list of these routines.)

file serial number
A *file serial number* is a file-system-unique identifier for a given file, and is also known as the file's *inode number*. Each *file serial number* identifies exactly one *inode*. *File serial numbers* are not necessarily unique across *file systems* in the *file hierarchy*.

file server, cluster
In the HP Clustered environment, the cluster server cnode is a *cluster file server* for the entire cluster. An *auxiliary file server* is a cnode with a locally mounted disk. An *auxiliary file server's* locally mounted disk(s) become part of the root file system and are accessible by any member cnode.

file status flags A characteristic of an *open file description*. These flags can be used to modify the behavior of system calls that access the file described by the *open file description*.

file system A collection of *files* and supporting data structures residing on a mass storage volume. A file system provides a name space for *file serial numbers* referring to those files. Refer to the System Administrator manuals supplied with your system for details concerning file system implementation and maintenance.

file times update
Each file has three associated time values that are updated when file data is accessed or modified, or when the file status is changed. These values are returned in the file characteristics structure, as described in **<sys/stat.h>**. For each function in HP-UX that reads or writes file data or changes the file status, the appropriate time-related files are noted as "marked-for-update". When an update point occurs, any marked fields are set to the current time and the update marks are cleared. One such update point occurs when the file is no longer open for any process. Updates are not performed for files on *read-only file systems*.

filter A command that reads data from the standard input, performs a transformation on the data, and writes it to the standard output.

foreground process group
Each session that has established a connection with a controlling terminal has exactly one process group of the session as a foreground process group of that controlling terminal. The

foreground process group has certain privileges when accessing its controlling terminal that are denied to background process groups. See *read*(2) and *write*(2).

foreground process group ID
: The process group ID of the foreground process group.

fork
: An HP-UX system call (*fork*(2)) which, when invoked by an existing process, causes a new process to be created. The new process is called the *child process*; the existing process is called the *parent process* . The child process is created by making an exact copy of the parent process. The parent and child processes are able to identify themselves by the value returned by their corresponding *fork* call (see *fork*(2)*for* details).

function number
: On Series 400 and 700 systems, when two or more interfaces reside on a single interface card, each interface is treated as a separate *function* and is assigned a corresponding unique function number.

graphic character
: A character other than a control character that has a visual representation when hand-written, printed, or displayed.

group
: An association of zero or more users who must all be permitted to access the same set of files. The members of a group are defined in the files **/etc/passwd** and **/etc/logingroup** (if it exists) via a numerical group ID that must be between zero and {UID_MAX}, inclusive. Users with identical group IDs are members of the same group. An ASCII group name is associated with each group ID in the file **/etc/group**. A group ID is also associated with every file in the *file hierarchy*, and the mode of each file contains a set of permission bits that apply only to this group. Thus, if you belong to a group that is associated with a file, and if the appropriate permissions are granted to your group in the file's mode, you can access the file. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID, one of the supplementary group IDs, or a saved set-group-ID. See *real group ID*, *effective*group*ID* , *privileged group*, *supplementary group ID*, and *set-group-ID bit*.

group access list
: The group access list is a set of *supplementary group IDs* used in determining resource accessibility. Access checks are performed as described in *file access permissions*.

hidden directory
: A specially marked *directory* used to implement *context-dependent files*. (see *cdf*(4)).

  This term applies to implementations to all implementations of the HP Clustered Environment.

hierarchical directory
: A directory (or file system) structure in which each directory can contain other directories as well as files.

home directory
: The directory name given by the value of the shell variable HOME. When you first log in, *login*(1) automatically sets HOME equal to your *login directory*. You can change its value at any time, however. This is usually done in the **.profile** file contained in your *login* directory. Setting HOME in no way affects your *login directory*, but simply gives you a convenient way of referring to what should be your most commonly used directory.

host name
: An *ASCII* string of at most 8 characters (of which only 6 are supported by all the various manufacturers' UNIX-like operating systems) which uniquely identifies an HP-UX system on a *uucp*(1) network. The *host name* for your system can be viewed and/or set with the *hostname*(1) command. Systems without a defined host name are described as "unknown" on the *uucp*(1) network. Do not confuse a host name with a *node name*, which is a string that uniquely identifies an HP-UX system on a Local Area Network (LAN). Although your host and node names may be identical, they are set and used by totally different software. See *node name*.

image
: The current state of your computer (or your portion of the computer, on a multi-user system) during the execution of a command. Often thought of as a "snapshot" of the state of

the machine at any particular moment during execution.

init A *system process* that performs initialization, is the ancestor of every other process in the system, and is used to start *login* processes. *init* usually has a *process ID* of 1.

interleave factor
A number that determines the order in which sectors on a mass storage medium are accessed. It can be optimized to make data acquisition more efficient.

inode An *inode* is a structure that describes a file and is identified in the system by a *file serial number*. Every file or directory has associated with it an *inode*. Permissions that specify who can access the file and how are kept in a 9-bit field that is part of the *inode*. The *inode* also contains the file size, the user and group ID of the file, the number of links, and pointers to the disk blocks where the file's contents can be found. Each connection between an *inode* and its entry in one or more directories is called a *link*.

inode number See *file serial number*.

Internal Terminal Emulator (ITE)
The "device driver" code contained in the HP-UX kernel that is associated with the computer's built-in keyboard and display or with a particular keyboard and display connected to the computer, depending on the Series and Model of system processor. See *system console* and the System Administrator manuals supplied with your system for details.

internationalization
The concept of providing software with the ability to support the *native language*, *local customs*, and *coded character set* of the user.

interrupt signal
The signal sent by SIGINT (see *signal*(2)). This signal generally terminates whatever program you are running. The key which sends this signal can be redefined with *ioctl*(2) or *stty*(1) (see *termio*(7)). It is often the ASCII DEL (rubout) character (the [DEL] key) or the [Break] key. [Ctrl]-[C] is often used instead.

intrinsic See *system call*.

I/O redirection A mechanism provided by the HP-UX shell for changing the source of data for standard input and/or the destination of data for standard output and standard error. See *sh*(1).

job control Job control allows users to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point.

The user employs this facility via the interactive interface jointly supplied by the system *tty* driver and either *csh*(1) or *ksh*(1). The *tty* driver recognizes a user-defined "suspend character" which causes the current foreground process group to stop and the user's job control shell to resume. The job control shell provides commands that continue stopped process groups in either the foreground or background. The *tty* also stops a background process group when any member of the background process group attempts to read from or write to the user's terminal. This allows the user to finish or suspend the *foreground process group* without interruption and continue the stopped *background process group* at a more convenient time.

See *csh*(1) and *stty*(1) for usage and installation details, and *csh*(1), *signal*(2), and *termio*(7) for implementation details.

kernel The HP-UX operating system. The kernel is the executable code responsible for managing the computer's resources, such as allocating memory, creating processes, and scheduling programs for execution. The kernel resides in RAM (Random Access Memory) whenever HP-UX is running.

LANG An environment variable that is used to inform a computer process of the user's requirements for *native language*, *local customs*, and *coded character set*.

library A file containing a set of subroutines and variables that can be accessed by user programs. Libraries can be either archives or shared libraries. For example, **/lib/libc.a** and **/lib/libc.sl** are libraries containings all functions of section (2) and all functions of section (3) that are marked (3C) and (3S) in the *HP-UX Reference*. Similarly, **/lib/libm.a** and

	/lib/libm.sl are libraries containing all functions in section (3) that are marked (3M) in the *HP-UX Reference.* See *intro*(3).
LIF	An acronym for Logical Interchange Format. A standard format for mass storage implemented on many Hewlett-Packard computers to aid in media transportability. The *lif**(1) commands are used to perform various functions using LIF.
line	A sequence of text characters consisting of zero or more non-new-line characters plus a terminating new-line character.
link	*Link* is a synonym for *directory entry*. It is an object that associates a file name with any type of file. The information constituting a *link* includes the name of the file and where the contents of that file can be found on a mass storage medium. One physical file can have several links to it. Several directory entries can associate names with a given file. If the links appear in different directories, the file may or may not have the same name in each. However, if the links appear in one directory, each link must have a unique name in that directory. Multiple links to directories are not allowed (except as created by a user with appropriate privileges). See *ln*(1), *link*(2), *unlink*(2),*and symbolic link*.
	Also, to prepare a program for execution; see *linker*.
link count	The *link count* of a file is the number of directory entries that refer to that file.
linker	The *linker* combines one or more object programs into one program, searches libraries to resolve user program references, and builds an executable file in *a.out* format. This executable file is ready to be executed through the program loader, *exec*(2). The linker is invoked with the *ld*(1) command. The linker is often called a *link editor*.
local customs	The conventions of a geographical area or territory for such things as date, time and currency formats.
localization	The process of adapting existing software to meet the local language, customs, and character set requirements of a particular geographical area.
logical block size	The smallest unit of memory that can be allocated on a Series 500 SDF volume; a multiple of the physical sector size.
login	The process of gaining access to HP-UX. This consists of successful execution of the *login* sequence defined by *login*(1) which varies depending on the system configuration. It includes providing a *login* name and possibly one or more passwords.
login directory	The directory in which you are placed immediately after you log in. This directory is defined for each user in the file **/etc/passwd** . The shell variable HOME is set automatically to your *login directory* by *login*(1) immediately after you log in. See *home directory*.
magic number	The first word of an *a.out*-format or archive file. This word contains the system ID, which states what machine (hardware) the file will run on, and the file type (executable, sharable executable, archive, etc.).
major number	A number used exclusively to create special files that enable I/O to or from specific devices. This number indicates which device driver to use for the device. Refer to *mknod*(2) and the System Administrator manual supplied with your system for details.
message catalog	Program strings, such as program messages and prompts, are stored in a *message catalog* corresponding to a particular geographical area. Retrieval of a string from a *message catalog* is based on the value of the user's LANG environment variable (see *LANG*).
message queue identifier (msqid)	A *message queue identifier* is a unique positive integer created by a *msgget*(2) system call. Each *msqid* has a message queue and a data structure associated with it. The data structure is referred to as *msqid_ds* and contains the following members:

```
struct    ipc_perm msg_perm;/* operation permission */
```

```
ushort  msg_qnum;       /* number of msgs on q */
ushort  msg_qbytes;     /* max number of bytes on q */
ushort  msg_lspid;      /* pid of last msgsnd operation */
ushort  msg_lrpid;      /* pid of last msgrcv operation */
time_t  msg_stime;      /* last msgsnd time */
time_t  msg_rtime;      /* last msgrcv time */
time_t  msg_ctime;      /* last change time */
                        /* Times measured in secs since */
                        /* 00:00:00 GMT, Jan. 1, 1970 */
```

