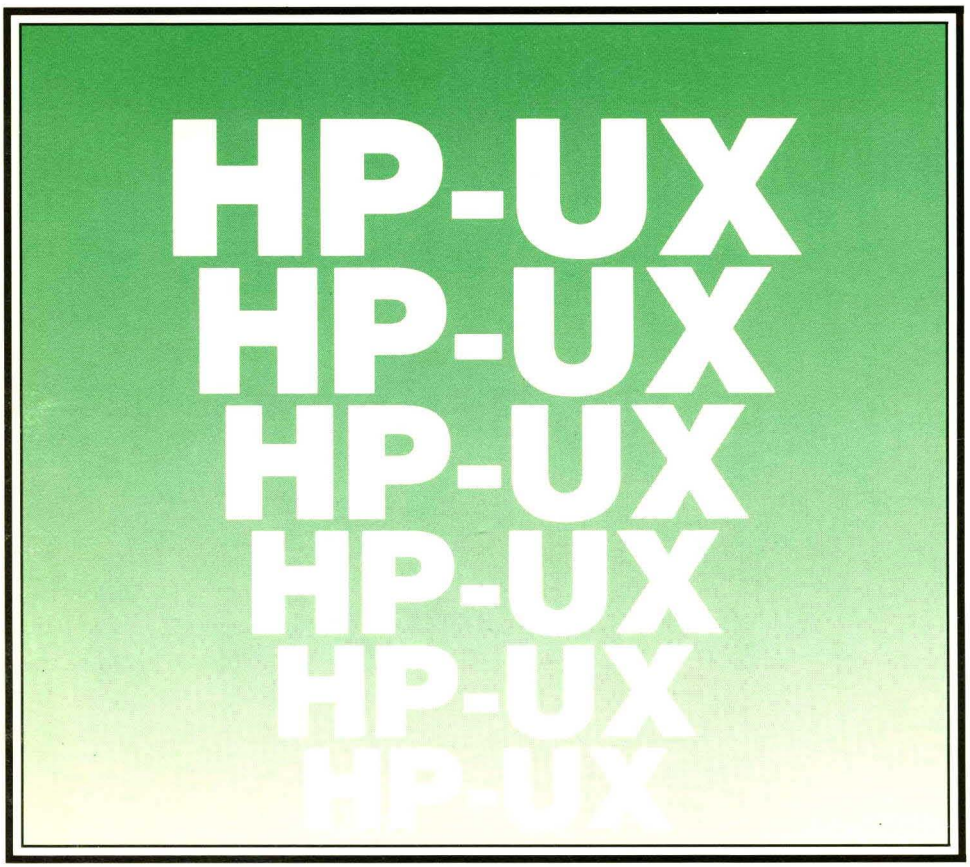
HEWLETT PACKARD

HP-UX Reference
Vol. 1A: Section 1



HP-UX Reference
Vol. 1A: Section 1 (A through L)

for

HP Part Number 09000-90008

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

July 1985...Edition 1. This manual replaces HP-UX Reference Manual 09000-90007 and documents HP-UX Release 5.0 for Series 200, 300 and 500.

November 1985...Edition 2. Updated from Edition 1 to reflect Series 200/300 HP-UX Release 5.1 changes. Several omitted pages in Edition 1 were also added.

June 1986...Edition 3. Update 1 incorporated.

September 1986...Edition 3 Update 1. This update reflects additions and changes incorporated in Series 500 HP-UX Release 5.1. Added command *autobackup*(1M) and core files support (*core*(5)), changed blocksize limitations for SDF file formats, and fixed various bugs.

TABLE OF CONTENTS

1. Commands

1M. System Maintenance Utilities

2. System Calls

3. Subroutines

4. Special Files

5. File Formats

6. Games

No games are currently supported.

7. Miscellaneous Facilities

INTRODUCTION

The *HP-UX Reference* describes the features of HP-UX in an alphabetical reference format arranged within several topical blocks. It is written for the user who is already familiar with UNIX or UNIX-like operating systems (UNIX is a trademark of AT&T technologies, Inc.). The reference is intended as a source for specific details concerning the HP-UX operating system.

For a general overview of HP-UX, see the supplied tutorial text *Introducing the UNIX System.* System implementation and maintenance details are explained in the *HP-UX System Administrator Manual.*

This manual is divided into several sections contained in four volumes:

Section 1 (*Commands and Application Programs*) describes programs intended to be invoked directly by users or command language procedures, as opposed to system calls (section 2) or subroutines (section 3) which are intended to be called by user programs. Commands usually reside in the directory **/bin** (for **bin**ary programs). Some 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 the shell (*sh*(1)). A few commands are also located in **/lib** and **/usr/lib**.

Section 1M (*System Maintenance Procedures*) 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 (*Subroutines*) *describes the available* Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files where they are stored.

Section 4 (*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 5 (*File Formats*) documents the structure of various kinds of files. For example, the link editor output-file format is described in *a.out*(5). Files that are used only by a single command (such as intermediate files used by assemblers) are not described. The C language **struct** declarations corresponding to the formats in this section can be found in the directories **/usr/include** and **/usr/include/sys**.

Section 6 (*Games*) is not present because no games are currently supported on HP-UX.

Section 7 (*Miscellaneous Facilities*) contains a variety of information such as descriptions of character sets, macro packages, and other topics.

Section 8 contains no topics. Items previously in Section 8 have been moved to Section 1M in Volume 2. *Intro*(8) is located as the last page in section 7. There is no tab divider for section 8.

Section 9 (*Glossary*) defines selected terms used in this manual.

Each section (except 9) consists of a number of independent entries of one or more pages each. The entry name appears on the upper corners of each page, and entries are arranged alphabetically (except for the introductory entry at the beginning of each section). Page numbering is arranged so that each entry starts on its own page 1. Some entries describe multiple commands, routines, etc. In such cases, the entry appears only once, arranged under its "major" name.

ENTRY FORMATS

All 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:

Boldface strings are literals, and are to be typed exactly as they appear in the manual.

Italic strings represent substitutable argument names and program names found elsewhere in the manual.

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 flag 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 =.

HP-UX COMPATIBILITY
shows the entry's HP-UX level and origin, based on the HP-UX Compatibility Model discussed later in this introduction. This part of the entry also shows whether an optional HP software package is required.

DESCRIPTION
discusses the function and behavior of each entry.

HARDWARE DEPENDENCIES
points out variations in HP-UX operation that are related to the use of specific hardware or combinations of hardware. **Any references to Series 200 computers in this manual set apply equally to Series 300 unless specifically noted to the contrary.**

EXAMPLES provides examples of typical usage, where appropriate.

FILES lists file names that are built into the program or command.

RETURN VALUE
discusses various values returned upon completion of program calls.

SEE ALSO provides pointers to related topics.

DIAGNOSTICS
discusses diagnostic indications that may be produced. Self-explanatory messages are not listed.

WARNINGS points out potential pitfalls.

BUGS discusses known bugs and observed deficiencies. Occasionally, suggested fixes are provided.

A table of contents and permuted index are included at the beginning and end, respectively, of each volume or binder. On each permuted index line, the title of the entry to which the line refers is followed by the appropriate section number listed in parentheses. This is important because names are frequently duplicated in various sections, particularly where commands were implemented to access certain system calls of the same name.

HOW TO GET STARTED

This discussion provides the basic information you need to get started on HP-UX: how to log in and log out, how to communicate through your machine, and how to run a program. (See the supplied tutorial text for more complete introduction to the system.)

Logging In

To log in you must have a valid user name, which may be obtained from the system. administrator of your system. Keep pressing the "break" or "del" until the **login:** message appears.

When a connection has been established, the system types **login:** and you then type your user name followed by pressing the "return" key (or "enter" key, on some terminals). If you have a password (and you should!), the system asks for it, but does not print it on the terminal.

It is important that you type in your login name in lower case if possible if you type upper-case letters, HP-UX assumes that your terminal cannot generate lower-case letters, and that all subsequent upper-case input is to be treated as lower case. When you have logged in successfully, the shell types a **$.** (The shell is described below under *How to run a program.*)

For more information, consult *login*(1) and *getty*(8), 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*(5) 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. The shell will terminate and the **login:** message will appear again.

How to Communicate Through Your Terminal

When you type to HP-UX, the system usually gathers your characters and saves them. These characters will not be given to a program until you type a "return" (or a "new-line").

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 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 in trouble. When the read-ahead limit is exceeded, the system throws away *all* the saved characters.

On an input line from the terminal, the character @ "kills" all characters typed before it. The character # erases the last character typed. Successive uses of # will erase characters back to, but not beyond, the beginning of the line; @ and # can be typed as themselves 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 useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when any character is typed. If **DC1** (control-q) or **DC3** are used to restart the program, they are not saved and passed to later programs. Any other characters are saved and passes as output to later programs.

The ASCII **DEL** character (sometimes labelled "rubout" or "rub") is not passed to programs, but instead generates an *interrupt signal* , just like the "break", "interrupt", or "attention" 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 editor printout without losing the file being edited.

The *quit* signal is generated by typing the ASCII octal 34 (control-\) character. It causes a running program to terminate.

Besides adapting to the speed of the terminal, HP-UX tries to be intelligent as to whether you have a terminal with the "new-line" key, or whether it must be simulated with a "carriage-return" and "line-feed" pair. In the latter case, all *input* "carriage-return" characters are changed to "line-feed" characters (the standard line delimiter), and a "carriage-return" and "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 (not currently supported on Series 500). The *stty*(1) command will set or reset this mode. The system assumes that tabs are set every eight character positions. The *tabs*(1) command will set tab stops on your terminal, if that is possible.

How to Run a Program

When you have successfully logged into HP-UX , a program called the shell is listening to you terminal. The shell reads the lines you type, splits them into command names and arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see *The current directory* 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 type a **$** at you 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 you 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).

Path Names

To refer to files not in the current directory, you must use a path name. Full 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 sub-directory of **usr** ; **usr** springs directly from the root directory). See the glossary for a formal definition of *path name.*

If you current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (*without* a prefixed /). Without important exception, a path name may be used anywhere a file name is required.

Important commands that modify the contents of files are *cp*(1), *mv*(1), and *rm*(1), which respectively copy, move (i.e., rename), and remove files. To find out the status of files or directories, use *ls*(1). Use *mkdir*(1) for making directories and *rmdir*(1) for destroying 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 *ed*(1), *ex*(1), or *vi*(1). The three principal languages available under HP-UX are C (see *cc*(1)), FORTRAN (see *fc*(1) *or 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 in the current directory named **a.out** (if that output is precious, use *mv*(1) to give it a less vulnerable name). If the program is written in assembly language, you will probably need to link library subroutines 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, argv,* and *envp* parameters. See the supplied C tutorial for details.

Text Processing

Almost all text is entered through editors *ed*(1), *ex*(1), or *vi*(1). The commands most often used to write text on a terminal are *cat*(1) and *pr*(1). The *cat*(1) command simply dumps ASCII text on the terminal, with no processing at all. The *pr*(1) command paginates the text, supplies headings, and has a facility for multi-column output.

Surprises

Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may direct them toward you. To communicate with another user currently loggin in, *write*(1) is used; *mail*(1) or *mailx*(1) will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these commands if you are their target.

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

HP-UX COMPATIBILITY MODEL

HP-UX is AT&T System V plus "HP value added". HP value added includes both Hewlett-Packard capabilities, such as graphics, and features from other UNIX systems, such as those from the University of California at Berkeley.

Compatibility Levels

The various HP-UX systems are listed below in order of increasing completeness; each contains all the elements of the previous one.

HP-UX/RUN ONLY — This describes a run-only kernel with no commands or applications attached.

HP-UX/NUCLEUS — This is the run-only kernel plus a minimum set of commands. It also provides a minimum command interpreter to permit access to the commands.

HP-UX DEVELOPMENT — This is the first "normal" UNIX , but it does not include the full UNIX command set.

HP-UX STANDARD — This is a nearly complete UNIX. It includes most of the capabilities from AT&T, but not everything that HP will make available.

HP-UX EXTENDED — This is the largest standard package. It contains almost everything HP-UX has to offer (a few AT&T capabilities are not included).

OPTIONAL — For the purposes of the model, there are also capabilities that are never required, even at the **HP-UX/EXTENDED** level. The term **OPTIONAL** designates capabilities in this category.

NON-STANDARD — This designation is given to those keywords which have either not yet been approved as part of the HP-UX standard, or never will be.

NAME

intro - introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, publicly–accessible commands. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.
(1C) Commands for communication with other systems.
(1G) Commands used primarily for graphics and computer–aided design.

COMMAND SYNTAX

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

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

where:

name	The name of an executable file.
option	- *noargletter*(*s*) or, - *argletter*<>*optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with - or, - by itself indicating the standard input.

HP–UX COMPATIBILITY

Level: This describes where in the HP–UX compatibility model this capability appears. See the *Introduction* to this manual for a detailed explanation of the model.

Origin: This gives authorship credit as appropriate. The following abbreviations are used:

System III	means from Bell UNIX System III.
System V	means from AT&T UNIX System V (release 2 unless noted otherwise).
HP	means written by HP.
UCB	means derived from U. C. Berkeley 4.1BSD.
V7	means included for UNIX Version 7 compatibility (and not in Bell System V).

Requires: This indicates any special hardware or software requirements for the code to operate properly. If a capability deviates from the HP–UX standard, the deviations will be displayed in one of two ways. Minor deviations will be in separate sections in the body of the manual. New pages will be generated where necessary, and the top center of the page will indicate the deviation.

Remarks: identifies which implementation(s) are described by the manual page.

DESCRIPTION

This section describes, in alphabetical order, publicly–accessible commands. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.
(1C) Commands for communication with other systems.

(1G) Commands used primarily for graphics and computer–aided design.
(1M) Commands used primarily for system maintenance.

HARDWARE DEPENDENCIES

This section gives details about specific implementations of HP–UX that deviate from information already given for that manual entry. It is very important that you check this section, if present, to make sure that certain options and/or capabilities are implemented on your computer. If there are extensive changes, new manual pages are generated and flagged as being implementation specific.

SEE ALSO

The **SEE ALSO** entries are chosen in part to guide the reader to related topics that might prove useful. The list may not always be relevant, depending on the user's needs. **SEE ALSO** entries may refer to capabilities not available in all implementations if they are relevant in the more complete implementations. Examples of **SEE ALSO** entries are:

getopt(1), exit(2), wait(2), getopt(3C).
Introduction to the HP–UX Reference at the front of this volume.

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non–zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

Unfortunately, many commands do not adhere to the aforementioned syntax.

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 upon encountering a null character (the string terminator) within a line.

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

acctcom [[options][file]] . . .

HP-UX COMPATIBILITY

Level: HP-UX/EXTENDED - multi-user

Origin: System III

DESCRIPTION

Acctcom reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(5) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME**, **USER**, **TTYNAME**, **START TIME**, **END TIME**, **REAL (SEC)**, **CPU (SEC)**, **MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: **1** for *fork* without *exec*), **STAT** (the system exit status), **HOG FACTOR**, **KCORE MIN**, **CPU FACTOR**, **CHARS TRNSFD**, and **BLOCKS READ** (total blocks read and written).