Message queue identifiers can be created using *stdipc*(3C).
msg_perm is a ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort cuid;        /* creator user id */
ushort cgid;        /* creator group id */
ushort uid;         /* user id */
ushort gid;         /* group id */
ushort mode;        /* r/w permission */
```

msg_qnum is the number of messages currently on the queue. **msg_qbytes** is the maximum number of bytes allowed on the queue. **msg_lspid** is the process id of the last process that performed a *msgsnd* operation. **msg_lrpid** is the process id of the last process that performed a *msgrcv* operation. **msg_stime** is the time of the last *msgsnd* operation, **msg_rtime** is the time of the last *msgrcv* operation, and **msg_ctime** is the time of the last *msgctl*(2) operation that changed a member of the above structure.

message operation permissions
In the *msgop*(2) and *msgctl*(2) system call descriptions, the permission required for an operation is indicated for each operation. Whether a particular process has these permissions for an object is determined by the object's permission mode bits as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a msqid are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches **msg_perm.[c]uid** in the data structure associated with *msqid* and the appropriate bit of the "user" portion (0600) of **msg_perm.mode** is set.

The process's effective user ID does not match **msg_perm.[c]uid** and either the process's effective group ID matches **msg_perm.[c]gid** or one of **msg_perm.[c]gid** is in the process's group access list and the appropriate bit of the "group" portion (060) of **msg_perm.mode** is set.

The process's effective user ID does not match **msg_perm.[c]uid** and the process's effective group ID does not match **msg_perm.[c]gid** and neither of **msg_perm.[c]gid** is in the process's group access list and the appropriate bit of the "other" portion (06) of **msg_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

metacharacter A character that has special meaning to the HP-UX shell, as well as to commands such as *ed*(1), *find*(1), and *egrep* (see *grep*(1)). The set of metacharacters includes: *, ?, !, [,], <, >, ;, |, ´, `, ", and &. Refer to *sh*(1) for the meaning associated with each. See also *regular expression.*

minor number A number that is an attribute of special files, specified during their creation and used whenever they are accessed, to enable I/O to or from specific devices. This number is passed to the device driver and is used to select which device in a family of devices is to be used, and possibly some operational modes. The exact format and meaning of the *minor* *number* is both system and driver dependent. Refer to the System Administrator manuals supplied with your system for details.

On Series 300/400 HP-IB devices, this number indicates the HP-IB *address*, *select code*, and the unit and/or volume numbers (see *address*). On Series 700 systems, a *minor number* indicates the device address, function number, and driver-dependent bits. On Series 800 systems, a *minor number* is an index into a table in the *kernel*.

mode A 16-bit word associated with every file in the file system, stored in the *inode*. The least-significant 12 bits of the *mode* determine the read, write, and execute permissions for the file owner, file group, and all others, and contain the set-user-ID, set-group-ID, and "sticky" (save text image after execution) bits. The least-significant 12 bits can be set by the *chmod*(1) command if you are the file's owner or the super-user. The sticky bit on a regular file can only be set by the super-user. These 12 bits are sometimes referred to as *permission bits*. The most-significant 4 bits specify the file type for the associated file and are set as the result of *open*(2) or *mknod*(2) system calls.

mountable file system
A removable blocked file system contained on some mass storage medium with its own root directory and an independent hierarchy of directories and files. See **block special file** and *mount*(1M).

multi-user state
The condition of the HP-UX operating system in which terminals (in addition to the system console) allow communication between the system and its users. By convention, multi-user run level is set at state 2, which is usually defined to contain all the terminal processes and *daemons* needed in a multi-user environment. Run levels are table driven, and are specified by *init*(1M), which sets the run level by looking at the file **/etc/inittab**. Do not confuse the multi-user system with the multi-user state. A multi-user system is a system which can have more than one user actively communicating with the system when it is in the multi-user state. The multi-user state removes the single-user restriction imposed by the single-user state (see *single-user state*, *inittab*(4)).

native language
A computer user's spoken or written language, such as Dutch, English, French, German, Greek, Italian, Spanish, Swedish, Katakana, Turkish, Korean, Chinese, etc.

Native Language Support (NLS)
A feature of HP-UX that provides the user with internationalized software and the application programmer with tools to develop this software.

new-line character
The character with an ASCII value of 10 (line-feed) used to separate lines of characters. It is represented by \n in the C language and in various utilities. The terminal driver (see *tty*(7)) normally interprets the carriage-return/line-feed sequence sent by a terminal as a single new-line character.

NLSPATH
An environment variable used to indicate the search path for message catalogs (see *message catalog*).

node name A string of up to 31 characters, not including control characters or spaces, that uniquely identifies a node on a Local Area Network (LAN). The *node name* for each system is set by the *npowerup* command, which is one of the commands supplied with optional LAN/9000 products. Do not confuse a node name with a *host name*, which is a string that uniquely identifies an HP-UX system on a *uucp* network. Your node and host names can be identical, but they are used and set by totally different software. See *host name*, *LAN/9000 User's Guide*, and *LAN/9000 Node Manager's Guide*.

non-spacing characters
Characters, such as a diacritical mark or accents, that are used in combination with other characters to form composite graphic symbols commonly found in non-English languages.

open file A file that is currently associated with a file descriptor.

open file description
: An *open file description* records how a process or group of processes are accessing a file. Each *file descriptor* refers to exactly one *open file description*, but an *open file description* can be referred to by more than one file descriptor. The *file offset*, *file status flags*, and *file access modes* are attributes of an *open file description*.

ordinary file
: A type of HP-UX file containing ASCII text (e.g. program source), binary data (e.g. executable code), etc. Ordinary files can be created by the user through I/O redirection, editors, or HP-UX commands.

orphan process
: Whenever a *parent process* terminates for any reason and leaves behind one or more active *child*process, the child processes are called *orphan processes* . *init*(1M) inherits (that is, becomes the effective parent of) all orphan processes.

orphaned process group
: A process group in which the parent of every member is either itself a member of the group or is not a member of the group's session.

owner
: The owner of a file is usually the creator of that file. However, the ownership of a file can be changed by the super-user or the current owner with the *chown*(1) command or the *chown*(2) system call. The file owner is able to do whatever he wants with his files, including remove them, copy them, move them, change their contents, etc. The owner can also change the files' modes.

parent directory
: A directory's *parent directory* is the directory one level above it in the *file hierarchy*, with the exception of the directory entries for *dot* and *dot-dot*. All directories except the *root directory* (/) have one (and only one) parent directory.

parent process
: Whenever a new process is created by a currently-existing process (via *fork*(2)), the currently existing process is said to be the parent process of the newly created process. Every process has exactly one parent process (except the init process, see *init*), but each process can create several new processes with the *fork*(2) system call. The parent process ID of any process is the *process ID* of its creator.

parent process ID
: A new process is created by a currently active process. The *parent process ID* of a process is the process ID of its creator for the lifetime of the creator. After the creator's lifetime has ended, the *parent process ID* is the process ID of *init*.

password
: A string of ASCII characters used to verify the identity of a user. Passwords can be associated with users and groups. If a user has a password, it is automatically encrypted and entered in the second field of that user's line in the **/etc/passwd** file. A user can create or change his or her own password by using the *passwd*(1) command.

path name
: (Sometimes written as one word, *pathname*). A sequence of directory names separated by slashes, and ending with any file name. All file names except the last in the sequence *must* be directories. If a path name begins with a *slash* (/), it is an absolute path name (see *absolute path name*); otherwise it is a relative path name (see *relative path name*). A path name defines the path to be followed through the hierarchical file system in order to find a particular file.

More precisely, a path name is a null-terminated character string constructed as follows:

```
<path-name>::=<file-name> | <path-prefix><file-name> | /
<path-prefix>::=<rtprefix> | /<rtprefix>
<rtprefix>::=<dirname>/ | <rtprefix><dirname>/
```

where `<file-name>` is a string of one or more characters other than the ASCII slash and null, and `<dirname>` is a string of one or more characters (other than the ASCII slash and null) that names a directory. (File and directory names can consist of up to 14 characters on systems supporting short file names and up to 255 characters on systems supporting long file names.)

A *slash* (/) by itself names the *root directory*. Two or more slashes in succession (*////...*) are treated as a single slash.

Unless specifically stated otherwise, the null or zero-length path name is treated as though it named a nonexistent file.

path name resolution
Path name resolution is performed for a process to resolve a path name to a particular file in a *file hierarchy*. Multiple path names can resolve to the same file, depending on whether resolution is sought in absolute or relative terms (see below). Each file name in the path name is located in the directory specified by its predecessor (for example, in the path name fragment **a/b**, file **b** is located in directory **a**). *Path name resolution* fails if this cannot be accomplished.

If the path name begins with a slash, the predecessor of the first file name in the path name is understood to be the *root directory* of the process, and the path name is referred to as an *absolute path name* . If the path name does not begin with a slash, the predecessor of the first file name of the path name is understood to be the current working directory of the process, and the path name is referred to as a *relative path name* . A path name consisting of a single slash resolves to the root directory of the process.

path prefix A *path prefix* is a *path name* with an optional ending *slash* that refers to a *directory*.

permission bits The nine least-significant bits of a file's *mode* are referred to as file *permission bits*. These bits determine read, write, and execute permissions for the file's *owner*, the file's *group*, and all others. The bits are divided into three parts: owner, group and other. Each part is used with the corresponding file class of processes. The bits are contained in the file mode, as described in *stat*(5). The detailed usage of the file permission bits in access decisions is described in *file access permissions.*

pipe An interprocess I/O channel used to pass data between two processes. It is commonly used by the *shell* to transfer data from the standard output of one process to the standard input of another. On a command line, a pipe is signaled by a vertical bar (|). Output from the command(s) to the left of the vertical bar is channeled directly into the standard input of the command(s) on the right.

portable file name character set
The following set of graphical characters are portable across conforming implementations of IEEE Standard P1003.1:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
01234567890._-

The last three characters are the dot, underscore and hyphen characters, respectively. The hyphen should not be used as the first character of a portable file name.

position-independent code
Object code that can run unmodified at any virtual address. Position-independent code can use PC-relative addressing modes and/or linkage tables. It is most often used in shared libraries, in which case the linkage tables are initialized by the dynamic loader. Position-independent code is generated when the **+z** or **+Z** compiler option is specified.

privileged groups
A *privileged group* is a group that has had a *setprivgrp* (see *getprivgrp*(2)) operation performed on it, giving it access to some system calls otherwise reserved for the super-user. See *appropriate privileges*.

proc1 See *init*.

position-independent code (PIC)
Code which can run unmodified at any virtual address. Position-independent code may use PC-relative addressing modes and/or linkage tables. It is most often used in shared libraries, in which case the linkage tables are initialized by the dynamic loader. Position independent code is generated when the **+z** or **+Z** option to the compilers is specified.

process	An invocation of a program, or the execution of an image (see *image*). Although all commands and utilities are executed within processes, not all commands or utilities have a one-to-one correspondence with processes. Some commands (such as *cd*) execute within a process, but do not create any new processes. Others (such as in the case of **ls | wc -l**) create multiple processes. Several processes can be running in the same program, but each might have different data and be in different stages of execution. A process can also be thought of as an *address space* and single thread of control that executes within that address space and its required system resources. A *process* is created by another process issuing the *fork*(2) function. The process that issues *fork*(2) is known as the *parent process* and the new process created by the *fork*(2) as the *child process*.
process group	Each process in the system is a member of a *process group*. This grouping permits the signaling of related processes. A newly created process joins the process group of its creator.
process group ID	Each process group in the system is uniquely identified during its lifetime by a *process group ID*, a positive integer less than or equal to {PIC_MAX}. A *process group ID* cannot be reused by the system until the process group lifetime ends.
process group leader	A *process group leader* is a process whose process ID is the same as its process group ID.
process group lifetime	A period of time that begins when a *process group* is created and ends when the last remaining process in the group leaves the group, either due to process termination or by calling the *setsid*(2) or *setpgid*(2) functions.
process ID	Each active process in the system is uniquely identified during its lifetime by a positive integer less than or equal to {PID_MAX} called a *process ID*. A process ID cannot be reused by the system until after the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID cannot be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1.
process lifetime	After a process is created with a *fork*(2) function, it is considered active. Its thread of control and *address space* exist until it terminates. It then enters an inactive state where certain resources may be returned to the system, although some resources, such as the *process ID* are still in use. When another process executes a *wait*(2), *wait3*(2), or *waitpid*(2) function for an inactive process, the remaining resources are returned to the system. The last resource to be returned to the system is the process ID. At this time, the lifetime of the process ends.
program	A sequence of instructions to the computer in the form of binary code (resulting from the compilation and assembly of program source).
prompt	The character(s) displayed by the *shell* on the display indicating that the system is ready for a command. The prompt is usually a dollar sign (**$**) for ordinary users and a pound sign (**#**) for the super-user, but the user can redefine it to be any string by setting the shell variable **PS1** in his or her **.profile** file. See also *secondary prompt*.
quit signal	The signal sent by SIGQUIT. See *signal*(2). The quit signal is generated by typing the character defined by the teletype handler as your quit signal. (See *stty*(1), *ioctl*(2), and *termio*(7).) The default is the ASCII FS character (ASCII value 28, generated by typing [Ctrl]-[\\]). This signal usually causes a running program to terminate and generates a file containing the "core image" of the terminated process. The core image is useful for debugging purposes. (Some systems do not support core images, and on those systems no such file is generated.)
radix character	The character that separates the integer part of a number from the fractional part. For example, in American usage, the *radix character* is a decimal point, while in Europe, a comma is used.
raw disk	The name given to a disk for which there exists a *character special file* that allows direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O call.

read-only file system
A characteristic of a *file system* that prevents file system modifications.

real group ID A positive integer which is assigned to every user on the system. The association of a user and his or her *real group ID* is done in the file **/etc/passwd** . The modifier "real" is used because a user can also have an *effective group ID*. The real group ID can then be mapped to a group name in the file **/etc/group**, although it need not be. Thus, every user is a member of some group (which can be nameless), even if that group has only one member.

Every time a process creates a child process (via *fork*(2)), that process has a real group ID equal to the parent process's real group ID. This is useful for determining file access privileges within the process.

real user ID A positive integer which is assigned to every user on the system. A real user ID is assigned to every valid *login* name in the file **/etc/passwd**. The modifier "real" is used because a user can also have an *effective user ID* (see *effective user ID*).

Every time a process creates a child process (via *fork*(2)), that process has a real user ID equal to the parent process's real user ID. This is useful for determining file access privileges within the process.

regular expression
A string of zero or more characters that selects text. All the characters contained in the string might be literal, meaning that the regular expression matches itself only; or one or more of the characters might be a *metacharacter*, meaning that a single regular expression could match several literal strings. Regular expressions are most often encountered in text editors (such as *ed*(1), *ex*(1), or *vi*(1)), where searches are performed for a specific piece of text, or in commands that were created to search for a particular string in a file (most notably *grep*(1)). Regular expressions are also encountered in the shell, *sh*(1), especially when referring to file names on command lines. See *ed*(1).

regular file A type of *file* that is a randomly accessible sequence of bytes, with no further structure imposed by the system. Its size can be extended. A regular file is also called an ordinary file.

relative path name
A *path name* that does not begin with a *slash* (/). It indicates that a file's location is given relative to your current *working directory*, and that the search begins there (instead of at the *root directory*). For example, **dirl/file2** searches for the directory **dirl** in your current working directory; then **dirl** is searched for the file **file2**.

root directory (1) The highest level directory of the hierarchical file system, from which all other files branch. In HP-UX, the *slash* (/) character refers to the *root directory*. The root directory is the only directory in the file system that is its own *parent directory*.

(2) Each process has associated with it a concept of a root directory for the purpose of resolving path name searches for those paths beginning with *slash* (/). A process's root directory need not be the root directory of the root file system, and can be changed by the *chroot*(1) command or *chroot*(2) system call. Such a directory appears to the process involved to have .. pointing to itself.

root server The node in a *cluster* to which the storage device containing the root file system of the *cluster* is physically attached. More properly referred to as the *cluster server*.

This term applies to implementations that include HP Clustered Environment networks.

root volume The mass storage volume which contains the boot area (which contains the HP-UX kernel) and the root directory of the HP-UX file system.

saved group ID Every process has a saved group ID that retains the process's *effective group ID* from the last successful *exec*(2)*or* *setresgid*(2), or from the last super-user call to *setgid*(2) or *setresuid*(2). *setgid* permits a process to set its effective group ID to this remembered value. Consequently, a process that executes a program with the set-group-ID bit set and with a group ID of 5 (for example) can set its effective group ID to 5 at any time until the program terminates. See *exec*(2), *setuid*(2), *saved user ID*, *effective group ID*, and *set-group-ID bit*. The saved group ID is also known as the saved set-group-ID.

saved process group ID
Every process has a saved process group ID that retains the process's group ID from the last successful *exec*(2). See *setpgrp*(2), *termio*(7), and process group ID.

saved user ID Every process has a *saved user ID* that retains the process's *effective user ID* from the last successful *exec*(2)*or setresuid*(2), or from the last super-user call to *setuid*(2). *setuid*(2) permits a process to set its effective user ID to this remembered value. Consequently, a process which executes a program with the set-user-ID bit set and with an owner ID of 5 (for example) can set its effective user ID to 5 at any time until the program terminates. See *exec*(2), *setuid*(2), *saved group ID:*, *effective user ID*, and *set-user-ID bit*. The saved user ID is also known as the saved set-user-ID.

SCCS An acronym for Source Code Control System. The Source Code Control System is a set of HP-UX commands which enable you to store changes to an SCCS file as separate "units" (called *deltas*). These units, each of which contains one or more textual changes to the file, can then be applied to or excluded from the SCCS file to obtain different versions of the file. The commands that make up SCCS are *admin*(1), *cdc*(1), *delta*(1), *get*(1), *prs*(1), *rmdel*(1), *sact*(1), *sccsdiff*(1), *unget*(1), *val*(1), and *what*(1). See *delta*, *SCCS file*.

SCCS file An ordinary text file which has been modified so that the Source Code Control System (SCCS) can be used with it. This modification is done automatically by the *admin*(1) command. See *SCCS*, *delta*.

SDF An acronym for Structured Directory Format. SDF was implemented on the obsolete Series 500 computers only, and provided tree-structured access to files through the root directory of the volume.

secondary prompt
One or more characters that the shell prints on the display, indicating that more input is needed. This prompt is not encountered nearly as frequently as the shell's primary prompt (see *prompt*). When it occurs, it is usually caused by an omitted right quote on a string (which confuses the shell), or when you enter a shell programming language control-flow construct (such as a **for** construct) from the command line. By default, the shell's secondary prompt is the greater-than sign (**>**), but you can re-define it by setting the shell variable **PS2** appropriately in your **.profile** file.

select code On Series 300/400 and 700 systems, part of an *address* used for devices. On Series 300/400 systems, a number determined by a setting on the interface card to which a peripheral device is connected, or by the particular I/O slot in which the I/O card resides. Multiple peripherals connected to the same interface card share the same select code. On Series 700 systems, *select code* consists of the bus and slot numbers for a device, both of which are determined by the particular I/O slot in which the I/O card resides. All functions on a multi-function card share the same select code.

semaphore identifier (semid)
A *semaphore identifier* is a unique positive integer created by a *semget*(2) system call. Each *semid* has a set of semaphores and a data structure associated with it. The data structure is referred to as *semid_ds* and contains the following members:

```
struct  ipc_perm sem_perm;/* operation permission */
ushort  sem_nsems;   /* number of sems in set */
time_t  sem_otime;   /* last operation time */
time_t  sem_ctime;   /* last change time */
                     /* Times measured in secs since */
                     /* 00:00:00 GMT, Jan. 1, 1970 */
```

Semaphore identifiers can be created using *stdipc*(3C).

sem_perm is a ipc_perm structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```
ushort  cuid;        /* creator user id */
ushort  cgid;        /* creator group id */
ushort uid;          /* user id */
ushort gid;          /* group id */
ushort mode;         /* r/a permission */
```

The value of **sem_nsems** is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a *sem_num*. *sem_num* values run sequentially from 0 to the value of sem_nsems minus 1. **sem_otime** is the time of the last *semop*(2) operation, and **sem_ctime** is the time of the last *semctl*(2) operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort semval;       /* semaphore value */
short  sempid;       /* pid of last operation  */
ushort semncnt;      /* # awaiting semval > cval */
ushort semzcnt;      /* # awaiting semval = 0 */
```

semval is a non-negative integer. **sempid** is equal to the process ID of the last process that performed a semaphore operation on this semaphore. **semncnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become greater than its current value. **semzcnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become zero.

semaphore operation permissions
In the *semop*(2) and *semctl*(2) system call descriptions, the permission required for an operation is indicated for each operation. Whether a particular process has these permissions for an object is determined by the object's permission mode bits as follows:

00400	Read by user
00200	Alter by user
00060	Read, Alter by group
00006	Read, Alter by others

Read and Alter permissions on a semid are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches **sem_perm.**[c]**uid** in the data structure associated with *semid* and the appropriate bit of the "user" portion (0600) of **sem_perm.mode** is set.

The process's effective user ID does not match **sem_perm.**[c]**uid** and the appropriate bit of the "group" portion (060) of **sem_perm.mode** is set.

The process's effective user ID does not match **sem_perm.**[c]**uid** and the process's effective group ID does not match **sem_perm.**[c]**gid** and neither of **sem_perm.**[c]**gid** is in the process's group access list and the appropriate bit of the "other" portion (06) of **sem_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

session Each process group is a member of a session. A process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership (see *setsid*(2)). A session can have multiple process groups (see *setpgid*(2)).

session leader A process that has created a session (see *setsid*(2)).

session lifetime The period between when a session is created and the end of the lifetime of all process groups that remain as members of the session.

set-group-ID bit
A single bit in the mode of every file in the file system. If a file is executed whose set-group-ID bit is set, the effective group ID of the process which executed the file is set equal to the real group ID of the owner of the file. See *effective group ID*, *group*, and *real group ID*.

set-user-ID bit A single bit in the mode of every file in the file system. If a file is executed whose set-user-ID bit is set, the effective user ID of the process which executed the file is set equal to the real user ID of the owner of the file. See *effective*user*ID* and *real user ID*.

shared library An executable file that can be shared between several different programs. Code from a shared library is not linked into the program by *ld*(1), but is instead mapped into the process' address space at run time by the dynamic loader. Shared libraries must contain position-independent code, and are created by *ld*(1). They typically have the filename suffix **.sl**.

shared memory identifier
A shared memory identifier (shmid) is a unique positive integer created by a *shmget*(2) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as *shmid_ds* and contains the following members:

```
struct  ipc_perm shm_perm;/* operation permission struct */
int     shm_segsz;         /* size of segment */
ushort  shm_cpid;          /* creator pid */
ushort  shm_lpid;          /* pid of last operation */
short   shm_nattch;        /* number of current attaches */
time_t  shm_atime;         /* last attach time */
time_t  shm_dtime;         /* last detach time */
time_t  shm_ctime;         /* last change time */
                           /* Times measured in secs since */
                           /* 00:00:00 GMT, Jan. 1, 1970 */
```

Shared memory identifiers can be created using *stdipc*(3C).

shm_perm is a ipc_perm structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```
ushort  cuid;       /* creator user id */
ushort  cgid;       /* creator group id */
ushort  uid;        /* user id */
ushort  gid;        /* group id */
ushort  mode;       /* r/w permission */
```

shm_segsz specifies the size of the shared memory segment. **shm_cpid** is the process id of the process that created the shared memory identifier. **shm_lpid** is the process id of the last process that performed a *shmop*(2) operation. **shm_nattch** is the number of processes that currently have this segment attached. **shm_atime** is the time of the last *shmat* operation, **shm_dtime** is the time of the last *shmdt* operation, and **shm_ctime** is the time of the last *shmctl*(2) operation that changed one of the members of the above structure.

shared memory operation permissions
In the *shmop*(2) and *shmctl*(2) system call descriptions, the permission required for an operation is indicated for each operation. Whether a particular process has these permissions for an object is determined by the object's permission mode bits as follows:

00400	Read by user
00200	Write by user
00060	Read, Write by group
00006	Read, Write by others

Read and Write permissions on a shmid are granted to a process if one or more of the following are true:

The process's effective user ID is super-user.

The process's effective user ID matches **shm_perm.**[c]**uid** in the data structure associated with *shmid* and the appropriate bit of the "user" portion (0600) of **shm_perm.mode** is set.

The process's effective user ID does not match **shm_perm.**[c]**uid** and either the process's effective group ID matches **shm_perm.**[c]**gid** or one of **shm_perm.**[c]**gid** is in the process's group access list and the appropriate bit of the "group" portion (060) of **shm_perm.mode** is set.

The process's effective user ID does not match **shm_perm.**[c]**uid** and the process's effective group ID does not match **shm_perm.**[c]**gid** and neither of **shm_perm.**[c]**gid** is in the process's group access list and the appropriate bit of the "other" portion (06) of **shm_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

shell The shell functions as both a command interpreter and an interpretive programming language. The shell is automatically invoked for every user who logs in, in order to provide a user-interface to the HP-UX operating system. See *sh*(1), *csh*(1), *ksh*(1), *pam*(1), and the tutorials supplied with your system for details.

shell program See *shell script*.

shell script A sequence of shell commands and shell programming language constructs stored in a file and invoked as a user command (program). No compilation is needed prior to execution because the shell recognizes the commands and constructs that make up the shell programming language. A shell script is often called a *shell program* or a *command file*. See the *Shells* User Guide.

signal Signals are software interrupts sent to processes, informing them of special situations or events. The term signal is also used to refer to the event itself. See *signal*(2).

single-user state
A condition of the HP-UX operating system in which the system console provides the only communication mechanism between the system and its user. By convention, single-user state is usually specified by *init*(1M) as run-level **S** or **s**. Do not confuse the single-user state, in which the software is limiting a multi-user system to a single-user communication, with a single-user system, which can never communicate with more than one fixed terminal (see *multi-user*state).

slash The term *slash* is used to represent the literal character "/". This character is also known as "solidus". A *path name* consisting of a single slash resolves to the *root directory* of the process. (Also see *path name resolution*.)

source code The fundamental high-level information (program) written in the syntax of a specified computer language. Object (machine-language) code is derived from source code. When dealing with HP-UX shell command language, *source code* is input to the command language interpreter. The term *shell script* is synonymous with this meaning. When dealing with the C Language, *source code* is input to the *cc*(1) command. *Source code* can also refer to a collection of sources meeting any of the above conditions.

special file Often called a *device file*, this is a file associated with an I/O device. Special files are read and written the same as ordinary files, but requests to read or write result in activation of the associated device. Due to convention and consistency, these files should always reside in the **/dev** directory.

special processes
Processes with certain (small) process IDs are special. On a typical system, the IDs of 0, 1, and 2 are assigned as follows: Process 0 is the scheduler. Process 1 is the initialization process *init*, and is the ancestor of every other process in the system. It is used to control the process structure. On paging systems with virtual memory process 2 is the paging daemon.

standard error The destination of error and special messages from a program, intended to be used for diagnostic messages. The standard error output is often called *stderr*, and is automatically opened for writing on file descriptor 2 for every command invoked. By default, the user's terminal is the destination of all data written to *stderr*, but it can be redirected elsewhere. Unlike standard input and standard output, which are never used for data transfer in the "wrong" direction, standard error is occasionally read. This is not recommended practice, since I/O redirection is likely to break a program doing this.

standard input The source of input data for a program. The standard input file is often called *stdin*, and is automatically opened for reading on file descriptor 0 for every command invoked. By default, the user's terminal is the source of all data read from *stdin*, but it can be redirected from another source.

standard output
The destination of output data from a program. The standard output file is often called *stdout*, and is automatically opened for writing on file descriptor 1 for every command invoked. By default, the user's terminal is the destination of all data written to *stdout*, but it can be redirected elsewhere.

stream A term most often used in conjunction with the standard I/O library routines documented in section (3) of this manual. A stream is simply a file pointer (declared as **FILE *stream**) returned by the *fopen*(3S) library routines. It may or may not have buffering associated with it (by default, buffering is assigned, but this can be modified with *setbuf*(3S)).

sticky bit A single bit in the mode of every file in the file system. If set on a regular file, the contents of the file stay permanently in memory instead of being swapped back out to disk when the file has finished executing. Only the super-user can set the sticky bit on a regular file. The sticky bit is read each time the file is executed (via *exec*(2)).