The command name is prepended with a # if it was executed with *super-user* privileges. If a process is not associated with a known terminal, a **?** is printed in the **TTYNAME** field.

If no *files* are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using **&** in the shell), **/usr/adm/pacct** is read; otherwise, the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/usr/adm/pacct***?*. The *options* are:

-a Show some average statistics about the processes selected. The statistics will be printed after the output records.

-b Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.

-f Print the *fork/exec* flag and system exit status columns in the output.

-h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:
(total CPU time)/(elapsed time).

-i Print columns containing the I/O counts in the output.

-k Instead of memory size, show total kcore-minutes.

-m Show mean core size (the default).

-r Show CPU factor (user time/(system-time + user-time).

-t Show separate system and user CPU times.

-v Exclude column headings from the output.

-l *line* Show only processes belonging to terminal **/dev/***line*.

-u *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a # which designates only those processes executed with *super-user* privileges, or **?** which designates only those processes associated with unknown user IDs.

-g *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

-s *time* Select processes existing at or after *time*, given in the format *hr*[:*min*[:*sec*]].

-e *time* Select processes existing at or before *time*.

-S *time* Select processes starting at or after *time*.

-E *time* Select processes ending at or before *time*. Using the same *time* for both **-S** and **-E** shows the processes that existed at *time*.

-n *pattern* Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that + means one or more occurrences.

-q Do not print any output records, just print the average statistics as with the **-a** option.

-o *ofile* Copy selected process records in the input data format to *ofile*; supress standard output printing.

-H *factor* Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **-h** above.

-O *time* Show only those processes with operating system CPU time that exceeds *time*.

-C *sec* Show only processes with total CPU time, system plus user, exceeding *sec* seconds.

-I *chars* Show only processes transferring more characters than the cut-off number given by *chars*.

Listing options together has the effect of a logical *and*.

FILES

/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO

ps(1), su(1), acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time, then *time* is interpreted as occurring on the previous day.

NAME

adb - debugger

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: System III

Remarks: *Adb* is implemented on the Series 200 only.

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of HP-UX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then 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 the standard input and responses are to the standard output. If the **-w** flag is present then *objfil* is created if necessary and opened for reading and writing so that it can be modified using *adb*. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

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

If *address* is present then *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will 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 then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see *ADDRESSES*.

EXPRESSIONS

. The value of *dot*.

\+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

integer An octal number if *integer* begins with a 0; a hexadecimal number if preceded by 0x; a decimal number if preceded by 0d; otherwise the base of *integer* is whatever the default number base for input is. This base is initialized to hexadecimal.

integer.fraction
A 32 bit floating point number.

'*cccc*' The ASCII value of up to 4 characters. \ may be used to escape a '.

< *name*
The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (see *VARIABLES*) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil*. The register names are **a0 ... a6 d0 ... d7 sp pc ps**.

symbol A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ will be inserted at the beginning of *symbol* if needed.

_ *symbol*
In C, the "true name" of an external symbol begins with _. It may be necessary to utter this name to distinguish it from a symbol generated in assembly language.

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

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.

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 a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands **?** and **/** may 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.

/*f* Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for **?**.

=*f* The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format **?** is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows:

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. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@. (The sign bit is ignored.)

s *n* Print the addressed characters until a zero character is reached. *N* is the length of the string, including the zero terminator.

S *n* Print a string using the @ escape convention. *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 MC68000 instructions. *n* is the number of bytes occupied by the instruction.

I *n* Same as **i**, except that immediate constants are printed in decimal.

a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/ local or global data symbol
? local or global text symbol
= local or global absolute symbol

p 4 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.

t 0 When preceded by an integer tabs 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.

"..." 0 Print the enclosed string.

^ *Dot* is decremented by the current increment. Nothing is printed.

\+ *Dot* is incremented by 1. Nothing is printed.

\- *Dot* is decremented by 1. Nothing is printed.

new–line

Repeat the previous command with a *count* of 1. Also can be used to repeat a **:s** or **:c** command.

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

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted then -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** *b1 e1 f1*[?/]

New values for (*b1, e1, f1*) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the **?** or **/** is followed by * then the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If the list is terminated by **?** or **/** then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, **/m?** will cause **/** to refer to *objfil*.)

>*name* *Dot* is assigned to the variable or register named.

! A shell is called to read the rest of the line following **!**.

$*modifier*

Miscellaneous commands. The available *modifiers* are:

<*f* Read commands from the file *f* and return.

>*f* Send output to the file *f*, which is created if it does not exist.

r Print the general registers and the instruction addressed by **pc**. *Dot* is set to **pc**.
b Print all breakpoints and their associated counts and commands.
c C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **a6**). If *count* is given then 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* (default 255).
o The default for all integers input is octal.
d The default for all integers input is decimal.
x The default for all integers input is hexadecimal.
q Exit from *adb*.
v Print all non zero variables in octal.
n Set the number of significant digits for floating point dump to *address*.
m Print the address map.

new–line
print the process id and register values.

:*modifier*

Manage a subprocess. Available modifiers are:

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 then the breakpoint causes a stop.
d Delete breakpoint at *address*.
d* Delete all breakpoints
r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be 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 on entry to the subprocess.
e Setup a subprocess as in **r**; no instructions are executed.
c*s* The subprocess is continued with signal *s* (see *signal*(2)). If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.
s*s* As for **c** except that the subprocess is single stepped *count* times.
S*s* As for **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 The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of 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.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a **core** file then these values are set from *objfil*.

b The base address of the data segment.
d The data segment size.
e The entry point.
m The "magic" number as defined in *magic.h*.
s The stack segment size.
t The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows:

If *address* is greater than or equal to *b1* **and** *address* is less than *e1*, then

file address=address+f1-b1

Otherwise,

if *address* is greater than or equal to *b2* **and** *address* is less than *e2*, then

file address=address+f2-b2

Otherwise, the requested *address* is not legal. If a **?** or **/** is followed by an * then only the second triple is used.

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

In order for *adb* to be used on large files, all appropriate values are kept as signed 32 bit integers.

FILES

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

SEE ALSO

ptrace(2), a.out(5), core(5).

DIAGNOSTICS

"Adb" when there is no current command or format, and comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned non-zero status.

BUGS

Local variables whose names are the same as an external variable may foul up the accessing of the external.

NAME

adjust - simple text formatter

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

DESCRIPTION

This command is a simple text formatter for filling, centering, left- and right-justifying, or right-justifying text paragraphs, 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. (You can use "—" as an argument to separate "–" from options.)

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

Options are:

-b Do not convert leading blanks to tabs on output. Thus there are no tabs in the output.

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

-j Justify text. After filling, insert blanks in each line (except the last line of each paragraph) as needed to right-justify, 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 at once.

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. Then it expands all tabs to blanks.

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 blank. "Words" ending in <terminal>[<quote>][<close>] are followed by two blanks, where <terminal> is any of ".:?!", <quote> is a single or double quote, and <close> is any of ")]}", e.g.:

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

The program starts a new output line whenever adding a word (other than the first one) to the current line would pass the right margin.

Adjust understands indented first lines of paragraphs (like this one) when filling. The second and subsequent lines of each paragraph are indented the same amount as the second line of the paragraph in the input, if there is a second line; otherwise they are indented the same as the first line.

* **Adjust** has a rudimentary understanding of tagged paragraphs (like 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, then the column positions of the tag words (before that one) are "frozen" (not altered).

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 blanks in output lines where they are most needed, until the lines extend to the right margin. First, all one–blank word separators are examined. One blank is added first to those separators with the most total letters in the words on both sides. If all one–blank separators are increased to two blanks, and more blanks must be inserted, it repeats the algorithm, this time with two–blank 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.

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.

You can give the **vi** command:

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

(where "^" denotes control characters) to set up a useful "finger macro". Then typing ^X will reformat the entire current paragraph.

Note that **adjust –m1** is a simple way to break text into separate words, without white space, except for tagged–paragraphs tags.

SEE ALSO

nroff(1)

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 of **-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 blanks).

BUGS

This program is designed to be simple and fast. It does not recognize backslash to escape white space or anything else. It does not recognize tagged paragraphs where the tag is on a line by itself. It knows that lines end in newline or null, and how to deal with tabs and backspaces, but it does not do anything special with other characters like form feed (they are just ignored). For complex operations, the 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.

These options were considered but not added, because the creeping featurism had to stop somewhere, before this program became another *nroff*(1):

-h Hyphenate. Allows the program to break and join words at existing hyphens (only). Words are broken across lines, at single hyphens surrounded by non–whitespace characters. Likewise, a word ending in a single hyphen, followed by whitespace, followed by a non–hyphen character, is joined to the next word without whitespace if needed.

-n Nofill. Only allowed with **-j** or **-r**, it inhibits filling, i.e. words are not moved from one line to another. Thus existing text can be left/right or right–justified without being otherwise modified. (Note that **-n** is always in effect for **-c**, centering.)

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**] files

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System III

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 keyletter arguments, which begin with **-**, and named files (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 keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* 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. Again, non–SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

-n This keyletter 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** keyletter for delta numbering scheme). If the **i** keyletter 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 keyletter is omitted, then the SCCS file is created with an empty initial delta. Only one SCCS file may be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **-i** keyletter). Note that the **-i** keyletter implies the **-n** keyletter.

-r*rel* The *rel*ease into which the initial delta is inserted. This keyletter may be used only if the **-i** keyletter is also used. If the **-r** keyletter 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** keyletter is used and *admin* is creating a new SCCS file (the **-n** and/or **-i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a **-t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **-t** keyletter 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 keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single *admin* command line. The allowable *flag*s and their values are:

b Allows use of the **-b** keyletter 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 may 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*(1) command.

i[*str*] Causes the "No id keywords (cm7)" message issued by *get*(1) or *delta*(1) to be treated as a fatal error. 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*(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

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*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may 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*(1).

m*mod* *Mod*ule name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*(1). 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*(1).

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** keyletter 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** keyletter may be specified only when processing existing SCCS files. Several **-d** keyletters may be supplied on a single *admin* command. See the **-f** keyletter for allowable *flag* names.

l*list* A *list* of releases to be "unlocked". See the **-f** keyletter 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 which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters may be used on a single *admin* command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may 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** keyletters may 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** keyletter 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** keyletter is valid only if the **-i** and/or **-n** keyletters are specified (i.e., a new SCCS file is being created).

-m[*mrlist*] The list of Modification Requests (*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). Diagnostics will 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*(5)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

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

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 with 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 the directories allows only the owner to modify SCCS files contained in the directories. 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 may be changed to 644 by the owner allowing use of *ed*(1). *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(5).
SCCS User's Guide in *HP-UX Concepts and Tutorials.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar - archive and library maintainer for portable archives

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

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. It can be used, though, for any similar purpose. The magic string and the 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. The portable archive format and structure is described in detail in *ar*(4). The archive symbol table (described in *ar*(4)) is used by the link editor (*ld*(1)) to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *ar* when there is at least one object file in the archive. The archive symbol table is in a specially named file which 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 such an archive, the symbol table is rebuilt. The **s** option described below will force the symbol table to be rebuilt.

Key must be present, and is an optional **-**, followed by one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcls**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters for operations on an archive are:

d Delete the named files from the archive file.

r Replace the named files, or add a new file to the archive. If the optional character **u** is used with **r**, then 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, then the *posname* argument must be present and specifies that new copies of the named 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* will create *afile* if it does not already exist. If there are no file *names*, *ar* will create an empty archive file whose only contents is the archive header (see *ar*(5)).

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check 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* will create *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 to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, 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 meanings of the remaining optional modifying characters are:

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**, it gives a long listing of all information about the files. When used with the **d**, **m**, **p**, **q**, and **x** options, the verbose option causes *ar* to print the key letter and file name associated with each file for that operation. For the **r** operation, *ar* will show an "a" if it added a new file, or an "r" if it replaced an existing one.

c Create. Normally *ar* will create *afile* when it needs to (for the **r** and **q** operations). The create option suppresses the normal message that is produced when *afile* is created.

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 **/tmp**. Only the **d**, **m**, **r** and **s** options use temporary files.

s Force the regeneration of the archive symbol table even if *ar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

Only the following combinations are meaningful:

d: **v**, **l**
r: **u**, **v**, **c**, **l**, and **a** | **b** | **i**
q: **v**, **c**
t: **v**, **s**
p: **v**, **s**
m: **v**, **l**, and **a** | **b** | **i**
x: **v**, **s**
For other combinations of modifiers with operations not shown in the above table, the modifier has no effect.

EXAMPLES

The command

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

will create a new file (if one does not already exist) in archive format with its constituents entered in the order shown in the above command line.

If you want to replace files f2 and f3 such that the new copies follow file f1, the commands

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

will produce the desired effect. The archive will now be ordered

```
newlib.a: f1 f2´ f3´ f4
```

where the single quote marks indicate updated files.

FILES

/tmp/ar* temporaries

SEE ALSO

arcv(1), ld(1), lorder(1), strip(1). tmpnam(3S), a.out(5), ar(5), ranlib(5).

WARNING

If you are the super–user, *ar* will alter any archive file, even if it is write–protected.

NOTES

This archive format is new to this release. The *arcv*(1) command can be used to change an older archive file into an archive file that is recognized by this *ar* command.

BUGS

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

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.

NAME

arcv - convert archives to new format

SYNOPSIS

arcv file ...

HP–UX COMPATIBILTY

Level: HP–UX/STANDARD

Origin: UCB

DESCRIPTION

Arcv converts archive files (see *ar*(1), and *ar*(5)) from a pre-HP–UX 5.0 format to the HP–UX 5.0 portable archive format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177545 at the start; new archives have a first line "!<arch>".

FILES

/tmp/arc*

SEE ALSO

ar(1), ar(5).

NAME

as - assembler for MC68000

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPPMENT

Origin: System III

Remarks: *As* is implemented on the Series 200 only.

DESCRIPTION

As assembles the named *file*, or the standard input if no file name is specified. The optional arguments **-A** or **-a** may be used to obtain an assembly listing with offsets and instruction codes listed in hex. 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*.

FILES

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

SEE ALSO

adb(1), ld(1), nm(1), a.out(5).
MC68000 Assembler on HP-UX, in *HP-UX Concepts and Tutorials.*

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 typed out together with the line number and the file name in which it occurred.

NAME

asa - interpret ASA carriage control characters

SYNOPSIS

asa [files]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

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 no file names are supplied. The first character of each line is assumed to be a control character; their meanings are:

' '	(blank) single new line before printing
0	double new line before printing
1	new page before printing
+	overprint previous line.

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

To view correctly the output of FORTRAN programs which use ASA carriage control characters, *asa* could be used as a filter thus:

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

and the output, properly formatted and paginated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by:

```
asa file
```

SEE ALSO

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

NAME

at, batch - execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]
at *–r*job...
at *–l[job...]*

batch

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

Native Language Support:
8-bit filenames.

DESCRIPTION

At and *batch* read commands from standard input to be executed at a later time. *At* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. *At* **–r** removes jobs previously scheduled with *at*. The **–l** option reports all jobs scheduled for the invoking user.

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

Users are permitted to use *at* if their name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the 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 either file is **at.deny**, global usage is permitted. The allow/deny files consist of one user name per line.

The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour:minute*. A suffix **am** or **pm** may be appended; otherwise a 24–hour clock time is understood. The suffix **zulu** may be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", **today** and **tomorrow** are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **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.

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.)

Thus legitimate commands include:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

At and *batch* write the job number and schedule time to standard error.

Batch submits a batch job. It is almost equivalent to "at now", but not quite. For one, it goes into a different queue. For another, "at now" will respond with the error message too late.

At –r removes jobs previously scheduled by *at* or *batch*. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at* –l. You can only remove your own jobs unless you are the super–user.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *Sh*(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
nroff filename >outfile
<control–D> (hold down 'control' and depress 'D')
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

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

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
```

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/lib/cron/queue – scheduling information
/usr/spool/cron/atjobs – spool area

SEE ALSO

crontab(1), kill(1), mail(1), nice(1), ps(1), sh(1), cron(1M).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

aterm - general purpose asynchronous terminal emulation

SYNOPSIS

aterm configfile

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Native Language Support:
8-bit data.

Remarks: *Aterm* is implemented on the Series 500 only.

DESCRIPTION

Aterm is a general purpose asynchronous terminal emulator designed for maximum connection flexibility and simple file transfers without remote host support. Transparent pass-through mode provides all user terminal capabilities in multi-user systems.

Configfile is used by *aterm* to match the particular terminal configuration needed for the remote system you are logging onto. This file consists of configuration commands, one to a line. Each line consists of the command name and its arguments, if any. Only configuration parameters which differ from the standard default need be specified. Most configuration commands can also be given from the keyboard while the emulator is running. You can exit *aterm* by typing "~.".

The following list shows the recognized configuration command names:

da Serial device file name (no default);

hn Name of remote computer system (no default);

db Number of data bits per character: 5, 6, 7, or 8 (default = 7);

sb Number of stop bits per character: 1, 1.5, or 2 (default = 1);

pa Character parity: none (n), odd (o), even (e), zero (0), or one (1) (default = o);

dr Rate for data sent and received: 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 9600, or 19200 baud (default = 2400 baud);

mc Modem control: enabled (+) for full-duplex modem, or disabled (-) for full-duplex hard-wired connection (default = -);

ss Switched service: auto-answer (a) or manual originate (o) (default = o);

ga Gap: number of character transmission times to delay between successive output characters; values range from 0 to 254 (default = 0);

ec Echo: enabled (+) if the host computer echos characters sent by the emulator, disabled (-) otherwise (default = -);

te Terminal ENQ/ACK: enabled (+) or disabled (-) (default = +);

he Host ENQ/ACK: enabled (+) or disabled (-) (default = -);

tx Terminal XON/XOFF: enabled (+) or disabled (-) (default = -);

hx Host XON/XOFF: enabled (+) or disabled (-) (default = -);

im Input mode: block (b), character (c), or line (l) (default = b);

om Output mode: character (c) or line (l) (default = c);

ph Prompt handshake: if enabled (+), the emulator waits for the prompt sequence before sending each line of data during an input diversion; if disabled (-), no wait is performed (default = -);

pt Prompt timeout: number of seconds to allow for receipt of a prompt sequence during an input diversion; values range from 1 to 600, with 0 disabling the timeout altogether (default = 0);

st Single text terminators: list of characters, any of which terminates a line sent by the host computer when the emulator is in input line mode; up to eight characters may be specified (default = no characters);

dt Double text terminator: a pair of characters which together terminate a line sent by the host computer when the emulator is in input line mode (default = carriage–return/line–feed);

ps Prompt sequence: one or two characters which terminate a line sent by the host computer when the emulator is in input line mode, and which satisfy the prompt handshake if enabled (default = DC1);

bl Beginning of line: character to be prefixed to each line sent to the host computer (default = none);

el End of line: one or two characters to be postfixed to each line sent to the host computer (default = carriage–return);

es Local command character: character which designates a local command to be interpreted by the emulator if it comes at the beginning of a line read from the standard input (default = ~).

Note that emulation does not include block or format modes.

SEE ALSO

cu(1C) if simple connections are adequate or if you are calling another HP–UX system;
uucp(1C) for file transfers with other HP–UX systems.

HP–UX Network Communications Guide.

BUGS

Does not work with 6-channel multiplexer.

NAME

atrans - translate assembly language

SYNOPSIS

atrans [**-j**] [**-n**] [filename]

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: HP

Remarks: *Atrans* is implemented on the Series 200 and Integral PC only.

DESCRIPTION

Atrans translates an assembly language *source file* from **Series 200** Pascal workstation assembly language syntax to **Series 200** HP-UX assembly language syntax. If no *filename* is given, input is assumed to come from **stdin.**

All uppercase characters are converted to lowercase characters, except those in comments or in quoted strings.

Hexadecimal constants are converted from **Series 200** Pascal workstation syntax, *£<hex number>* to the **Series 200** HP-UX syntax, *0x<hex number>*.

Operands whose effective address is *program counter* with displacement will have the string *pc* inserted in them (e.g. 8(d6) will become 8(pc,d6)).

Lines containing the following list of **Series 200** Pascal workstation pseudo-ops have no parallel in **Series 200** HP-UX syntax and are translated as comment lines: *decimal, end, llen, list, lprint, nolist, noobj, nosyms, page, spc, sprint, ttl.*

Lines containing the *mname* or *src* pseudo-ops are translated as comment lines, and a warning is printed stating that modules are not supported by the *Series 200* HP-UX assembler.

The pseudo-ops, *def, refa,* and *refr,* are translated as *globl.*

The file name operand of an *include* pseudo-op will have quote marks put around it.

Certain pseudo-ops require manual intervention to translate. Each Line containing these pseudo_ops will cause a message to be printed stating that an error will be generated by the **Series 200** HP-UX assembler. These pseudo-ops are: *com, lmode, org, rorg, rmode, smode, start.*

The **-j** option converts opcodes with the **b***cc*[**.s**|**.l**] branch syntax to the **j***cc* syntax. It also converts *bsr[.s|.l]* to *jbsr*.

The **-n** option converts groups of blanks to tabs.

SEE ALSO

as(1).

NAME

awk – text pattern scanning and processing language

SYNOPSIS

awk [**–F** c] [**–f** file | prog] [parameters] [files]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as –f *file*. The *prog* string should be enclosed in single quotes (*'*) to protect it from the shell.

Parameters, in the form x=... y=... etc., may be passed to *awk*.

Files are read in order; if there are no files, the standard input is read. The file name – means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the form:

pattern { action }

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, –, *, /, %, and concatenation (indicated by a blank). The **C** operators ++, --, +=, -=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (*"*). Single quotes (*'*) are not recognized.

The *print* statement prints its arguments on the standard output (or on a file if *>expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format (see *printf*(3S)).

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf*(*fmt*, *expr*, *expr*, ...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see *grep*(1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

where a relop is any of the six relational operators in C, and a matchop is either ~ (for *contains*) or !~ (for *does not contain*). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

BEGIN { FS = *c* }

or by using the **–F***c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default **%.6g**).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

```
{ for (i = NF; i > 0; —i) print $i }
```

Print all lines between start/stop pairs:

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

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
       { print }
```

command line: awk –f program n=5 input

SEE ALSO

grep(1), lex(1), sed(1), malloc(3x).
Awk: A Programming Language for Manipulating Data in *HP-UX Concepts and Tutorials.*

BUGS

Input white space is not preserved on output if fields are involved.
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 the null string ("") to it.

NAME

banner - make posters in large letters

SYNOPSIS

banner strings

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

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

Each argument is on a separate line.

SEE ALSO

echo(1).

ad.b

NAME

basename, dirname – extract portions of path names

SYNOPSIS

basename string [suffix]
dirname string

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8–bit filenames.

DESCRIPTION

Basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside command substitution marks (`...`) within shell procedures. If *string* does not contain the indicated suffix, *basename* returns an unpredictable value consisting either of a single character string, or the null string.

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

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 will set the shell variable **NAME** to **/usr/src/cmd**:

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

RETURN VALUE

Both commands return 0 for success, 1 for failure. *Dirname* always succeeds, and thus always returns 0.

SEE ALSO

expr(1), sh(1).

BUGS

When using *basename*, be aware that suffixes are not guaranteed to occur at the end of the string. Thus,

```
basename file.c.old .c
```

returns "file".

NAME

bc - arbitrary–precision arithmetic language

SYNOPSIS

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

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

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. The options are as follows:

-c compile only. *Bc* is actually a preprocessor for *dc*(1), which *bc* invokes automatically. Specifying **-c** prohibits invocation of *dc*, and sends the *dc* input to the standard output.

-l causes an arbitrary precision math library to be pre–defined.

The syntax for *bc* programs is as follows;

L means a letter in the range a-z;
E means expression;
S means statement;
R means relational expression.

Comments
are enclosed in /* and */.

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

Other operands
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 (yield an E as a result)
+ - * / % ^ (% is remainder; ^ is power)
++ -- (prefix and postfix; apply to names)
= =+ =- =* =/ =% =^

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 the **-l** math library:

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.

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 may 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 may be used as an array, a function, and a simple variable simultaneously. 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.

EXAMPLE

```
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
        }
}
```

defines a function to compute an approximate value of the exponential function, and

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

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

FILES

/usr/lib/lib.b	mathematical library
/usr/bin/dc	desk calculator proper

SEE ALSO

bs(1), dc(1).

BUGS

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 like 2+2 will tend to get it back into phase.

NAME

bdiff - big diff

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Bdiff is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is **-**, the standard input is read. The optional **-s** (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*). If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

bfs - big file scanner

SYNOPSIS

bfs [**-**] name

HP-UX COMPATIBILITY

Level: HP-UX STANDARD

Origin: System V

DESCRIPTION

The *Bfs* command is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

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 carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may 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** may be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **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** will 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**. The **xf** commands may be nested to a depth of 10.

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

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. The *label* is terminated by new-line, and blanks between the **:** and the start of the *label* are ignored. This command may 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, so it may 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.

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**. The commands **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. The command **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
```

will all print the first 100 lines.

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

would globally search for the characters **100** and print each line containing a match. To escape the special meaning of %, a \ must precede it.

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

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.
Another feature of the **xv** command is that the first line of output from an HP-UX system 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
```

would 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 will test the last saved *return code* from the execution of a HP-UX system command (**!***command*) for a zero or nonzero value, respectively, and cause a branch to the specified label. The two examples below both search the file for the next five lines which contain the string **size**.

```
xv55
: l
/size/
xv5!expr %5 - 1
!if [ %5 != 0 ] ; then exit 2 ; fi
xbn l
```

```
xv45
: l
/size/
xv4!expr %4 - 1
!if [ %4 = 0 ] ; then exit 2 ; fi
xbz l
```

xc [*switch*]

If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it is not. 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.

SEE ALSO

csplit(1), ed(1), regex(3), regcmp(3X).

DIAGNOSTICS

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

BUGS

When searching a file which contains a line that is longer than 512 characters (including the new-line), a message "line too long – output truncated," will be printed every time that line is searched.

NAME

bifchmod - change mode of a BIF file

SYNOPSIS

bifchmod mode device:file ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Bifchmod is intended to mimic chmod(1).

A BIF file name is recognized by the embedded colon (:) delimiter (see *bif(5)* for BIF file naming conventions).

The permissions of each named file are changed according to *mode*, which may 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*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, 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 will be 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** and **t** only works with **u**.

EXAMPLES

The first example denies write permission to others, and the second makes a file executable (using symbolic mode):

```
bifchmod o-w file
bifchmod +x file
```

The next example below assigns read and execute permission to everybody, and sets the set-user-id bit. The second assigns read and write permission to the file owner, and read permission to everybody else (using absolute mode):

```
bifchmod 4555 file
bifchmod 644 file
```

The following two examples perform the same function, namely to 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/mfd.0:

```
bifchmod a=rx,u+w /dev/mfd.0:/etc/script
bifchmod 755 /dev/mfd.0:/etc/script
```

SEE ALSO

bif(5), chmod(1), chmod(2).

NAME

bifchown, bifchgrp - change file owner or group

SYNOPSIS

bifchown owner device:file ...

bifchgrp group device:file ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

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(5)* for BIF file naming conventions).

Bifchown changes the owner of the *files* to *owner*. The owner may 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*. The group may 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/rfd.

The first example sets the owner of the BIF file /users/abc/phone.num to adm:

```
bifchown adm /dev/rfd:/users/abc/phone.num
```

The second example sets the group ID of the BIF file /tmp/b.date to the decimal number 105:

```
bifchgrp 105 /dev/rfd:/tmp/b.date
```

FILES

/etc/passwd
/etc/group

SEE ALSO

bif(5), chown(1), chgrp(1), group(5), passwd(5).

NAME

bifcp - copy to or from BIF files

SYNOPSIS

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

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: HP

DESCRIPTION

Bifcp is intended to mimic cp(1).

A BIF file name is recognized by the embedded colon (:) delimiter (see *bif(5)* 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.

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

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

EXAMPLES

bifcp abc /dev/hd.c:x/y/z
copy the HP–UX file *abc* to the BIF file *x/y/z* within HP–UX device */dev/hd.c*

bifcp /dev/fd.0:/backup/log logcopy
copy BIF file */backup/log* within */dev/fd.0* to HP–UX file *logcopy* within the current directory.

bifcp /dev/bb:archive –
copy BIF file *archive* within HP–UX device */dev/bb* to standard output.