If set on a directory, the files in that directory can be removed or renamed only by the owner of the file, owner of the directory containing the file, or the super-user. See the *chmod*(2), *rename*(2), *rmdir*(2), and *unlink*(2) manual entries for more information.

sub-directory A directory that is one (or perhaps more) levels lower in the file system hierarchy than a given directory. Sometimes called a *subordinate directory*.

subordinate directory
See *sub-directory*.

super block A block on each file system's mass storage medium which describes the file system. The contents of the super-block vary between implementations. Refer to the *System Administrator* manuals supplied with your system, and the appropriate *fs*(4) entry for details.

super-user The HP-UX system administrator. This user has access to all files, and can perform privileged operations. He has a real and effective user ID of 0, and, by convention, the user name of *root*.

superior directory
See *parent directory*.

supplementary group ID
A process has up to {NGROUPS_MAX} supplementary group IDs used in determining file access permissions, in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created.

swap server In the HP Clustered environment, the cluster node that provides swap space for a given cluster node. A client cnode can have the root (cluster) server as its *swap server*, use a local disk for swap space, or have an *auxiliary swap server* (a non-cluster-server cnode with local swap space). The cluster server always uses its own local swap space. The **/etc/clusterconf** file (see *clusterconf*(4)) contains a field that specifies the swap location for each cnode.

symbolic link A type of file that indirectly refers to a path name. See *symlink*(4).

system The term *system* is used to refer to the HP-UX operating system.

system asynchronous I/O
A method of performing I/O whereby a process informs a driver or subsystem that it wants to know when data has arrived or when it is possible to perform a write request. The driver or subsystem maintains a set of buffers through which the process performs I/O. See the *ioctl*(2), *select*(2), *read*(2), and *write*(2) manual entries for more information.

system call An HP-UX operating system kernel function available to the user through a high-level language (such as FORTRAN, Pascal, or C). Also called an "intrinsic" or a "system intrinsic."

The available system calls are documented in section (2) of the *HP-UX Reference*.

system console A keyboard and display (or terminal) given a unique status by HP-UX and associated with the special file **/dev/console**. All boot ROM error messages, HP-UX system error messages, and certain system status messages are sent to the system console. Under certain conditions (such as the single-user state), the system console provides the only mechanism for communicating with HP-UX. See the System Administrator manuals and user guides provided with your system for details on configuration and use of the system console.

system process A *system process* is a process that runs on behalf of the system. It may have special implementation-defined characteristics.

terminal (or terminal device)
A *character special file* that obeys the specifications of *termio*(7).

terminal affiliation
The process by which a process group leader establishes an association between itself and a particular terminal. A terminal becomes affiliated with a process group leader (and subsequently all processes created by the process group leader, see *terminal group*) whenever the process group leader executes (either directly or indirectly) an *open*(2) or *creat*(2) system call to open a terminal. Then, *if* the process which is executing *open*(2) or *creat*(2) is a process group leader, and *if* that process group leader is not yet affiliated with a terminal, and *if* the terminal being opened is not yet affiliated with a process group, the affiliation is established (however, see *open*(2) description of **O_NOCTTY**).

An affiliated terminal keeps track of its process group affiliation by storing the process group's process group ID in an internal structure.

Two benefits are realized by terminal affiliation. First, all signals sent from the terminal are sent to all processes in the terminal group. Second, all processes in the terminal group can perform I/O to/from the generic terminal driver **/dev/tty**, which automatically selects the affiliated terminal.

Terminal affiliation is broken with a terminal group when the process group leader terminates, after which the hangup signal is sent to all processes remaining in the process group. Also, if a process (which is not a process group leader) in the terminal group becomes a process group leader via the *setpgrp*(2) system call, its terminal affiliation is broken.

See *process group*, *process group leader*, *terminal group*, and *setpgrp*(2).

text file A file that contains characters organized into one or more lines. The lines shall not contain NUL characters, and none shall exceed {LINE_MAX} bytes in length including the terminating new-line character. Although neither the kernel nor the C language implementation distinguishes between text files and binary files (see ANSI C Standard X3-159-19xx), many utilities behave predictably only when operating on text files.

tty Originally an abbreviation for teletypewriter, *tty* now general refers to a *terminal*.

upshifting The conversion of a lowercase character to its uppercase representation.

user ID Each system user is identified by an integer known as a *user*ID, which is in the range of zero to {UID_MAX}, inclusive. Depending on how the user is identified with a process, a *user ID* value is referred to as a *real user ID*, an *effective user ID*, or a *saved set user ID*.

utility An executable file, which might contain executable object code (that is, a *program*), or a list of *commands* to execute in a given order (that is, a *shell script*). You can write your own utilities, either as executable programs or shell scripts (which are written in the shell programming language).

volume number Part of an address used for devices. A number whose meaning is software- and device-dependent, but which is often used to specify a particular volume on a multi-volume disk drive. See the *System Administrator* manuals supplied with your system for details.

white space One or more characters which, when displayed, cause a movement of the cursor or print head, but do not result in the display of any visible graphic. The white-space characters in the ASCII code set are space, tab, new-line, form-feed, carriage return, and vertical tab. A particular command or routine might interpret some, but not necessarily all, white-space characters as delimiting fields, words, or command options.

working directory
Each process has associated with it the concept of a current working directory. For a shell, this appears as the directory in which you currently "reside". This is the directory in which relative path name (i.e., a path name that does not begin with "/") searches begin. It is sometimes referred to as the *current directory*, or the *current working directory*.

zombie process The name given to a process which terminates for any reason, but whose parent process has not yet waited for it to terminate (via *wait*(2)). The process which terminated continues to occupy a slot in the process table until its parent process waits for it. Because it has terminated, however, there is no other space allocated to it either in user or kernel space. It is therefore a relatively harmless occurrence which will rectify itself the next time its parent process waits. The *ps*(1) command lists zombie processes as "defunct."

Index to Volume 1

Index
Volume 1

Description	Entry Name(Section)
3179G/3192G, 3270, 3777 terminal emulator, IBM	**sna(1)**
3270, 3777 terminal emulator, IBM 3179G/3192G,	**sna(1)**
3777 terminal emulator, IBM 3179G/3192G, 3270,	**sna(1)**
absolute debugger	**adb(1)**
access and manage the pathalias database	**uupath(1)**
access control lists (ACLs) of files, list	**lsacl(1)**
access control lists; add, modify, delete, copy, or summarize	**chacl(1)**
access environment, database	**isql(1)**
access, modification, and/or change times of a file, update	**touch(1)**
access permissions mode mask for file-creation, set	**umask(1)**
access permissions on a file (mode), change	**chmod(1)**
access privileges for group, list	**getprivgrp(1)**
access rights to file(s), list	**getaccess(1)**
access the terminfo database	**tput(1)**
access **uucp** and **uux** transactions summary log	**uucp(1)**
activity, print current SCCS file editing	**sact(1)**
activity reporter, system	**sar(1)**
adb – absolute debugger	**adb(1)**
add line number in front of each line in a file	**nl(1)**
add, modify, delete, copy, or summarize file access control lists (ACLs)	**chacl(1)**
address router, electronic	**pathalias(1)**
adjust – simple text formatter	**adjust(1)**
admin – create and administer SCCS files	**admin(1)**
administer and create SCCS files	**admin(1)**
affirmative responses, repetitively	**yes(1)**
agent, check connectivity with the SNMP	**chksnmpd**
aliases and paths, locate a program file including	**which(1)**
aliases, **elm** user and system, verify and create	**elmalias(1)**
aliases, print system-wide sendmail	**praliases(1)**
alias – substitute command and/or filename	**csh(1)**
alias – substitute command and/or filename	**ksh(1)**
alias – substitute command and/or filename	**sh-posix(1)**
ALLBASE/SQL database, generate command files to unload, reload	**sqlgen(1)**
ALLBASE/SQL DBEnvironment, configure and maintain	**sqlutil(1)**
ALLBASE/SQL DBEnvironment, migrate ALLBASE/SQL	**sqlmig(1)**
ALLBASE/SQL interactive SQL interface	**isql(1)**
ALLBASE/SQL Pascal, C, FORTRAN, or COBOL source programs, preprocess	**psqlc(1)**
allocate reserved space for a disk storage file	**prealloc(1)**
allocation space of object files, print section sizes and	**size(1)**
alloc – show dynamic memory usage	**csh(1)**
alter contents of and copy a (tape) file	**dd(1)**
alter priority of running processes	**renice(1)**
alter selected characters	**tr(1)**
analysis of text, generate programs for lexical	**lex(1)**
another system over LAN, log in on	**vt(1)**
ANSI C function prototype generator	**protogen(1)**
append characters to obtain text line of specified length	**newform(1)**
ar – archive and library maintainer for portable archives	**ar(1)**
arbitrary-precision arithmetic language	**bc(1)**
archive and library maintainer for portable archives	**ar(1)**
archive exchange, portable	**pax(1)**
archiver for tape files	**tar(1)**
archiver, tape file	**tar(1)**
archives, copy file archives in and out	**cpio(1)**
argument list(s), large, construct and execute command	**xargs(1)**
arguments, echo (print)	**echo(1)**
arguments, evaluate as an expression	**expr(1)**
arguments, print formatted	**printf(1)**
arguments, too many (error); construct argument list(s) and execute command	**xargs(1)**