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

```
sdfcp /dev/hd1:/a /dev/hd1:/b /dev/hd1:/c /users/dave
```

The last example shows how you can implement a "cat" program for concatenating 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
```

SEE ALSO

bif(5), cp(1).

DIAGNOSTICS

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

BUGS

The '–' (stdio) notation may not work in some situations.

NAME

bifdf - report number of free disk blocks

SYNOPSIS

bifdf [**-t**] [**-f**] [file–systems]

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: HP

DESCRIPTION

Bifdf prints out the number of free blocks and free i–nodes available for on–line Bell file systems by examining the counts kept in the super–blocks; *file–systems* may be specified by device name (e.g. /dev/rhd).

The **-t** flag causes the total allocated block figures to be reported as well.

If the **-f** flag is given, only an actual count of the blocks in the free list is made (free i–nodes are not reported). With this option, *bifdf* will report on raw devices.

HARDWARE DEPENDENCIES

Series 500:

Bifdf can only report on unmounted raw devices.

SEE ALSO

bif(5), biffsck(1), df(1).

NAME

biffind - find files in a BIF system

SYNOPSIS

biffind path–name–list expression

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: HP

DESCRIPTION

Biffind is intended to mimic find(1).

A BIF file name is recognized by the embedded colon (:) delimiter (see *bif(5)* 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 may 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 (a.k.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* 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 *find* command line anytime you want *find* 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 which satisfy the other primaries in the *expression*.

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

EXAMPLES

To print the names of all files on the BIF volume */dev/archive1:*

```
biffind /dev/archive1: -print
```

The following command finds all files in */dev/old:/usr/lib* which are directories:

```
biffind /dev/old:/usr/lib -type d -print
```

Finally,

```
find /dev/games:/users -type d -exec bifls -l {} \;
```

gives a long listing of every directory under */users* on the device */dev/games.*

FILES

/etc/passwd
/etc/group

SEE ALSO

bif(5), find(1).

NAME

biffsck - Bell file system consistency check and interactive repair

SYNOPSIS

biffsck [**-y**] [**-n**] [**-sX**] [**-SX**] [**-t**filename] [file-system] ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

Biffsck audits and interactively repairs inconsistent conditions in a Bell file system. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *biffsck* will default to the **-n** option described below.

The following flags are interpreted by *biffsck*.

-y Assume a yes response to all questions asked by *biffsck*.

-n Assume a no response to all questions asked by *biffsck*; and do not open the file system for writing.

-s*X* Ignore the actual free list and unconditionally reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done.

The **-s***X* option allows for creating an optimal free-list organization. The following forms of *X* are supported for the following devices:

-sBlocks-per-cylinder:Blocks-to-skip

If *X* is not given, the values used when the file system was created are used. If these values were not specified, then the default values shown below are used:

An HP 7908A uses 35:2;
An HP 7933A uses 23:15;
An HP 7911A uses 16:12;
An HP 7912A uses 16:12;
An HP 7914A uses 16:12;
The default for *biffsck*(1) is 400:9;
The default for *bifmkfs*(1) is 500:3.

-S*X* Conditionally reconstruct the free list. This option is like **-s***X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **-S** will force a **no** response to all questions asked by *biffsck*. This option is useful for forcing free list reorganization on uncontaminated Bell file systems.

-t If *biffsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, *biffsck* will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked. If the file does not exist, *biffsck* will create it. If the scratch file is not a special file, it is removed when *biffsck* completes.

File-system is a device file name on which the file system to be checked resides (i.e. /dev/rhd).

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.

2. Blocks claimed by an i–node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:

   Incorrect number of blocks.
   Directory size not 16–byte aligned.
5. Bad i–node format.
6. Blocks not accounted for anywhere.
7. Directory checks:

   File pointing to unallocated i–node.
   I–node number out of range.
8. Super Block checks:

   More than 65536 i–nodes.
   More blocks for i–nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i–node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **/lost+found** directory on the bif volume. The name assigned is the i–node number. The only restriction is that the directory **lost+found** must pre–exist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory (optimally in multiples of 64), and then removing them before *biffsck* is executed.

Biffsck can check file systems on both raw and blocked devices. Checking raw devices is almost always faster, but should not be used on a mounted file system.

SEE ALSO

bif(5).

DIAGNOSTICS

The diagnostics produced by *biffsck* are intended to be self–explanatory.

WARNING

It is recommended that the system administrator have total responsibility for running *biffsck*.

BUGS

Inode numbers for **.** and **..** in each directory should be checked for validity.

NAME

biffsdb - Bell file system debugger

SYNOPSIS

biffsdb special [-]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

Biffsdb can be used to patch up a damaged Bell file system after a crash/failure. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the Bell file system tree.

Biffsdb contains several error checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *biffsdb* with the optional - argument or by the use of the **O** symbol. (*Biffsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Biffsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *biffsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,-	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i–nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i–node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs and spaces may be used as function delimiters but are not necessary. A line with just a new–line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i–node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i–number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for i–node examination and refer to the current working i–node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
s0	high byte of file size
s1	low word of file size
a#	data block numbers (0 - 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i–number 386 in an i–node format. This now becomes the current working i–node.
ln=4	changes the link count for the working i–node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i–node.
2i.fd	prints the first 32 directory entries for the root i–node of this file system.
d5i.fc	changes the current i–node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first 512 bytes of the file are then printed in ASCII.
1b.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i–number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one com–

mand line.

d7.nm="name" changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.

SEE ALSO

bif(5), biffsck(1).

WARNING

The use of *biffsdb* should be limited to experienced *biffsdb* users.

NAME

bifls - list contents of BIF directories

SYNOPSIS

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

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Bifls is intended to mimic ls(1).

A BIF file name is recognized by the embedded colon (:) delimiter (see *bif(5)* 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.

If you are the super-user, *bifls* defaults to listing all files except **.** (current directory) and **..** (parent directory). If invoked by the name *bifll,* the -l option is implied.

There are several options to 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 the super-user, 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 an BIF directory structure exists on the HP-UX device file **/dev/fd.0**.

The first example will list all the files in the root directory of the BIF directory structure:

bifls -a /dev/fd.0:

The second example gives (in long format) all the information about the BIF directory **/users/root** itself (but not the files in the directory):

bifls -ld /dev/fd.0:/users/root

FILES

/etc/passwd to get user ids.
/etc/group to get group ids.

NOTE

Remember, to obtain a listing of the BIF files on /dev/fd, you must not say **bifls /dev/fd** but you must include the colon, as in **bifls /dev/fd:**. If the colon is omitted, you get a listing of the HP-UX file /dev/fd, not its BIF contents.

SEE ALSO

bif(5), ls(1).

NAME

bifmkdir - make a bif directory

SYNOPSIS

bifmkdir device:dirname ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Bifmkdir is intended to mimic mkdir(1).

A BIF file name is recognized by the embedded colon (:) delimiter (see *bif(5)* for BIF file naming conventions).

Bifmkdir creates specified directories in mode 777. Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

EXAMPLES

Create an empty subdirectory named sysmods under the directory /usr/lib on HP-UX device /dev/bif2:

```
bifmkdir /dev/bif2:/usr/lib/sysmods
```

SEE ALSO

bif(5), mkdir(1).

DIAGNOSTICS

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

NAME

bifmkfs - construct a Bell file system

SYNOPSIS

bifmkfs special blocks[:inodes] [gap blocks]
bifmkfs special proto [gap blocks]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Bifmkfs constructs a Bell file system by writing on the special file according to the directions found in the remainder of the command line. If the second argument is given as a string of digits, *bifmkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of blocks divided by 4.

If the second argument is a file name that can be opened, *bifmkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the i-list size in blocks The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **-bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or **-** to specify set-user-id mode or not. The third is **g** or **-** for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions (see *bifchmod*(1)).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a path name whence the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers. If the file is a directory, *bifmkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *blocks* can be specified. For RP04 type drives, these numbers should be 7 and 418.

EXAMPLES

To put a Bell file system on a double density micro floppy with 770 1K blocks of capacity:

```
bifmkfs /dev/rfd9122.0 770
```

where /dev/rfd9122.0 is the device special file for the micro floppy.

SEE ALSO

bif(5).

BUGS

If a prototype is used, it is not possible to initialize a file with second– or third–level indirects.

NAME

bifrm, bifrmdir - remove BIF files or directories

SYNOPSIS

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

bifrmdir device:dir ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

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(5)* 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).

The options are:

-f removes a file with no questions asked, even if the file has no write permission.

-r causes *bifrm* to recursively delete the entire contents of a directory, and 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 an BIF directory structure exists on the HP-UX device file **/dev/bifdisc**.

The first example recursively combs through the BIF directory **/tmp** and asks if each BIF file should be removed (forced, with no file mode checks):

bifrm -irf /dev/bifdisc:/tmp

The second example removes the BIF directory **/users/doug**:

bifrmdir /dev/bifdisc:/users/doug

SEE ALSO

bif(5), rm(1), rmdir(1).

NAME

bs - a compiler/interpreter for modest–sized programs

SYNOPSIS

bs [file [args]]

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

Bs is a remote descendant of Basic and Snobol4 with a little C language thrown in. *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 the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from *file* are compiled for later execution. Likewise, statements entered from the console 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 inner–most *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* option is present on the command line, that 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 it's 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 may not 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*, 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* allows for a 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

The 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 may not be nested.

obase *N*

Obase sets the output base to *N* (see *ibase* above).

onintr label
onintr

The *onintr* command provides program control of interrupts. In the first form, control will pass 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 will cause *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 will be 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. It is used to interject commentary in a program.

Expression Syntax:

name

A name is used to specify a variable. Names are composed of a letter (upper or lower case) 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. They return zero otherwise. Relational operators at the same level extend as follows: $a>b>c$ is the same as $a>b$ & $b>c$. 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.

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 similar to the regular expression syntax of the *ed*(1) command. The characters ., [,], ^ (inside brackets), * and $ are special. The *mstring* function 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)
The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may 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*, the *name* reverts to being an ordinary variable. If *name* was a pipe, a *wait*(2) is executed before the close completes. 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*(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)
The *iskey* function 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)

The *plot* function produces output on devices recognized by *tplot*(1G). The *requests* are as follows:

Call	*Function*
plot(0, term)	causes further *plot* output to be piped into *tplot*(1G) with an argument of **-T***term*.
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*(1G). See *plot*(4) for more details.

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.

Programming Tips:

Using *bs* as a calculator:

```
$ 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 "oldfile" to "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")
```

SEE ALSO

ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), plot(5).
See Section 3 of the Reference Manual for a further description of the mathematical functions (*pow* on *exp*(3M) is used for exponentiation); *bs* uses the Standard Input/Output package.

VARIABLES

EDITOR
: the editor to use for the **edit** command.

BUGS

Bs is not extremely tolerant of some errors. Mistyping a *fun* declaration is painful, as a new definition cannot be be made without doing a *clear*. Starting using the *edit* command is the best solution in this case.

HARDWARE DEPENDENCIES

The graphics mode is nearly useless without *tplot*, which is not currently available.

NAME

cal - print calendar

SYNOPSIS

cal [[month] year]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

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. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

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.

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in the login directory and sends them any positive results by *mail*(1). Normally this is done daily in the early morning hours under control of *cron*(1M).

FILES

calendar

/usr/lib/calprog to figure out today's and tomorrow's dates

/etc/passwd

/tmp/cal*

/usr/lib/crontab

SEE ALSO

mail(1), cron(1M).

BUGS

Your calendar must be public information for you to get reminder service.
Calendar's extended idea of "tomorrow" does not account for holidays.

NAME

cat - concatenate, copy, and print files

SYNOPSIS

cat [**-u**] [**-s**] [**-v** [**-t**] [**-e**]] file ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8-bit file names, 8-bit and 16-bit data, customs, messages

DESCRIPTION

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

```
cat file
```

prints the file, and:

```
cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument **-** is encountered, *cat* reads from the standard input file, enabling you to combine standard input with other files.

The options are:

-u causes output to be unbuffered (character-by-character); normally, output is buffered.

-s makes *cat* silent about non-existent files, identical input and output, and write errors. Normally, no input file may be the same as the output file unless it is a special file. (The 4.2BSD **cat** **-s** feature is provided by *ssp*(1).)

-v causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. Control characters are printed ^*X* (control-*X*); the DEL character (octal 0177) is printed ^**?**. Non-ASCII characters (with the high bit set) are printed as **M-***x*, where *x* is the character specified by the seven low order bits.

-t when used with the **-v** option, **-t** causes tabs to be printed as ^**I**'s.

-e when used with the **-v** option, causes a **$** character to be printed at the end of each line (before the new-line).

The **-t** and **-e** options are ignored if the **-v** option is not specified.

SEE ALSO

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

WARNING

Command formats such as

```
cat file1 file2 >file1
```

overwrite the data in *file1* before the concatenation begins. Therefore, take care when using shell special characters.

NAME

cb - C program beautifier, formatter

SYNOPSIS

cb [**-s**] [**-j**] [**-l** leng] [file ...]

HP-UX COMPATIBILITY

Level: HP-UX/EXTENDED

Origin: System V

DESCRIPTION

Cb reads C programs either from its arguments 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. Under the **-s** flag *cb* canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The **-j** flag causes split lines to be put back together. The **-l** flag causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).

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

BUGS

Punctuation that is hidden in preprocessor statements will cause indentation errors.

NAME

cc – C compiler

SYNOPSIS

cc [options] files

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: HP

INTERNATIONAL SUPPORT

16- and 8-bit characters in strings and comments

DESCRIPTION

Cc is the HP-UX C compiler. It accepts several types of arguments:

Arguments whose names end with **.c** are taken to be C source programs. They are compiled, and each object program is left on the file whose name is that of the source with **.o** substituted for **.c**. However, if a single C program is compiled and linked all in one step, the **.o** file is deleted.

Similarly, arguments whose names end with **.s** are taken to be assembly source programs and are assembled, producing a **.o** file.

Arguments whose names end with **.o** are taken to be relocatable object files which are to be included in the link operation.

Arguments can be passed to the compiler through the **CCOPTS** environment variable as well as on the command line. The compiler picks up the value of **CCOPTS** and places its contents before any arguments on the command line. For example (in *sh*(1) notation),

```
CCOPTS=–v
export CCOPTS
cc –g prog.c
```

is equivalent to

```
cc –v –g prog.c
```

The following options are recognized by *cc*.

–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*(1) on the named C or assembly programs, and send the result to the standard output.

–g Cause the compiler to generate additional information needed by the symbolic debugger.

–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* Cause the linker to search the library **lib***x***.a**. 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.

–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*(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

–P Run only *cpp*(1) on the named C programs and leave the result on corresponding files suffixed **.i**.

–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. The use of this option will prevent the use of a symbolic debugger on the resulting program. See *ld*(1) for more details.

–S Compile the named C programs, and leave the assembly language output on corresponding files suffixed **.s**.

–t *c,name* Substitute or insert subprocess *c* with *name* where *c* is one or more of a set of identifiers indicating the subprocess(es). This option 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.

c can take one or more of the values:

p	preprocessor (standard suffix is *cpp*)
c	compiler body (standard suffix is *ccom*)
0	same as *c*
a	assembler (standard suffix is *as*)
2	optimizer (standard suffix is *c2*)
l	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, producing a step-by-step description of the compilation process on *stderr.* Also echoes **CCOPTS** if it is set.

–w Suppress warning messages.

–W *c,arg1[,arg2...]*

Hand off the argument[s] *argi* to pass *c* where *c* 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, on the Series 300,

```
–W d,–x
```

causes the driver to call various subprocesses needed to generate MC68020 code. Furthermore, a shorthand notation for this mechanism can be used by placing "+" in front of the option name as in

+x

which is equivalent to the previous option example. Some commonly used subprocess options can also be abbreviated in a similar fashion. Note that for simplicity, this shorthand must be applied to each option individually. Options that can be abbreviated using "+" are implementation-dependent, and are listed under HARDWARE DEPENDENCIES.

−Y Enable support of 16-bit characters inside string literals and comments. Note that 8-bit parsing is always supported. See *HPNLS*(7) for more details on Native Language Support.

−z Do not bind anything to address zero. This option will allow runtime detection of null pointers. See the note on *pointers* below .