Index
Volume 1

Description	Entry Name(Section)
arithmetic desk calculator	**dc(1)**
arithmetic language, arbitrary-precision	**bc(1)**
ASA carriage control characters, interpret	**asa(1)**
asa – interpret ASA carriage control characters	**asa(1)**
as – assembler (architecture-dependent general entry)	**as(1)**
as – assembler (Series 300/400 implementation)	**as_300(1)**
as – assembler (Series 700/800 implementation)	**as_800(1)**
ASCII, convert binary file to, for transmission by mailer	**uuencode(1)**
assembler (architecture-dependent general entry)	**as(1)**
assembler debugger	**adb(1)**
assembler (Series 300/400 implementation)	**as_300(1)**
assembler (Series 700/800 implementation)	**as_800(1)**
assembly language instruction sequence, time an	**atime(1)**
assembly language, translate	**astrn(1)**
assembly language, translate	**atrans(1)**
assign network node name	**nodename(1)**
astrn – translate assembly language	**astrn(1)**
at, batch – execute commands at a later time	**at(1)**
atime – time an assembly language instruction sequence	**atime(1)**
atrans – translate assembly language	**atrans(1)**
attributes, change program's internal	**chatr(1)**
attributes, change RCS file	**rcs(1)**
attributes for group, get special	**getprivgrp(1)**
attributes of a DOS file, change	**doschmod(1)**
available DOS disk clusters, report number of	**dosdf(1)**
awk – text pattern scanning and processing language	**awk(1)**
background processes to complete, wait for	**wait(1)**
backspaces and reverse line-feeds, remove from text	**col(1)**
banner – make posters in large letters	**banner(1)**
basename, dirname – extract portions of path names	**basename(1)**
batch, at – execute commands at a later time	**at(1)**
bc – arbitrary-precision arithmetic language	**bc(1)**
bdiff – big diff	**bdiff(1)**
beautifier, formatter for C programs	**cb(1)**
beginning of file, list first few lines at	**head(1)**
bfs – big file scanner	**bfs(1)**
bifchgrp, bifchown – change BIF file owner or group	**bifchown(1)**
bifchmod – change mode of a BIF file	**bifchmod(1)**
bifchown, bifchgrp – change BIF file owner or group	**bifchown(1)**
bifcp – copy to or from BIF files	**bifcp(1)**
BIF directories, list contents of	**bifls(1)**
BIF directory, make a	**bifmkdir(1)**
BIF file, change mode of a	**bifchmod(1)**
BIF files, copy to or from	**bifcp(1)**
BIF files or directories, remove	**bifrm(1)**
BIF file system, find files in a	**biffind(1)**
biffind – find files in a BIF file system	**biffind(1)**
bifls – list contents of BIF directories	**bifls(1)**
bifmkdir – make a BIF directory	**bifmkdir(1)**
bifrm, bifrmdir – remove BIF files or directories	**bifrm(1)**
bifrmdir, bifrm – remove BIF files or directories	**bifrm(1)**
big diff	**bdiff(1)**
big file scanner	**bfs(1)**
binary file, convert to ASCII for transmission by mailer	**uuencode(1)**
binary or object file, find the printable strings in an	**strings(1)**
binary program files for given name, find location of	**whereis(1)**
blank lines, reduce multiple adjacent to single blank line	**ssp(1)**
blank lines, remove all from file	**rmnl(1)**

Description	**Entry Name(Section)**
block count and checksum of a file, print	**sum(1)**
Bourne shell	**sh(1)**
break a file into multiple *n*-line pieces	**split(1)**
break – exit from enclosing for/next loop	**csh(1)**
break – exit from enclosing for/next loop	**ksh(1)**
break – exit from enclosing for/next loop	**sh(1)**
break – exit from enclosing for/next loop	**sh-posix(1)**
breaksw – break from switch and resume after endsw	**csh(1)**
bs – a compiler/interpreter for modest-sized programs	**bs(1)**
bss (uninitialized data) allocation space of object files, print section sizes and	**size(1)**
buffer, preserve editor	**expreserve(1)**
build a makefile	**mkmf(1)**
calculator, desk	**dc(1)**
calendar, print	**cal(1)**
calendar – reminder service	**calendar(1)**
call another (UNIX) system; terminal emulator	**cu(1)**
call graph execution profile data, display	**gprof(1)**
call terminal– spawn getty to remote terminal	**ct(1)**
cal – print calendar	**cal(1)**
cancel – cancel requests to an LP line printer or plotter	**lp(1)**
cancel or send requests to an LP line printer or plotter	**lp(1)**
capabilities, terminal, get from terminfo database	**tput(1)**
carriage control characters, ASA, interpret	**asa(1)**
cartridge, partition DDS tape	**mediainit(1)**
cartridge tape, hard disk, or flexible disk media, initialize	**mediainit(1)**
Cartridge Tape I/O Utility, CS/80	**tcio(1)**
case – execute *list* associated with *pattern* that matches *word*	**ksh(1)**
case – execute *list* associated with *pattern* that matches *word*	**sh-posix(1)**
case – label in a switch statement	**csh(1)**
cat after uncompacting Huffman coded files (see **pack(1)**)	**compact(1)**
catalogs, message, find strings for inclusion in	**findstr(1)**
catalogue file, generate a formatted message	**gencat(1)**
catalogue file, message, create for modification	**findmsg(1)**
cat – concatenate, copy, and print files	**cat(1)**
catgets(3C), insert calls to based on **findstr**(1) output	**insertmsg(1)**
cb – C program beautifier, formatter	**cb(1)**
ccat – uncompact and cat files using Huffman code (see **pack**(1))	**compact(1)**
cc – C compiler	**cc(1)**
C, C++, FORTRAN, and Pascal symbolic debugger	**xdb(1)**
C, C++, FORTRAN, Pascal symbolic debugger	**cdb(1)**
C compiler	**cc(1)**
cdb, **fdb**, **pdb** – C, C++, FORTRAN, Pascal symbolic debugger	**cdb(1)**
cdc – change the delta commentary of an SCCS delta	**cdc(1)**
cd – change working directory	**cd(1)**
cd – change working directory	**csh(1)**
cd – change working directory	**ksh(1)**
cd – change working directory	**sh(1)**
cd – change working directory	**sh-posix(1)**
certify mass storage media	**mediainit(1)**
cflow – generate C flow graph	**cflow(1)**
C flow graph, generate	**cflow(1)**
C function prototype generator, ANSI	**protogen(1)**
chacl – add, modify, delete, copy, or summarize file access control lists (ACLs)	**chacl(1)**
change, access, and/or modification times of a file, update	**touch(1)**
change attributes of a DOS file	**doschmod(1)**
change BIF file owner or group	**bifchown(1)**
change current login to a new group	**newgrp(1)**
change data format of and copy a (tape) file	**dd(1)**

Index
Volume 1

Index
Volume 1

Description	**Entry Name(Section)**
C program cross-reference, generate	**cxref(1)**
C programs, extract strings from C programs to implement shared strings	**xstr(1)**
cps – report process status for entire cluster	**ps(1)**
create a BIF directory	**bifmkdir(1)**
create a directory	**mkdir(1)**
create a DOS directory	**dosmkdir(1)**
create a makefile	**mkmf(1)**
create and administer SCCS files	**admin(1)**
create and verify elm user and system aliases	**elmalias(1)**
create an SDF directory	**sdfmkdir(1)**
create a tags file	**ctags(1)**
create message catalogue file for modification	**findmsg(1)**
create network node name	**nodename(1)**
create zero-length file	**cat(1)**
create zero-length file	**cp(1)**
create zero-length file	**touch(1)**
crontab file operations, user	**crontab(1)**
crontab – user crontab file operations	**crontab(1)**
cross-reference, generate C program	**cxref(1)**
CRT or line-printer output, format text file for	**nroff(1)**
CRT terminal, record displayed output simultaneously in a file	**script(1)**
CRT terminals, peruse file on	**more(1)**
crypt – encode/decode files	**crypt(1)**
CS/80 Cartridge Tape I/O Utility	**tcio(1)**
csh – a shell (command interpreter) with C-like syntax	**csh(1)**
C source, extract error messages from into a file	**mkstr(1)**
csplit – context split	**csplit(1)**
ctags – create a tags file	**ctags(1)**
ct – spawn getty to remote terminal (call terminal)	**ct(1)**
cu – call another (UNIX) system; terminal emulator	**cu(1)**
cue – HP Character-Terminal User Environment (CUE)	**cue(1)**
(CUE), HP Character-Terminal User Environment	**cue(1)**
current context, display	**getcontext(1)**
current host system, set or print name of	**hostname(1)**
current HP-UX version, print name of	**uname(1)**
current processes, list	**ps(1)**
current SCCS file editing activity, print	**sact(1)**
current system users, list	**who(1)**
current user id, print or display	**whoami(1)**
cut – cut out selected fields of each line in a file	**cut(1)**
cxref – generate C program cross-reference	**cxref(1)**
Cyclical Redundancy Check on a file	**sum(1)**
data allocation space of object files, print section sizes and	**size(1)**
database access environment	**isql(1)**
database, pathalias, access and manage the	**uupath(1)**
database, relational, join two relations in	**join(1)**
data, expand or compress	**compress(1)**
data order for display/keyboard, convert file	**forder(1)**
date – print or set the system-clock date and time	**date(1)**
DBEnvironment, configure and maintain ALLBASE/SQL	**sqlutil(1)**
DBEnvironment, migrate ALLBASE/SQL	**sqlmig(1)**
dc – desk calculator	**dc(1)**
dd – convert, reblock, translate, and copy a (tape) file	**dd(1)**
DDS tape cartridge, partition	**mediainit(1)**
debugger:	
absolute debugger	**adb(1)**
assembler debugger	**adb(1)**
object code debugger	**adb(1)**