−Z Allow dereferencing of null pointers. See the note on *pointers* below.

Any other options encountered will generate a warning to *stderr*.

Other arguments are taken 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 Kernighan and Ritchie C text, and the various addenda to it, comprise the best available reference on C. The documents are intentionally ambiguous in some areas. HP-UX specifies some of these below.

char
The **char** type is treated as signed by default. It may 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 importability 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. The header file *model.h* (see *model*(5)) contains a number of standard definitions for objects of a constant width, independent of the individual implementation. 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.

HARDWARE DEPENDENCIES

Series 200/300:

The following options are not supported: **−w** **−z**

The default is to allow null pointer dereferencing, hence using **−Z** has no effect.

The default is to generate code for the processor on the machine where the compilation is taking place. For example, on a Series 300 with a MC68020 processor, the compiler will generate MC68020 code.

The compiler driver supports the following cross-compilation options which may also be passed to it from *cc* using the **-W d** option.

+x causes the compiler to generate inline code for the MC68020 and MC68881. This option may also be passed to the driver as -W d,-x .

+X causes the compiler to generate "generic" MC68010 code. The code will also run on MC68020 processors, but it will not take advantage of new architectural capabilities. This option may also be passed to the driver as -W d,-X .

The compiler subprocess *ccom* supports the following options which may be passed to it from *cc* using the **-W c** option. Some of these can be passed directly to the driver using the "+" notation.

+b *or* **-W c,–b**
causes the compiler to generate code for floating point operations that will use floating point hardware if it is installed in the computer at run-time. This option cannot be used when code is being generated explicitly for the MC68020, either by default on a MC68020 based system or via the +x option.

+f *or* **-W c,–f**
causes the compiler to generate code for floating point operations that will use floating point hardware. This code will not run unless floating point hardware is installed. This option cannot be used when code is being generated explicitly for the MC68020, either by default on a MC68020 based system or via the +x option.

+M *or* **-W c,–M**
causes the compiler to not generate in-line code for the MC68881 floating-point coprocessor. Library routines will be referenced for *matherr* capability. Meaningless on MC68010 based systems or in conjunction with +X.

+N< *secondary*>< *n*> *or* **-W c,-N**< *secondary*>< *n*>
This option adjusts the size of internal compiler tables. The compiler uses fixed size arrays for certain internal tables. *Secondary* is one of the letters from the set {**abdepstw**}, and *n* is an integer value. *Secondary* and *n* are **not** optional. The table sizes can be re-specified using one of the secondary letters and the number *n* as follows:

- **a** maximum size of the asciz table Default = 10000 table entries.
- **b** maximum size of the bc table. This table saves break and continue labels within a switch statement. Default = 100 table entries.
- **d** max size of the dimtab table. This table maintains information about the definitions of all structures, unions, and arrays. Default = 1000 table entries.
- **e** max number of nodes per statement. Default = 350 table entries.
- **p** max size of the parameter stack. Default = 150 table entries.
- **s** max size of the symbol table. Default = 1000 table entries.
- **t** max size of the tasciz table. Default = 20000 table entries.
- **w** max size of the switch table stack. Default = 250 table entries.

–W c,–YE
This option causes 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.

Series 500:

The following options are not supported: **–p –w**

The default is not to allow null pointer dereferencing, hence using **–z** has no effect.

The file /lib/mcrt0.o is not currently supported.

The compiler subprocess *ccom* supports the following options which may be passed to it from *cc* using the **−W c** option. Some of these can be passed directly to the driver using the "+" notation.

+N< *secondary*><*n*> *or* **-W c,-N**< *secondary*><*n*>

This option adjusts the size of internal compiler tables. The compiler uses fixed size arrays for certain internal tables. *Secondary* is one of the letters from the set {**bpwgi**}, and *n* is an integer value. *Secondary* and *n* are **not** optional. The table sizes can be respecified using one of the secondary letters and the number *n* as follows:

b maximum size of the bc table. This table saves break and continue labels within a switch statement. Default = 100 table entries.

p max size of the parameter stack. Default = 150 table entries.

g max size of the argument stack. Default = 100 table entries.

w max size of the switch table. Default = 250 table entries.

i max size of the instruction table for generated code. Default = 300 table entries.

EXAMPLE

The following will compile the C program *prog.c*, creating a *prog.o* file, and will then invoke the link editor *ld*(1) 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.a*; the resulting executable program is output in *prog*:

```
cc prog.c procedure.o -o prog
```

FILES

file.c	input file
file.o	object file
a.out	linked output
/tmp/ctm*	temporary
/usr/tmp/ctm*	temporary
/lib/cpp	preprocessor
/lib/ccom	compiler, *cc*
/lib/c2	optional optimizer (for Series 200 and Series 500 only)
/bin/as	assembler, *as*(1)
/bin/ld	link editor, *ld*(1)
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/libc.a	standard C library, see section LIB of this manual
/usr/include	standard directory for **#include** files
Series 200/300:	
/lib/ccom10	compiler, MC68010 version (linked to */lib/ccom* on MC68010 systems).
/lib/ccom20	compiler, MC68020 version (linked to */lib/ccom* on MC68020 systems).
/lib/c210	optimizer, MC68010 version (linked to */lib/c2* on MC68010 systems).

/lib/c220	optimizer, MC68020 version (linked to */lib/c2* on MC68020 systems).
/bin/as10	assembler, MC68010 version (linked to */bin/as* on MC68010 systems).
/bin/as20	assembler, MC68020 version (linked to */bin/as* on MC68020 systems).

SEE ALSO

adb(1), cdb(1), cpp(1), as(1), ld(1), prof(1), exit(2), monitor(3C), matherr(3M), model(5).
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978.

NOTES

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*(2) or to leave the function **main()** with a '**return** *expression*;' construct.

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor.

WARNINGS

Options not recognized by *cc* are not passed on to the link editor. The option **-W l,***arg* may be used to pass any such option to the link editor.

NAME

cd - change working directory

SYNOPSIS

cd [directory]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

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*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and is internal to the shell.

VARIABLES

HOME default working directory

CDPATH directories to search for *directory*.

SEE ALSO

pwd(1), sh(1), chdir(2).

NAME

cdb, fdb, pdb - C, FORTRAN, Pascal symbolic debugger

SYNOPSIS

cdb [**-d** *dir*] [**-r** *file*] [**-p** *file*] [**-S** *num*] [*objectfile* [*corefile*]]

fdb [*cdb* options]

pdb [*cdb* options]

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: HP

Remarks: This debugger is currently implemented on the Series 200 and the Series 500.

DESCRIPTION

Cdb, *fdb*, and *pdb* are alternate names for a source level debugger for C that provides a controlled execution environment for HP FORTRAN, and HP Pascal programs.

Objectfile is an executable program file having one or more of its component modules compiled with the **-g** option. The support module */usr/lib/end.o* must be included as the last object file in the list of those linked, except for libraries included with the **-l** option to *ld*(1). (Some systems automate this; see the *Hardware Dependencies* section below.) The default for *objectfile* is **a.out**.

Corefile is a core image from a failed execution of *objectfile*. The default *corefile* is **core** (Series 500 does not support corefiles).

The options are:

-d *dir* names an alternate directory where source files are located.

-r *file* names a *record* file which is invoked immediately (for overwrite, not for append). Used with *Record/Playback Commands*.

-p *file* names a *playback* file which is invoked immediately. Used with *Record/Playback Commands*.

-S *num* sets string cache size to *num* bytes (default *num* varies with symbol table format; not available for all formats). String cache holds data read from *objectfile*.

Only one *objectfile* and one *corefile* is allowed per debugging session. The program (*objectfile*) is not invoked as a child process until an appropriate *Job Control Command* command is given. The same program can be restarted many times (as different child processes) during a single debugging session.

COMMANDS

The debugger has a various commands for viewing and manipulating the program being debugged.

File Viewing Commands

May change current viewing position, but do not affect the next statement (if any) to be executed in the child process.

dir *"directory"*

Add *directory* to the list of alternate source directories. Same as using –**d** invocation option. Main procedure file must reside in the current directory or be specified with the –**d** option.

e Show current file, procedure, line number, and source line

e (*file* | *proc*)

Enter (view) *file* or *proc* and print its first executable line. *File* can be any file, but must not be object code.

[*depth*] **E**
: Like "**e**", but sets viewing location to the current location in *proc* on the stack at depth *depth* (not necessarily first executable line in the procedure). Default *Depth* is zero (where program is currently stopped).

L
: Synonym for **0E**.

line
: Print source line number *line* in current file.

[*line*] **p** [*count*]
: Print one (or *count*) lines starting at current line (or line number *line*). If multiple lines are printed, current line is marked with "=" in leftmost column.

+[*lines*]
: Move to *lines* (default one) lines after current line.

-[*lines*]
: Move to *lines* (default one) lines before current line.

[*line*] **w** [*size*]
: Print window of text containing *size* (default 11) lines centered around current line (or *line*). Target line is marked with "=" in leftmost column if more than one line is printed.

[*line*] **W** [*size*]
: Same as "**w**", but *size* defaults to 21 lines.

+**w** [*size*]
+**W** [*size*]
: Print window of text of given or default *size*, beginning at end of previous window if the previous command was a window command; otherwise at current line.

-**w** [*size*]
-**W** [*size*]
: Print window of text of given or default *size*, ending at beginning of previous window if previous command was a window command; otherwise at 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 previous "/" or "?" command using same *string* as before.

N
: Same as "**n**", but search goes in opposite direction from that specified by previous "/" or "?" command.

Display Formats

A *format* is of form "[*][*count*]*formchar*[*size*]". Display formats apply only to Data Viewing Commands, described in the next sub-section.

"*" means "use alternate address map" (if maps are supported).

Count is the number of times to apply the format style *formchar* (must be a *number*).

Size is number of bytes to be formatted for each *count* (overrides default *size* for the format style); must be positive decimal *number* (except short hand notations). *Size* is disallowed with *formchar*s where it makes no sense.

Uppercase characters can be used with formats that print numbers to obtain same results as appending "**l**" (useful on systems where **integer** is shorter than **long**). Available formats include:

n
: Print in "normal" format, based on type. **char** arrays 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**).
(b | B)	Print a byte in decimal (either way).
(c | C)	Print a character (either way).
(e | E)	Print in "e" floating point notation (as **float** or **double**) (see *printf*(3)) (floating point constants are always doubles).
(f | F)	Print in "f" floating point notation (as **float** or **double**).
(g | G)	Print in "g" floating point notation (as **float** or **double**).
a	Print a string using *expr* as the address of the first byte.
s	Print a string using *expr* as the address of a pointer to the first byte (same as "******expr*/**a**", except for arrays).
t	Show type of *expr* (usually a variable or procedure name). For true procedure types you must actually call the procedure, e.g. "def (2)/t".
p	Print the name of the procedure containing address *expr*.
S	Do a formatted dump of a structure. *expr* must be address of a structure, not address of pointer to a structure.

Shorthand notations for *size* can be appended to *formchar* instead of a numeric *size*:

b	1 byte (**char**).
s	2 bytes (**short**).
l	4 bytes (**long**).

If you view an object with *size* (explicitly or implicitly) less than or equal to size of a **long**, debugger changes basetype appropriately for that *size* so "**.**" (*dot*) can work correctly for assignments (may reduce accuracy or produce wrong value).

Data Viewing Commands

expr If *expr* does not resemble anything else (such as a command), it is handled as "*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 address of *expr* using *format*.

^ [[/]*format*]
Back up to preceding memory location (based on the size of last thing displayed). Uses *format* if supplied, or the previous *format* if not. No "/" is needed after "^". To reverse direction again (e.g. start going forward), enter "." or *dot* (always an alias for current location) followed by carriage return.

l [*proc*[.*depth*]]
List all parameters and local variables for current procedure (or *proc*, if given, at the specified *depth*, if any). Datadisplay uses "/**n**" format, except arrays and pointers are shown as addresses; only the first word of a structure is shown.

l (**a** | **b** | **d** | **z**)
List all **a**ssertions, **b**reakpoints, **d**irectories, or **z**ignals.

l (f | g | l | p | r | s) [*string*]
List all **f**iles (source files which built *objectfile*), **g**lobal variables, **l**abels (program entry points known to the linker), **p**rocedure names, **r**egisters, or **s**pecial variables (except registers). If *string* is present, only those things with the same initial characters are listed.

Stack Viewing Commands

[*depth*] **t**
Trace stack for the first *depth* (default 20) levels.

[*depth*] **T**
Same as "**t**", but local variables are also displayed using "**/n**" format (except that arrays and pointers are shown as addresses; structures show first word only).

Job Control Commands

Parent (debugger) and child (*objectfile*) processes take turns running. Debugger is active only while child process is stopped due to a signal (reaching a breakpoint, or terminated for whatever reason).

r [*arguments*]
Run a new child process with given argument list, if any (existing child process, if any, is terminated first). If no *arguments* are given, those used with last "**r**" command are used again (none if "**R**" was used last).

Arguments may contain "<" and ">" for redirecting standard input and standard output. ("<" does *open*(2) on file descriptor 0 for read-only; ">" does *creat*(2) on file descriptor 1 with mode 0666). *Arguments* may contain shell variables, metacharacters, quote marks, or other special syntax (expanded by a Bourne shell). "**{}**" are shell metacharacters, so "**r**" cannot be safely saved in a breakpoint or assertion command list.

R Run new child process with no argument list.

k Terminate (kill) current child process, if any.

[*count*] **c** [*line*]
Continue after breakpoint or signal, ignoring the signal, if any. If *count* is given, 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 *count* of -1 (see *Breakpoint Commands*).

[*count*] **C** [*line*]
Continue like "**c**", but allow signal (if any) to be received.

[*count*] **s** Single step 1 (or *count*) statements (successive carriage-returns repeat with *count* of 1). If *count* less than one, child process is not stepped. Child process continues with the current signal, if any (set "$signal = 0" to prevent).

[*count*] **S**
Single step like "**s**", but treat procedure calls as single statements (don't follow them down). If a breakpoint is hit in such a procedure, or in one that it calls, its *commands* are executed. (usually all right unless there is a "**c**" command in that breakpoint's command list).

Debugger has no knowledge about or control over child processes forked in turn by the process being debugged. Process being debugged should not execute a different program via *exec*(2).

Child process output may be buffered, so it may not appear immediately after you step through an output statement such as *printf*(3). It may not appear at all if you kill the process.

Breakpoint Commands

A breakpoint has three associated attributes:

address Commands to set breakpoints are alternate ways to specify the breakpoint address. The breakpoint is encountered whenever *address* is about to be executed, regardless of the

path taken to get there. Only one breakpoint at a time (of any type or count) can be set at a given *address*. Setting a new breakpoint at *address* replaces the old one, if any.

count The number of times the breakpoint is encountered prior to recognition. If *count* is positive, the breakpoint is "permanent", and *count* decrements with each encounter. Each time *count* goes to zero, the breakpoint is recognized and *count* is reset to one (so it stays there until explicitly set to a different value by "**c**" or "**C**").

If *count* is negative, the breakpoint is "temporary" and *count* increments with each encounter. When *count* goes to zero, the breakpoint is recognized then deleted.

commands

These are actions to be taken upon recognition of a breakpoint before waiting for command input. Separated by "**;**"; may be enclosed in "**{}**" to delimit the list, saved with the breakpoint, from other commands on the same line.

Results of expressions followed by "**;**" or "**}**" are not printed unless you specify print format.

Saved commands are not parsed until breakpoint is recognized. If *commands* are nil at recognition of breakpoint, debugger waits for command input.

Breakpoint commands:

l b
B list all breakpoints in the format "*num*: count: *nnn proc*: *ln*: *contents*", followed by "{*commands*}".

Leftmost number is an index number for use with "**d**" (delete) command.

[*line*] **b** [*commands*]
Set permanent breakpoint at current line (or at *line* in the current procedure). For immediate continuation, finish the command list with "**c**".

[*expr*] **d**
Delete breakpoint number *expr*. If *expr* is absent, delete the breakpoint at the current line, if any. If there is none, the debugger executes a "**B**" command instead.

bp [*commands*]
Set permanent breakpoints at the beginning (first executable line) of every debuggable procedure. When any procedure breakpoint is hit, *commands* are executed.

D [**b**] Delete all breakpoints (including "procedure" breakpoints). "**b**" is optional.

D p Delete all "procedure" breakpoints. All breakpoints set by commands other than "**bp**" remain set.

For the following commands, if the second character is upper case (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 breakpoint at beginning (first executable line) of procedure at specified stack *depth*. If *depth* not specified, current procedure is used (may not be same as stack *depth* zero).

[*depth*] **bx** [*commands*]
[*depth*] **bX** [*commands*]
Set a breakpoint at exit (last executable line) of procedure at the given stack *depth*. If *depth* is not specified, current procedure is used (may not be same as stack *depth* zero. The breakpoint is set such that all returns of any kind go through it.

[*depth*] **bu** [*commands*]
[*depth*] **bU** [*commands*]
Set an up-level breakpoint. Breakpoint is set immediately after return to the procedure

at specified stack *depth* (default one, not zero). Zero *depth* means "current location".

[*depth*] **bt** [*proc*] [*commands*]
[*depth*] **bT** [*proc*] [*commands*]
Trace current procedure (or procedure at *depth*, or *proc*). Sets breakpoints at entrance and exit of a procedure. Default entry breakpoint *commands* are "**Q;2t;c**", (shows the top two procedures on the stack and continues). The exit breakpoint executes "**Q;L;c**" (prints the current location and continues).

If *depth* is given, *proc* must be absent or it is taken as part of *commands*. If *depth* is missing but *proc* is specified, the named procedure is traced. If both *depth* and *proc* are omitted, the current procedure is traced, which might not be the same as the one at *depth* zero.

If *commands* are present, they are used for the entrance breakpoint, instead of the default shown above.

address **ba** [*commands*]
address **bA** [*commands*]
Set breakpoint at given code address. *address* can be the name of a procedure or an expression containing such a name. If the child process is stopped in a non-debuggable procedure, or in prologue code (before the first executable line of a procedure), results may seem a little strange.

The next few commands, are not strictly part of the breakpoint group, but are used almost exclusively as 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. All other "**{}**" blocks are always ignored (skipped), except when given as an argument to an "**a**", "**b**", or "**!**" command. The "**if**" command is nestable, and can be abbreviated to "**i**".

Q If the "quiet" command appears as the first command in a breakpoint's command list, the normal announcement of "*proc: line: text*" is not made. This allows quiet checks of variables, etc. to be made without cluttering up the screen with unwanted output. The "**Q**" command is ignored if it appears anywhere else.

"*any string you like*"
Print the given string. Accepts standard backslashed character escapes, including "**\n**" for newline. Useful for labelling output from breakpoint commands.

Assertion Control Commands

Assertions are command lists that are executed before every statement. Thus, if there is even one active assertion, the program is single stepped at the machine instruction level (runs very slowly). They are primarily used for tracking down nasty bugs (such as a corrupt global variable).

Assertions can be activated or suspended individually (*cdb* also supports overall assertions mode).

a *commands*
Create new assertion with given command list. List is parsed at execution time. Command list can be enclosed in "**{}**" to delimit it from other commands on the same line. "**l a**" command lists all current assertions and the overall mode.

expr **a** (**a** | **d** | **s**)
Modify the assertion numbered *expr*: **a**ctivate it, **d**elete it, or **s**uspend it. Suspended assertions continue to exist, but do nothing until reactivated.

A Toggle overall state of assertions mechanism between *active* and *suspended.*

D a Delete all assertions.

[*flag*] **x** Force exit from assertions mode. If *flag* is absent or evaluates to zero, exit immediately. Otherwise, finish executing current assertion first. If an assertion executes an "**x**" command, the child process stops and the assertion doing the "**x**" is identified.

Debugger has only one active command line at a time. When assertion command begins execution, any remaining debugger command line is lost.

Signal Control Commands

Debugger catches all signals bound for a child process before the child process sees them (a function of *ptrace*(2) mechanism).

[*signal*] **z** [**i**][**r**][**s**][**Q**]
Maintains the "zignal" (signal) handling table. *Signal* is a valid signal number (default is current signal). Options (which must be all one word) toggle the state of the appropriate flag: **i**gnore, **r**eport, or **s**top. If **Q** is present, the new signal state is not printed.

"**1 z**" lists current handling of all signals (**z** with no options shows the state of the current or selected signal.

When a child process stops or terminates on a signal it is always reported, unless the breakpoint signal command starts with "**Q**".

When debugger ignores a signal, "**c**" does not know about it. The signal is never ignored when a child process terminates; only when it stops.

Record and Playback Commands

Debugger supports a record/playback feature to help recreate program states and record all debugger output.

Commands are:

>*file* Set or change recordfile to *file* and turn recording on. Rewrites *file* from the start. Only commands are recorded to this file.

>>*file* Same as >*file* but appends to *file* instead of overwriting.

>@*file*
>>@*file*
Set or change record–all file to *file*, for overwriting or appending. Record–all file can be opened or closed, independent of recordfile. All debugger standard output is copied to the record–all file, including prompts, commands entered, and command output (does not capture child process output.

>(**t** | **f** | **c**)
Turn recording on (**t**) or off (**f**), or close the recording file (**c**). When recording is resumed, new commands are appended to previous file contents. In this context, >> is equivalent to >.

>@(**t** | **f** | **c**)
Turn record–all on, off, or close the record–all file. In this context, >>@ is the same as >@.

> Tell current recording status (same as >>).

>@ Tell current record–all status. (same as >>@).

<*file* Start playback from *file*.

<<*file* Start playback from *file*, using single–step feature of playback.

Only command lines read from the keyboard or a playback file are recorded in the recordfile.

Command lines beginning with ">", "<", or "**!**" are not copied to current recordfile (they are copied to record–all file). To override this, begin such lines with blanks.

NOTE: Debugger can be invoked with standard input, standard output, and/or standard error redirected, independent of record and playback. If debugger encounters end–of–file while standard input is redirected from anything other than a terminal, it prints a message to standard output and exits, returning zero.

Miscellaneous Commands

<carriage–return>
An empty line or a " " command causes the debugger to repeat the last command, if possible, with an appropriate increment, if any. Repeatable commands are those which print a line, print a window of lines, print a data value, single step, and single step over procedures. <carriage–return> is saved in *record* file as a " " command, to distinguish from ^**D**.

^**D** Control–D is like <carriage–return>, but repeats the previous command ten times. Note that this command is saved in a *record* file as an empty line.

! [*command–line*]
This shell escape invokes a shell program. If *command–line* is present, it is executed via *system*(3). Otherwise, the environment variable SHELL gives the name of the shell program to invoke with a **-i** option, also using *system*(3). If SHELL is not found, the debugger executes "/bin/sh -i". In any case, the debugger then waits for the shell or *command–line* to complete.

As with breakpoints, *command–line* can be enclosed in "{}" to delimit it from other (debugger) commands on the same line.

f ["*printf–style–format*"]
Set address printing format, using *printf*(3) format specifications (**not** debugger format styles). Only the first 19 characters are used. If there is no argument, the format is set to a system–dependent default. All addresses are assumed to be of type **long**, so you should handle all four bytes to get something meaningful.

F Find and fix bug (a useless but humorous command).

g *line* Go to an address in the procedure on the stack at *depth* zero (not necessarily the current procedure). Changes the program counter so *line* is the next line to be executed.

h
help Print debugger help file (command summary) using *more*(1).

I Print information (inquire) about the state of the debugger.

M Print current text (*objectfile*) and core (*corefile*) address maps.

M (**t** | **c**) [*expr*; [*expr*;...]]
Set **t**ext (*objectfile*) or **c**ore (*corefile*) address map. The first zero to six map values are set to the *expr*s given.

q Quit the debugger. Requests confirmation to prevent losing a valuable environment.

Z Toggle case sensitivity in searches. This affects everything: file names, procedure names, variables, and string searches! The debugger starts out as **not** case sensitive.

HARDWARE DEPENDENCIES

The "**bx**" (break on exit) command requires that compilers support it by funneling all exits through one point. The breakpoint is always set at the last line of the procedure, which should be, but may not be, the sole exit point.

Series 200 and Series 500:

When a C parameter is declared as an array of anything, the highest type qualifier (array) shows up as a pointer instead. For example, "int x[]" looks like "int *x", and "char (*x)[]" looks like "char **x", but "char *x[]" is treated correctly as "pointer to array of **char**".

There is limited support for command-line calls of functions which return structures. The debugger interprets the start of heap as a structure of the return type. However, a call such as "abc()/t" displays the return type correctly.

$short and **$long** are available in addition to **$result**. However, **$result** is only set to (valid as) the return value from the last procedure called from the command line. If the procedure returns a **double**, **$result** is set to the value cast to **long**.

The source file *end.c* is not supported, so you can't customize */usr/lib/end.o*. The buffer size is fixed at 200 bytes. To force linking of library routines not otherwise referenced, use **-u** option to *ld*(1).

All compiler front ends (*cc*(1), *fc*(1), and *pc*(1)) automatically tell the linker to include */usr/lib/end.o* for you if you give the **-g** (debug) option (compiler front end cannot detect debug options when they are placed in source code instead).

Series 200 only:

Series 200 supports two types of string formats in addition to null-terminated C strings. FORTRAN *character* variables consist of a string of bytes (no null terminator). Pascal *string* variables consist of a length byte, followed by the string characters. The "\s" and "\a" formats will display these types correctly, only if the current language is FORTRAN or Pascal.

Series 500 only:

"**bx**" works, except for FORTRAN multiple returns. The compilers emit a special source line symbol for this exit point, after the last "visible" source line.

Series 500 supports two types of string formats in addition to null-terminated C strings. FORTRAN **character** variables consist of four-word (16-byte) string markers, where the second word plus the third word plus three is the byte address of the string itself, and the fourth word is the length of the string. Pascal **string** variables consist of a four-byte, word-aligned length word followed by the string characters.

If the current language is FORTRAN, or if you use "**/s**" format with **fdb** or **pdb**, the debugger interprets the variable (or expression) as a string marker (or address thereof), which is a null pointer if the second word of the marker is zero. Multiple-count formats show a series of fixed-length strings, beginning with the first one pointed to by the marker. Using "<cr>" or "^" to go forward or backward in memory uses the four words after or before the current string marker as the new marker.

If the current language is Pascal, or if you use "**/a**" format with **fdb** or **pdb**, the debugger interprets the variable (or expression) as a Pascal **string** (or address thereof). Multiple-count formats show a series of random-length strings, using successive length words, skipping any wasted bytes in the last word of the previous string. Likewise, using "<cr>" or "^" to go through memory skips the total bytes consumed in the last display.

There is never a *corefile*, so all features which depend on it don't work. Also, there are no address maps in the usual sense, so the "**M**" command is not supported.

If a child process receives a signal and you then step with the "**s**" command (or run with assertions active), the process free-runs through the signal handler procedure (if any) before pausing (or doing assertions).

Code and data pointers in *objectfile* both contain segment numbers. At *exec*(2) time, all such pointers are mapped from *ld*(1) pseudo-values to real values based on actual segment numbers allocated. The debugger operates in "pseudo-address-space", so you won't notice anything unusual most of the time. All addresses look the same each time you invoke a new child process. For example, the heap always begins at "broken" address zero (0).

WARNING: The debugger's interaction with a child process is somewhat complicated, due to the "fixing" of pointer values written to the child and the "breaking" of pointers read from the child. If you tell the debugger to treat a pointer as a non-pointer, it may get confused, with unpredictable results. In particular, if you set a debugger special variable equal to a pointer value, then attempt to dereference that special variable, you will either get garbage or cause an access error.

In the rare case where maxheap is set very large (greater than 70Mb) and your program uses shared EMS segments (from *memallc*(2)), the debugger may confuse pointers into the EMS segments with large addresses in the heap.

Addresses of unknown (non-debuggable) procedures are shown as call-type pointers, not data pointers. They can be distinguished because the high bit is set (e.g., the decimal value looks negative). Pointers of this form are not usable for anything; you can't dereference them nor set breakpoints based on them.

SYMBOL TABLE DEPENDENCIES

Series 200 and Series 500 compilers use the HP9000 Symbol Table Format.

HP9000 Symbol Table Format:

Procedures in FORTRAN and Pascal may have alias names in addition to normal names. Aliases are shown by the "**l p**" (list procedures) command. They can be used in place of the normal name, as desired.

The procedure name "_MAIN_" is used as the alias name for the main program (main procedure) in all supported languages. Do not use it for any debuggable procedures.

FORTRAN ENTRY points are flagged "ENTRY" by the "**l p**" command.

When a compiler does not know array dimensions, such as for some C and FORTRAN array parameters, it uses 0:MAXINT or 1:MAXINT, as appropriate. The "**/t**" format shows such cases with "[]" (no bounds specified), and subscripts from 0 (or 1) to MAXINT are allowed in expressions.

Even though the symbol table supports C structure, union, and enumeration tags, C typedefs, and Pascal types, the debugger does not know how to search for them, even for the "**/t**" format. They are "invisible".

Some variables are indirect, so a child process must exist in order for the debugger to know their addresses. When there is no child process, the address of any such variable is shown as 0xfffffffe.

The optional pattern given with the "**l g**" (list globals) command must be an exact match, not just a leading pattern.

The string cache (see the **-S** option) defaults to 1Kbyte in size. This cache holds data read from the Value Table.

Symbol names in the Value Table are never preceded by underscores, so the debugger never bothers to search for names of that form. The only time a prefixed underscore is expected is when searching the Linker Symbol Table for names of non-debuggable procedures.

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.errors
: Text file which explains debugger error and warning messages.

/usr/lib/end.o
Object file to link with all debuggable programs.