Index
Volume 1

Description	Entry Name(Section)
Cyclical Redundancy Check on a file	**sum(1)**
determine file type	**file(1)**
differential file comparator	**diff(1)**
display file on CRT terminals	**more(1)**
display file on soft-copy terminals	**pg(1)**
dump file in octal or hexadecimal format	**od(1)**
eliminate adjacent repeated lines in a file	**uniq(1)**
encrypt/decrypt files	**crypt(1)**
error message file, extract error messages from C source into	**mkstr(1)**
expand compressed file	**compress(1)**
expand compressed files in a directory	**compress(1)**
find differences among three files	**diff3(1)**
find differences between two files	**diff(1)**
find files in a BIF file system	**biffind(1)**
find files in an SDF file system	**sdffind(1)**
find (search for) files	**find(1)**
find the printable strings in an object or other binary file	**strings(1)**
format and print files	**pr(1)**
get a version of an SCCS file	**get(1)**
get first few lines in a file	**head(1)**
get lines from last part of a file	**tail(1)**
get SCCS identification information from files	**what(1)**
LIF files	see LIF files
link existing file to a new filename	**ln(1)**
list access control lists (ACLs) of files	**lsacl(1)**
list access rights to file(s)	**getaccess(1)**
make a delta (change) to an SCCS file	**delta(1)**
make unprintable characters in a file visible or invisible	**vis(1)**
merge corresponding lines of several files or subsequent lines of one file	**paste(1)**
move directory subtree and files to another directory	**mv(1)**
move file to new location	**mv(1)**
move multiple files to another directory	**mv(1)**
name for a temporary file, make a	**mktemp(1)**
overwrite file with an existing file	**cp(1)**
overwrite file with an existing file	**mv(1)**
print (and format) files	**pr(1)**
print and summarize an SCCS file	**prs(1)**
print checksum and block count of a file	**sum(1)**
print first few lines in a file	**head(1)**
print section sizes and allocation space of object files	**size(1)**
read portable archive	**pax(1)**
reduce multiple adjacent blank lines to single blank line	**ssp(1)**
remove a delta from an SCCS file	**rmdel(1)**
remove all blank lines from file	**rmnl(1)**
remove BIF files or directories	**bifrm(1)**
remove DOS files or directories	**dosrm(1)**
remove one or more files	**rm(1)**
remove SDF file or directory	**sdfrm(1)**
rename directory	**mv(1)**
rename file	**mv(1)**
report adjacent repeated lines in a file	**uniq(1)**
reverse the left-to-right text character sequence in each line of a file	**rev(1)**
search a file for a string or expression	**grep(1)**
select/reject lines common to two sorted files	**comm(1)**
send copy of standard output to specified file	**tee(1)**
send to system log	**logger(1)**
sort and/or merge files	**sort(1)**
split a file into multiple *n*-line pieces	**split(1)**

Index
Volume 1

Description	**Entry Name(Section)**
identify terminal type	**ttytype(1)**
`id` – print user and group IDs and names	**id(1)**
IDs and names, print user and group	**id(1)**
`ied` – input editor and command history for interactive programs	**ied(1)**
if – execute command if expression evaluates true	**csh(1)**
`if` – execute command if previous command returns exit status 0	**sh-posix(1)**
if – execute command if previous command returns exit status 0	**ksh(1)**
images on an X11 display, display TIFF file	**imageview(1)**
`imageview` – display bit-mapped file images on an X11 display	**imageview(1)**
include and conditional instructions, process C language	**cpp(1)**
incoming mailbox file, print out mail in the	**prmail(1)**
incoming messages from other users to terminal, deny or permit `write`(1)	**mesg(1)**
index, generate permuted	**ptx(1)**
information about specified cluster nodes, display	**cnodes(1)**
information about top processes on system, display and update	**top(1)**
information, current user, look up	**finger(1)**
information, display X.25 configuration	**x25stat(1)**
information, get locale-specific (NLS)	**locale(1)**
information, get Terminal Session Manager state	**tsm.info(1)**
information, SCCS identification, get from files	**what(1)**
initialize hard disk, flexible disk, or cartridge tape media	**mediainit(1)**
initialize terminal based on terminal type	**tset(1)**
input editor and command history for interactive programs	**ied(1)**
input, read a line from standard	**read(1)**
input single line from user keyboard	**line(1)**
insert calls to `catgets`(3C) based on `findstr`(1) output	**insertmsg(1)**
`insertmsg` – use `findstr`(1) output to insert calls to `catgets`(3C)	**insertmsg(1)**
instruction sequence, assembly language, time an	**atime(1)**
interactive ALLBASE/SQL SQL interface	**isql(1)**
interactive database access environment	**isql(1)**
interactively write (talk) to another user	**write(1)**
interactive mail message processing system	**mailx(1)**
interactive programs, input editor and command history for	**ied(1)**
interface, ALLBASE/SQL interactive SQL	**isql(1)**
interface status, display X.25	**x25stat(1)**
interface tool, LISP SoftBench	**lispbench(1)**
interface to the TELNET protocol, user	**telnet(1)**
internal attributes, change program's	**chatr(1)**
interpret ASA carriage control characters	**asa(1)**
interpreter, command (shell) with C-like syntax	**csh(1)**
interpreter/compiler for modest-sized programs	**bs(1)**
inter-procedural checker for FORTRAN	**lintfor(1)**
inter-process communication facilities, report status	**ipcs(1)**
interval, suspend execution for a time	**sleep(1)**
`intro` – introduction to command utilities and application programs	**intro(1)**
introduction to glossary of terms	**intro(9)**
`inv` – make unprintable characters in a file invisible	**vis(1)**
I/O, faster tape	**ftio(1)**
I/O statistics, report	**iostat(1)**
`iostat` – report I/O statistics	**iostat(1)**
`ipcrm` – remove a message queue, semaphore set, or shared memory identifier	**ipcrm(1)**
`ipcs` – report inter-process communication facilities status	**ipcs(1)**
`isql` – ALLBASE/SQL interactive SQL interface	**isql(1)**
job control, **uucp** status inquiry and	**uustat(1)**
jobs – list active jobs	**csh(1)**
jobs – list active jobs	**ksh(1)**
`jobs` – list active jobs	**sh-posix(1)**
join corresponding lines of several files or subsequent lines of one file	**paste(1)**

Index
Volume 1

Description	**Entry Name(Section)**
`login` – log in on system	**login(1)**
login name, change to super-user or another user	**su(1)**
login name, get	**logname(1)**
log in on another system over LAN	**vt(1)**
log in on system	**login(1)**
login password, change	**passwd(1)**
login password in Network Information System, change	**yppasswd(1)**
login, remote	**rlogin(1)**
login shell, redefine default in `passwd` file	**chsh(1)**
login – terminate login shell	**csh(1)**
log in to a new group	**newgrp(1)**
log messages and other information about RCS files, print	**rlog(1)**
`logname` – get login name	**logname(1)**
log of `uucp` and `uux` transactions, access summary	**uucp(1)**
logouts, hangups, and quits, run a command immune to	**nohup(1)**
logout – terminate login shell	**csh(1)**
long files, find differences between	**bdiff(1)**
long lines, fold for finite-width output device	**fold(1)**
look up current user information	**finger(1)**
`lorder` – find ordering relation for an object code library	**lorder(1)**
low priority, run a command at	**nice(1)**
`lpalt` – alter requests to an LP line printer or plotter	**lp(1)**
`lpfilter` – filters invoked by lp interface scripts	**lpfilter(1)**
LP line printer, send or cancel requests to an	**lp(1)**
LP printers, enable/disable	**enable(1)**
LP requests: print status information	**lpstat(1)**
`lprpp` – laserjet filter	**lpfilter(1)**
`lp` – send requests to an LP line printer or plotter	**lp(1)**
`lpstat` – print LP request status information	**lpstat(1)**
`lsacl` – list access control lists (ACLs) of files	**lsacl(1)**
`lsf` – list directory contents, showing subdirectories and executable files	**ls(1)**
`ls` – list directory contents	**ls(1)**
`lsr` – recursively list directory contents and subtrees	**ls(1)**
`lsx` – list directory contents, sort across page instead of down	**ls(1)**
`m4` – macro processor	**m4(1)**
`machid` – provide truth value about processor type	**machid(1)**
machines, determine who is logged in on local network	**rusers(1)**
machines (RPC version), show host status of local	**rup(1)**
machines, show status of local	**ruptime(1)**
machines, show who is logged in on local	**rwho(1)**
macro processor	**m4(1)**
macros, check or print documents formatted using `mm`	**mm(1)**
magnetic tape manipulating program	**mt(1)**
mailboxes, notify users of new mail in	**newmail(1)**
mailbox file, print out mail in the incoming	**prmail(1)**
mailbox, read mail from specified	**readmail(1)**
mailer, convert binary file to ASCII for transmission by	**uuencode(1)**
mail folders, summarize by subject and sender	**mailfrom(1)**
`mailfrom` – summarize mail folders by subject and sender	**mailfrom(1)**
mail in the incoming mailbox file, print out	**prmail(1)**
mail message processing system, interactive	**mailx(1)**
mail, notify users of new	**newmail(1)**
mail, process through screen-oriented interface	**elm(1)**
mail, read from specified mailbox	**readmail(1)**
mail response: “I am not here” indication	**vacation(1)**
`mail` – send mail to users or read mail	**mail(1)**
`mailstats`: print mail traffic statistics	**mailstats(1)**
mail traffic statistics, print	**mailstats(1)**