SEE ALSO

cc(1), echo(1), ld(1), more(1), creat(2), exec(2), fork(2), open(2), printf(3), system(3), a.out(5), and the *cdb Debugger* tutorial in *HP-UX Concepts and Tutorials.*

On some systems any of the following may exist: adb(1), fc(1), pc(1), ptrace(2), core(5), symtab(5), user(5).

DIAGNOSTICS

Most errors cause a reasonably accurate message to be given. Normal debugger exits return zero and error exits return one. All debugger output goes to standard output except error messages given just before non-zero exits, which go to standard error.

Debugger errors are preceded by "panic: ", while user errors are not. If any error occurs during initialization, the debugger then prints "cannot continue" and quits. If any error happens after initialization, the debugger attempts to reset itself to an idle state, waiting for command input. If any error occurs while executing a procedure call from the command line, the context is reset to that of the normal program.

Child process (program) errors result in signals which are communicated to the debugger via the *ptrace*(2) mechanism. If a program error occurs while executing a procedure call from the command line, it is handled like any other error (i.e. you can investigate the called procedure). To recover from this, or to abort a procedure call from the command line, type DEL, BREAK, ^C, or whatever your interrupt character is.

For more information, see the text file */usr/lib/cdb.errors.*

WARNINGS

Code that is not 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 not debuggable. The main procedure (main program) must be debuggable and have a corresponding source file.

If the *address* given to a "**ba**" command is a not a code address in the child process, strange results or errors may ensue.

If you set the address printing format to something *printf*(3) doesn't 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. If you change its state you had better know what you are doing or be a very good sport!

If you single step or run with assertions through a call to *longjmp*(3), 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 playback from a file currently open for recording, or vice versa; strange things can happen.

Many compilers only issue source line symbols at the end of each logical statement or physical line, *whichever is greater.* This means that, if you are in the habit of saying "a = 0; b = 1;" on one line, there is no way to put a breakpoint after the assignment to "a" but before the assignment to "b".

Some statements do not emit code where you would expect it. For example, assume:

```
 99:     for (i = 0; i < 9; i++) {
100:             xyz (i);
101:     }
```

A breakpoint placed on line 99 will be hit only once in some cases. The code for incrementing is placed at line 101. Each compiler is a little different; you must get used to what your particular compiler does. A good way of finding out is to use single stepping to see in what order the source lines are executed.

The output of some program generators, such as *yacc*(1), 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. Removal of line directives fixes the problem, but makes it more difficult to find error locations in the original source file. The following script, run after *yacc*(1) and before *cc*(1), comments out line number changes in C programs:

```
sed "/# *line/s/^.*$/\/*&*\//" y.tab.c >temp.c
```

yacc(1) will leave out line directives if invoked with the **-l** option. In general, line number directives (or compiler options) are only safe so long as they never set the number backwards.

BUGS

The C operators "**++**", "**--**", and "**?:**" are not available. The debugger always understands all the other C operators, except "**sizeof**", if the default language is FORTRAN or Pascal.

For FORTRAN, only the additional operators "**.NE.**", "**.EQ.**", "**.LT.**", "**.LE.**", "**.GT.**", and "**.GE.**" are supported.

For Pascal, only the operators "**:=**", "**<>**", "**^**", "**^.**" (as in "x^.y"), "**and**", "**or**", "**not**", "**div**", "**mod**", "**addr**", and "**sizeof**" are added.

There is no support for FORTRAN **complex** variables, except as a series of two separate **float**s or **double**s.

The debugger doesn't understand C type casts.

The C operators "**&&**" and "**||**" aren't short circuit evaluated as in the compiler. All parts of expressions involving them are evaluated, with any side-effects, even if it's not necessary.

The debugger doesn't understand C pointer arithmetic. "*(a+n)" is not the same as "a[n]" unless "a" has an element size of 1.

There is no support for C local variables declared in nested blocks, nor for any local overriding a parameter with the same name. When looking up a local by name, parameters come first, then locals in the order of the "}"s of the blocks in which they are declared. When listing all locals, they are shown in the same order. When there is a name overlap, the address or data shown is that of the first variable with that name.

CDB does not support identically-named procedures (legal in Pascal if the procedures are in different scopes). CDB will always use the first procedure with the given name.

Pascal WITH statements are not understood. To access any variable you must specify the complete "path" to it.

The debugger supports call-by-reference only for known parameters of known (debuggable) procedures. If the object to pass lives in the child process, you can fake such a call by passing "& object", i.e. the address of the object.

Array parameters are always passed to command-line procedure calls by address. This is correct except for Pascal call-by-value parameters. Structure parameters are passed by address or value, as appropriate, but only a maximum of eight bytes is passed, which can totally confuse the called procedure. Series 500 FORTRAN string markers are never passed correctly. Only the first number of a complex pair is passed as a parameter. Functions which return complex numbers are are not called correctly; insufficient stack space is allocated for the return area, which can lead to

overwriting the parameter values.

Assignments into objects greater than four bytes in size, from debugger special variables, result in errors or invalid results.

Command lines longer than 1024 bytes are broken into pieces of that size. This may be relevant if you run the debugger with playback or with input redirected from a file.

NAME

cdc - change the delta commentary of an SCCS delta

SYNOPSIS

cdc -rSID [**-m**[mrlist]] [**-y**[comment]] files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Cdc changes the *delta commentary*, for the *SID* specified by the **-r** keyletter, 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** keyletters).

If a directory is named, *cdc* 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 (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to *cdc*, which may appear in any order, consist of *keyletter* arguments and file names.

All the described *keyletter* 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** flag set (see *admin*(1)) then a list of **MR** numbers to be added and/or deleted in the delta commentary of the *SID* specified by the **-r** keyletter *may* be supplied. A null **MR** list has no effect.

MR entries are added to the list of **MR**s in the same manner as that of *delta*(1). In order 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 **MR**s, it is removed and changed into a "comment" line. A list of all deleted **MR**s 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** keyletter).

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 taken to be the name of a program (or shell procedure) which 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*(s) already existing for the delta specified by the **-r** keyletter. The 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 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.

The exact permissions necessary to modify the SCCS file are documented in the article indicated under SEE ALSO. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

cdc -r1.6 -m"bl78–12345 !bl77–54321 bl79–00001" -ytrouble s.file

adds bl78–12345 and bl79–00001 to the **MR** list, removes bl77–54321 from the **MR** list, and adds the comment **trouble** to delta 1.6 of s.file.

cdc -r1.6 s.file
MRs? !bl77–54321 bl78–12345 bl79–00001
comments? trouble

does the same thing.

FILES

x–file (see *delta*(1))
z–file (see *delta*(1))

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).
SCCS User's Guide in *HP–UX Concepts and Tutorials.*

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (**-** on the command line), then the **-m** and **-y** keyletters must also be used.

NAME

cflow- generate C flow graph

SYNOPSIS

cflow [**-r**] [**-ix**] [**-i_**] [**-d**num] files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

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 YACC'd, LEX'd, and C-preprocessed (bypassed for **.i** files) as appropriate and then run through the first pass of *lint*(1). (The **-Y** option of *cc*(1) and the **-I**, **-D**, and **-U** options of the C-preprocessor 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. Then 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 (e.g., **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 may 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 also sorted in lexicographical order by callee.

-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 nonpositive integer will be met with contempt.

HARDWARE DEPENDENCIES

Series 200:

The size of the internal compiler tables, used by the first pass of *lint*(1), can be adjusted by using the **–N** option. The syntax for this option is described in the HARDWARE DEPENDENCIES section for Series 200 in the manual page for *cc*(1).

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C–preprocessor).

SEE ALSO

as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

chatr - change program's internal attributes

SYNOPSIS

chatr [**-n**] [**-q**] [**-s**] files

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: This manual page describes *chatr* as implemented on Series 200 computers. Refer to other *chatr*(1) manual pages for information valid for other implementations.

DESCRIPTION

Chatr, by default, prints each *file*'s magic number and file attributes to the standard output. With one or more optional arguments, *chatr* performs the following operations:

-n change the file from demand loaded to shared.

-q change the file from shared to demand loaded.

-s perform action silently.

Upon completion, *chatr* prints the file's old and new values to the standard output file, unless **-s** is in effect.

RETURN VALUE

Chatr returns zero on success. If the call to *chatr* 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.

SEE ALSO

ld(1), a.out(5), magic(5).

DIAGNOSTICS

The error messages produced by *chatr* should be self-explanatory.

NAME

chatr – change program's internal attributes

SYNOPSIS

/lbin/chatr [**+c**|**-c**] [**+g**|**-g**] [**+h**|**-h**] [**-m***n*] [**+n**|**-n**] [**+p**|**-p**] [**-q**|**-Q**] [**-s**] [**+z**|**-z**] file ...

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: This manual page describes *chatr* as implemented on Series 500 computers. Refer to other *chatr*(1) manual pages for information valid for other implementations.

DESCRIPTION

Chatr, by default, prints each *file*'s magic number and file attributes to the standard output. With one or more optional arguments, *chatr* performs the following operations:

c set (+) or clear (-) the virtual bit for each code segment.

g set (+) or clear (-) the virtual bit of the global data segment.

h set (+) or clear (-) the virtual bit for the heap of a two data segment program.

-m*n* change the maximum heap size to *n* bytes.

n mark code as shareable (+) (magic number = SHARE_MAGIC), or unshareable (-) (magic number = EXEC_MAGIC).

p set (+) or clear (-) the paged and virtual bits for the heap of a two data segment program.

-q set the demand load bit for each segment.

-Q clear the demand load bit for each segment.

-s perform action silently.

-w*n* change the maximum working set size to *n* bytes.

z set (+) or clear (-) the demand load bit for each segment.

Upon completion, *chatr* prints the file's old and new values to the standard output file, unless **-s** is in effect.

RETURN VALUE

Chatr returns zero on success. If the call to *chatr* 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.

SEE ALSO

ld(1), a.out(5), magic(5).

DIAGNOSTICS

Chatr generates an error message for the following conditions:

no arguments are supplied – in this case the syntax is printed to the standard error file;

cannot open a file;

a request is made to modify a file which is not EXEC_MAGIC or SHARE_MAGIC;

working set size is larger than heap size.

Chatr generates a warning message for the following conditions:

the **+p**, **−p**, **+h**, or **−h** option is specified for a file which is a one data segment program;

the **−m** or **−w** option is specified for a file which is a one data segment program, or a file for which the data is unpaged.

NAME

chmod - change mode

SYNOPSIS

chmod mode file ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8-bit filenames.

DESCRIPTION

The permissions of the named *file*s are changed according to *mode*, which may 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*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, 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 will be 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** and **t** only works with **u**.

Only the owner of a file (or the super-user) may change its mode. Only the super-user may set the sticky bit. In order to set the group ID, the group of the file must correspond to your current group ID.

EXAMPLES

The first example denies write permission to others, and the second makes a file executable (using symbolic mode):

```
chmod o-w file
chmod +x file
```

The first example below assigns read and execute permission to everybody, and sets the set-user-id bit. The second assigns read and write permission to the file owner, and read permission to everybody else (using absolute mode):

```
chmod 4555 file
chmod 644 file
```

SEE ALSO

ls(1), chmod(2).

NAME

chown, chgrp - change file owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8-bit filenames.

DESCRIPTION

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

Chgrp changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

In order to change the owner or group, you must own the file or be the super-user. If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), chown(2), group(5), passwd(5).

NAME

chsh - change default login shell

SYNOPSIS

chsh name [shell]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

Chsh is a command similar to *passwd*(1), except that it is used to change the login shell field of the password file rather than the password entry. If no *shell* is specified then the shell reverts to the default login shell */bin/sh*.

An example use of this command is:

```
chsh bill /bin/csh
chsh bill /bin/sh
chsh john /bin/PAM
```

SEE ALSO

csh(1), passwd(1), passwd(5).

NAME

clear - clear terminal screen

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It reads the **TERM** environment variable for the terminal type and then reads the appropriate terminfo data base to figure out how to clear the screen.

FILES

/usr/lib/terminfo/?/* terminal capability files

SEE ALSO

terminfo(5).

NAME

cmp - compare two files

SYNOPSIS

cmp [**-l**] [**-s**] file1 file2

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8-bit and 16-bit data, customs, messages.

DESCRIPTION

The two files are compared. (If *file1* 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.

Options:

-l Print the byte number (decimal) and the differing bytes (octal) for each difference. (Byte numbering begins at 1, rather than at 0 as is common.)

-s Print nothing for differing files; return codes only.

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

col - filter reverse line–feeds and backspaces

SYNOPSIS

col [**-bflpx**]

HP–UX COMPATABILITY

Level: HP–UX/STANDARD

Origin: System V

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**). It also removes backspaces in favor of multiply overstruck lines. *Col* is particularly useful for filtering multicolumn output made with the **.rt** command of *nroff*(1) and output resulting from use of the *tbl*(1) preprocessor.

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 will be 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 minimun 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* will convert 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**, **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* will ignore any unrecognized escape sequences found in its input; the **-p** option may 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.

Note that the input format accepted by *col* matches the output produced by *nroff*(1) with either the **-T37** or **-Tlp** options. Use **-T37** (and the **-f** option of *col*) if the ultimate disposition of the output of *col* will be a device that can interpret half–line motions, and **-Tlp** otherwise.

SEE ALSO

nroff(1), tbl(1).

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* will be a device that can interpret half–line motions, and **-Tlp** otherwise.

BUGS

Cannot back up more than 128 lines.
Allows at most 800 characters, including backspaces, on a line.
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.

NAME

comm - select or reject lines common to two sorted files

SYNOPSIS

comm [**-** [**123**]] file1 file2

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8-bit data, customs, messages.

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name **-** means the standard input.

Flags 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** is a no-op.

SEE ALSO

cmp(1), diff(1), sdiff(1), sort(1), uniq(1).

NAME

compact, uncompact, ccat - compress and uncompress files, and cat them

SYNOPSIS

compact [name ...]
uncompact [name ...]
ccat [file ...]

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: UCB

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted 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 prepend a decoding tree to the compressed file since 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 and the resulting file is placed in *file.C; file* is unlinked. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

Uncompact restores the original file from a file compressed by *compact.* If no file names are given, the standard input is uncompacted to the standard output.

Ccat cats the original file from a file compressed by *compact,* without uncompressing the file.

RESTRICTION

The last segment of the filename must contain fewer than thirteen characters to allow space for the appended '.C'.

FILES

*.C compacted file created by compact, removed by uncompact

SEE ALSO

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

Colin L. Mc Master

NAME

cp, ln, mv - copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target
ln [**-f**] [**-s**] file1 [file2 ...] target
mv [**-f**] file1 [file2 ...] target

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8-bit file names, customs, messages

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same (take care when using *sh*(1) metacharacters). If *target* is a directory, then one or more files are copied (linked, moved) to that directory. If two or more files are specified for any of these commands (not counting *target*), then *target* must be a directory. If *target* is a file, its contents are destroyed.