Description	**Entry Name(Section)**
remove DOS files or directories	**dosrm(1)**
remove extra new-line characters from file	**rmnl(1)**
remove file or directory	**rm(1)**
remove multiple line-feeds from output	**ssp(1)**
remove nroff/troff, tbl, and neqn constructs	**deroff(1)**
remove preprocessor lines	**unifdef(1)**
remove reverse line-feeds and backspaces from text	**col(1)**
remove SDF file or directory	**sdfrm(1)**
remove symbol and line number information from an object file	**strip(1)**
remsh: execute from a remote shell	**remsh(1)**
rename directory	**mv(1)**
rename file	**mv(1)**
rename LIF files	**lifrename(1)**
renice – alter priority of running processes	**renice(1)**
repeated (adjacent) lines in a file, count, extract, or eliminate	**uniq(1)**
repeat – execute command more than once	**csh(1)**
repetitively affirmative responses	**yes(1)**
replace selected characters	**tr(1)**
report adjacent repeated lines in a file	**uniq(1)**
report disk usage	**du(1)**
reporter, system activity	**sar(1)**
report inter-process communication facilities status	**ipcs(1)**
report I/O statistics	**iostat(1)**
report number of free DOS disk clusters	**dosdf(1)**
report process status	**ps(1)**
report virtual memory statistics	**vmstat(1)**
requests, LP	see LP requests
requests to an LP line printer or plotter, send or cancel	**lp(1)**
reserve disk space	**prealloc(1)**
responses, repetitively affirmative	**yes(1)**
restricted mailer (send only)	**mail(1)**
return – exit function with return value	**sh(1)**
return network node name	**nodename(1)**
return network status	**netstat(1)**
return – shell function return to invoking script	**sh-posix(1)**
return – shell function return to invoking script	**ksh(1)**
reverse line-feeds and backspaces, remove from text	**col(1)**
reverse order, show last commands executed in	**lastcomm(1)**
reverse – reverse printer pages for collating	**lpfilter(1)**
reverse the left-to-right text character sequence in each line of a file	**rev(1)**
Revision Control System	(see RCS)
revisions, compare RCS	**rcsdiff(1)**
revisions, RCS, check in	**ci(1)**
revisions, RCS, check out	**co(1)**
rev – reverse the text character sequence in each line of a file	**rev(1)**
rewind magnetic tape	**mt(1)**
right or left justify lines for NLS printing	**nljust(1)**
rights, access, to file(s), list	**getaccess(1)**
rksh – restricted Korn shell command programming language	**ksh(1)**
rlogin: remote login	**rlogin(1)**
rlog – print log messages and other information about RCS files	**rlog(1)**
rmail – restricted mailer (send only)	**mail(1)**
rmdel – remove a delta from an SCCS file	**rmdel(1)**
rmdir – remove directories	**rmdir(1)**
rmnl – remove extra new-line characters from file	**rmnl(1)**
rm – remove file	**rm(1)**
route and path between hosts, compute shortest	**pathalias(1)**
router, electronic address	**pathalias(1)**

Index
Volume 1

Description	**Entry Name(Section)**
state information, get Terminal Session Manager	**tsm.info(1)**
statistics, display virtual circuit	**x25stat(1)**
statistics, print mail traffic	**mailstats(1)**
statistics, report I/O	**iostat(1)**
statistics, report virtual memory	**vmstat(1)**
status, display X.25 interface	**x25stat(1)**
status, exit, do nothing and return zero or non-zero	**true(1)**
status, host, of local machines (RPC version), show	**rup(1)**
status information, LP request, print	**lpstat(1)**
status inquiry and job control, `uucp`	**uustat(1)**
status of each invocation of `ppl`, give	**pplstat(1)**
status of inter-process communication facilities, report	**ipcs(1)**
status of local machines, show	**ruptime(1)**
status, report process	**ps(1)**
status show network	**netstat(1)**
storage, preallocate disk	**prealloc(1)**
streaming text editor	**sed(1)**
string or expression, search a file for a	**grep(1)**
strings: extract strings from C programs to implement shared strings	**xstr(1)**
strings, find for inclusion in message catalogs	**findstr(1)**
`strings` – find the printable strings in an object, or other binary, file	**strings(1)**
strip nroff/troff, tbl, and neqn constructs from a file	**deroff(1)**
`strip` – strip symbol and line number information from an object file	**strip(1)**
strip symbol and line number information from an object file	**strip(1)**
`stty` – set the options for a terminal port	**stty(1)**
subdirectories in directory, list	**ls(1)**
`su` – become super-user or another user	**su(1)**
subroutine call graph execution profile data, display	**gprof(1)**
substitute selected characters	**tr(1)**
summarize, add, modify, delete, or copy file access control lists (ACLs)	**chacl(1)**
summarize and print an SCCS file	**prs(1)**
summarize disk usage	**du(1)**
summary log of `uucp` and `uux` transactions, access	**uucp(1)**
`sum` – print checksum and block count of a file	**sum(1)**
super-user or another user, change login name to	**su(1)**
suspend execution for a time interval	**sleep(1)**
suspend foreground until background processes are finished	**wait(1)**
swap the left-to-right text character sequence in each line of a file	**rev(1)**
swap up to eight characters (to first tab in line) to end of line	**newform(1)**
switch – define switch statement	**csh(1)**
symbol and line number information, strip from an object file	**strip(1)**
symbolic debugger for C, C++, FORTRAN, and Pascal	**xdb(1)**
symbolic debugger for C, C++, FORTRAN, Pascal	**cdb(1)**
symbolic links between files or directories, create	**ln(1)**
symbol table for object code file, print (Series 300/400)	**nm_300(1)**
symbol table for object code file, print (Series 700/800)	**nm_800(1)**
syntax, a shell (command interpreter) with C-like	**csh(1)**
system activity reporter	**sar(1)**
system and user aliases, `elm`, verify and create	**elmalias(1)**
system, call another (UNIX); terminal emulator	**cu(1)**
system-clock date and time, print current or set to new value	**date(1)**
system, display and update information about top processes on	**top(1)**
system, list users currently on the	**users(1)**
system, log in on	**login(1)**
system log, make entries in	**logger(1)**
system over LAN, log in on another	**vt(1)**
systems, copy files between various networked	**dscopy(1)**
system, set or print name of current host	**hostname(1)**

Index
Volume 1

Description	**Entry Name(Section)**
wait – wait for process and report termination status	**sh(1)**
wc – count words, lines, and characters in a file	**wc(1)**
what – get SCCS identification information from files	**what(1)**
whence – define interpretation of name as a command	**sh-posix(1)**
whence – define interpretation of name as a command	**ksh(1)**
whereis – locate source, binary, and/or manual files for program	**whereis(1)**
which – locate a program file including aliases and paths	**which(1)**
while – execute commands while expression is non-zero	**sh-posix(1)**
while – execute commands while expression is non-zero	**csh(1)**
while – execute commands while expression is non-zero	**ksh(1)**
whoami – print effective current user id	**whoami(1)**
who is logged in on local machines, show	**rwho(1)**
who is logged in on local network machines, determine	**rusers(1)**
who is my mail from?	**from(1)**
who – who is using the system	**who(1)**
words, find hyphenated	**hyphen(1)**
words, lines, and characters in a file, count	**wc(1)**
working directory, change	**cd(1)**
working directory name, print	**pwd(1)**
write(1) messages from other users to terminal, deny or permit	**mesg(1)**
write end-of-file marks on magnetic tape	**mt(1)**
write – interactively write (talk) to another user	**write(1)**
write LIF volume header on file	**lifinit(1)**
X11 display, display TIFF file images on an	**imageview(1)**
X11 graphics, display TIFF file images on X11 display	**imageview(1)**
X.25/9000 interface with PAD support, pad emulation for	**padem(1)**
x25stat – display interface status, configuration information, and virtual circuit statistics.	**x25stat(1)**
xargs – construct argument list(s) and execute command	**xargs(1)**
xdb – C, C++, FORTRAN, and Pascal Symbolic Debugger	**xdb(1)**
xd – hexadecimal file dump	**od(1)**
XMODEM-protocol file transfer program	**umodem(1)**
xstr – extract strings from C programs to implement shared strings	**xstr(1)**
yacc – yet another compiler-compiler	**yacc(1)**
yes – repetitively affirmative responses	**yes(1)**
ypcat: print values in Network Information Service map	**ypcat(1)**
ypmatch: print the values of selected keys in Network Information Service map	**ypmatch(1)**
yppasswd: change login password in Network Information System	**yppasswd(1)**
ypwhich: list which host is Network Information System server or map master	**ypwhich(1)**
zcat, compress, uncompress – compress or expand data	**compress(1)**
zero-length file, create	**cat(1)**
zero-length file, create	**cp(1)**
zero-length file, create	**touch(1)**

HEWLETT PACKARD

Manual Part No.
B2355-90033

Manufacturing
Part No.
B2355-90033

B2355-90033