If *mv* or *ln* determines that the mode of *target* forbids writing, it will ask permission to overwrite the file. This is done by printing the mode (see *chmod*(2)) followed by the first letters for the words *yes* and *no* in the current native language, asking for a response, and reading the standard input for one line. If the line (which you type in) begins with the first of the choices displayed, the operation occurs, if permissable; if not, the command exits. No questions are asked and the *mv* or *ln* is done when the **-f** option is used or if the standard input is not a terminal.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file. When using *cp*, if *target* is not a file, a new file is created which has the same mode as *file1* except that the sticky bit is not set unless you are super-user; the owner and group of *target* are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, nor group. The last modification time of *target* (and last access time, if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

You cannot use *mv* to perform the following operations:

rename either the current working directory or its parent directory using the "." or ".." notation;

rename a directory such that its new name is the same as the name of a file contained in that directory.

SEE ALSO

cpio(1), link(1M), rm(1), chmod(2).

BUGS

If *file1* and *target* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner becomes that of the copying process and any linking relationship with other files is lost.

Ln cannot not create hard links across file systems.

You cannot use *mv* to rename a directory when its name ends in a slash (/).

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o [**acBvxh**]

cpio -i [**BcdmPrtuvfsSbx6**] [patterns]

cpio -p [**adlmuvx**] directory

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8-bit filenames.

DESCRIPTION

Cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary.

Cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1). In *patterns*, meta-characters **?**, *****, and [...] match the slash **/** character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is ***** (i.e., 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 will be those of the previous **cpio -o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous **cpio -o**.

Cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below. Destination path names are interpreted relative to the named *directory*.

The meanings of the available options are:

a Reset access times of input files after they have been copied.

B Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option); meaningful only with data directed to or from devices which support variable length records such as magnetic tape.

d *Directories* are to be created as needed.

c Write *header* information in ASCII character form for portability.

r Interactively *rename* files. 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 will not replace a newer file with the same name).

x Save or restore device special files. *Mknod(2)* will be used to recreate these files on a restore, and thus -ix can only be used by the super-user. Restoring device files onto a different system can be very dangerous. This is intended for intrasystem (backup) use.

v *Verbose*: causes 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 is ineffective on 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 will probably need further processing to be readable; this processing often requires knowledge of the content of the file 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 where 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 halfwords of the file. Use only with the **-i** option.

b Swap both bytes and halfwords. Use only with the **-i** option.

6 Process an old (i.e., UNIX System *Sixth* Edition format) file. Only useful with **-i** (copy in).

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* will prompt the user for a new special file and continue.

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 (e.g. /dev/tty03) will not transport to other implementations of HP–UX.

HARDWARE DEPENDENCIES

General

The use of *cpio* with cartrige tape units requires additional comments. For an explanation of the constraints on cartrige tapes, see *ct*(4).

Warning: 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). *Tcio*(1) buffers data into larger pieces suitable for cartrige tapes.

The **-B** option *must* be used when writing directly (i.e. without using *tcio*(1)) to a CS–80 cartridge tape unit.

Series 500:

All files with i–nodes greater than or equal to 65535 are unlinkable with the **-i** option. A separate copy of each file is made instead.

The number of blocks reported by *cpio* is always in units of 512–byte blocks, regardless of the block size of the initialized media.

Note that the **-B** option must *not* be used when performing raw I/O to the internal miniature flexible disc drive (HP 9130K), if the I/O requires more than one volume.

At and before release 4.0 on the 500 and 2.2 on the 200, these systems wrote a format which, when crossing media boundaries on some kinds of discs, differs from the format specified by System V.2 (although it matched that written by System III). The program */etc/ocpio* will read and write this format. */etc/ocpio* has essentially the same features as cpio except that options **-S**, **-b** and –**f** are omitted. */etc/ocpio* is considered obsolescent.

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/mt/0m

cd olddir
find . -depth -print | cpio -pdl newdir
```

The trivial case "find . -depth -print | cpio -oB >/dev/rmt/0m" can be handled more efficiently by:

```
find . -cpio /dev/rmt/0m
```

SEE ALSO

ar(1), find(1), tar(1), tcio(1), cpio(5).

DIAGNOSTICS

The diagnostic message *"out of phase"* indicates that *cpio* could not successfully read its particular "magic number" in the header. Try changing header mode (**c option**) or byte swapping the header (**P** or **s** options).

WARNING

Do not redirect the output of *cpio* to a named *cpio* archive file which resides in the same directory as the original files which are part of that *cpio* archive. This can cause loss of data.

BUGS

Path names are restricted to 256 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super–user can copy special files.

Cpio tapes written on HP machines with the **-ox[c]** options can mislead (non–HP) versions of *cpio* which do not support the **-x** option. If a non–HP (and non–Bell) version of *cpio* happens to be modified so that (HP) *cpio* recognizes it as a device special file, a spurious device file could be created.

If /dev/tty is not accessible, *cpio* issues a complaint, or refuses to work.

The **-pd** option will not create the directory typed on the command line.

The **-idr** option will not make empty directories.

The **-plu** option will not link files to existing files.

Cpio will fail while restoring files from a backup tape (**cpio -i**) if the following conditions are met:

> your working directory during the restore is **not** the root directory (/), **and** the files being restored have multiple links, **and** their path names begin with slash (/).

If these conditions are met, the following occurs:

(1) The first file on the backup tape is restored correctly;
(2) The second file is removed, and the restore fails.

Note that the second file is removed before the restore fails!

Cpio then writes the message "Cannot link *file1* & *file2*" to *stderr*, but also writes *"file1* linked to *file2"* on *stdout*, as if everything went fine. The correct message is that written to *stderr*.

There are two work–arounds for this bug, either of which will solve the problem. The first is to make sure that your working directory is the root directory during the restore process. The second is to use relative file names (path names not beginning with slash) in your backup.

NAME

cpp - C language preprocessor

SYNOPSIS

/lib/cpp [option ...] [ifile [ofile]]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. 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* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc*(1) 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 supplied.

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:	mert, ibm, gcos, os, tss, unix
hardware:	hp9000s500, hp9000s200, interdata, pdp11, u370, u3b, u3b5, vax
UNIX system variant:	RES, RT, TS, PWB, hpux
lint(1):	lint

All HP-UX systems will have the symbols PWB, hpux, and unix defined. Each system will define exactly one hardware variant, as appropriate. The lint symbol will be defined when *lint*(1) is running.

-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. That is, if the same name is used in both a **-U** option and a **-D** option, the name will be undefined regardless of the order of the options.

-T On HP-UX, preprocessor symbols are no longer restricted 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 " " will be 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, the 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 proportionally. The default table size is 36000 bytes with 2000 symbols. This option serves to eliminate the "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 known 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* may 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 will then be 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, 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 will 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 will 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 will 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 utility of **#ifdef** and **#ifndef** in a **#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 will 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.

FILES

/usr/include standard directory for **#include** files

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

The 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.

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.

NAME

crontab – user crontab file

SYNOPSIS

crontab [file]
crontab –r
crontab –l

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The -r option removes a user's crontab from the crontab directory. *Crontab* -l will list the crontab file for the invoking user.

A user is permitted to use *crontab* if their name appears 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 either file is **at.deny**, 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 may 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 minus sign (meaning an inclusive range). Note that the specification of days may 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 would run 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 would run 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 **arg0** 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:/usr/lbin)**.

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

sh(1), cron(1M).

NAME

csh - a shell (command interpreter) with C–like syntax

SYNOPSIS

csh [**-cefinstvVxX**] [command file] [argument list ...]

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: UCB

DESCRIPTION

Csh is a command language interpreter incorporating a command history buffer and a C–like syntax.

The command options are interpreted as follows:

-c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

-e The 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 Forces *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, no matter which options are selected.

-n This option causes commands to be parsed, but **not** executed. This may be used in syntactic checking of shell scripts. All substitutions are performed (history, command, alias, etc.).

-s Command input is taken from the standard input.

-t A single line of input is read and executed. This option combines the **-n** option described above with automatic execution of the command.

-v This option causes the *verbose* shell variable to be set. This causes command input to be echoed to your standard output device after history substitutions are made.

-x This option causes the *echo* shell variable to be set. This causes all commands to be echoed to the standard output immediately before execution.

-V This option causes the *verbose* variable to be set before *.cshrc* is executed. This means all *.cshrc* commands are also echoed to the standard output.

-X This option causes the *echo* variable to be set before *.cshrc* is executed. This means 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 is made the input of the next command in the pipeline. Sequences of pipelines may be separated by semicolons (;), and are then executed sequentially. A sequence of pipelines may be executed in background mode by following the last entry with an ampersand (&) character.

Any pipeline may be placed in parenthesis to form a simple command which in turn may be a component of another pipeline. It is also possible to separate pipelines with "||" or "&&" indicating, as in the C language, that the second pipeline is to be executed only if the first fails or succeeds, respectively.

Built–In Commands

Built–in commands are executed within the shell. If a built–in command occurs as any component of a pipeline except the last then 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 final form assigns the specified *wordlist* as the alias of *name*. Command and filename substitution are performed on *wordlist*. *Name* cannot be **alias** or **unalias**.

alloc This comand shows the amount of dynamic core in use, broken down into used and free core, and the address of the last location in the heap. With an argument, *alloc* shows each used and free block on the internal dynamic memory chain indicating its address, size, and whether it is used or free. This is a debugging command.

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 no argument is given, then *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 "../"), then 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 '/', then 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 description of the *foreach*, *if*, *switch*, and *while* statements below.

eval arguments ...
(As in *sh*(1).) The 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*)
The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expression* (second form).

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* may be used to continue the loop prematurely and the built–in command *break* terminates 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 you can then rub it out.

glob *wordlist*
Like *echo* but no '\' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to perform filename expan–sion on a list of words.

goto *word*
The specified *word* is filename 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 *exec*'s). An *exec* is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/'.

history
history *n*
history -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.

if (*expression*) *command*
If the specified expression evaluates true, then the single *command* with arguments is exe–cuted. 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 com–mand list, or a parenthesized command list. Input/output redirection occurs even if *expression* is false, when *command* is **not** executed (this is a bug).

if (*expression1*) **then**
...
else if (*expression2*) **then**
...
else
...
endif If the specified *expression1* is true then the commands to the first *else* are executed; otherwise if *expression2* is true then the commands to the second else are executed, etc. Any number of **else**–**if** pairs are possible; 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 the active jobs; the **-l** option lists process id's in addition to the normal 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 saying just **kill** does not send a signal to the current job.

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.

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 super–user may raise the priority by specifying negative niceness using **nice** -*number* .... *Command* is always executed in a sub–shell, and the restrictions place 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.

notify [%*job* ...]
Causes the shell to notify the user asynchronously when the status of the current (no argument) 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 is to terminate shell scripts or to return to the terminal command input level. If **-** is specified, causes all interrupts to be ignored. If

a *label* is given, causes the shell to execute 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.

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, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

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 which have other than a single word as value 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 filename expanded.

These arguments may 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*]
With no argument, 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. With an argument, *shift* performs the same function on the specified *variable*.

source *name*
The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Input during *source* commands is **never**

placed on the history list.

stop [*%job* ...]
Stops the current (no argument) or specified job which is executing in the background.

switch (*string*)
case *str1*:
...
breaksw
...
default:
...
breaksw
endsw Each *case* label (*str1*) is successively matched against the specified *string* which is first command and filename expanded. The file metacharacters *, ?, and [...] may be used in the *case* labels, which are variable expanded. If none of the labels match before a **default** label is found, then the execution begins after the **default** label. Each *case* label and the **default** label must appear at the beginning of a line. The command *breaksw* 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*]
With no argument, a summary of time used by this shell and its children is printed. If an argument is given, the specified 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 (no argument) 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 read permission only 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* is omitted.

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 distasteful side-effects. No error occurs if *pattern* is omitted.

unsetenv *pattern*
Removes all variables whose names match the specified *pattern* from the environment. See also the *setenv* command above and *printenv*(1).

wait All background jobs are waited for. If the shell is interactive, then 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* may 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.

@

@ *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 "|", then 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 may 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 (i.e. @ i++).

Non–Built–In Command Execution

When a command to be executed is not a built–in command, the shell 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.

Parenthesized commands are always executed in a subshell. Thus

(cd ; pwd)

prints the *home* directory but leaves you where you were.

cd ; pwd

does the same thing, but leaves you in the *home* directory.

Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary file, then it is assumed to be a shell script, and a new shell is spawned to read it.

If there is an *alias* for *shell* then 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 shell (e.g. "$shell"). Note that this is a special, late–occurring case of *alias* substitution, which inserts words into the argument list without modification.

Command Substitution

Command substitution is indicated by a command enclosed in single quotes (*` ...`*). 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.

History Substitutions

History substitutions enable you to use words from previous commands as portions of new commands, repeat commands, repeat arguments of a previous command in the current command, and fix spelling mistakes in the previous command.

History substitutions begin with an exclamation point (!). Substitutions may begin anywhere in the input stream, but may **not** be nested. The exclamation point can be preceded by a backslash to prevent 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 right parenthesis. Any input line which contains history substitution is echoed on the terminal before it is executed for verification.

Commands input from the terminal which 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 most previous command.

To select words from a command you can follow the event specification by a colon (:) and a designator for the desired words. The words of a input line are numbered from zero. The basic word designators are:

0 selects the first word (i.e. the command name itself).

n selects the *n*th word.

$ selects the last word.

a–*b* selects the range of words from *a* to *b*. Special cases are **-y**, which is an abbreviation for "word 0 through word *y*", and **x-**, which stands for "word *x* up to, but not including, word **$**".

***** indicates the range from the second word to 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 each word designator, you can place a sequence of modifiers, each preceded by a colon. The following modifiers are defined:

h Use only the first component of a pathname 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 pathname by removing all leading pathname 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 global command; used as a prefix to cause the specified change to be made globally (all words in the command are changed).

Unless preceded by a **g**, the modification is applied only to the first modifiable word. You get an error if a substitution is attempted and cannot be completed (i.e. if you have a history buffer of 10 commands and ask for a substitution of **!11**).

The left hand side of substitutions are not regular expressions in the sense of the HP–UX editors, but rather strings. Any character may be used as the delimiter in place of a slash (/); a backslash quotes the delimiter into the *l* and *r* strings. The character & in the right hand side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in **!?***s*?. The trailing delimiter in the substitution may be omitted if a newline follows immediately, as may the trailing **?** in a contextual scan.

A history reference may be given without an event specification (e.g. **!$**). 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 caret (^). 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 may be surrounded with curly braces { } if necessary to insulate it from the characters which follow. Thus, after

ls -ld ~paul

we might 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 backslash (\) and double quotes (") can be used to prevent all or some of the remaining substitutions. Strings enclosed in backslashes 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 does a double–quoted string yield parts of more than one word; single–quoted strings never do.

Alias Substitution

The shell maintains a list of aliases which 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, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then 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, then 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 we can execute

```
alias print 'pr \!* | lpr'
```

to make a command which uses *pr*(1) to print its arguments on the line printer.

Expressions

A number 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 ==, !=, =~, and !~ operators compare their arguments as strings; all others operate on numbers. The operators =~ and !~ are like != 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 which begin with **0** are considered octal numbers. Null or missing arguments are considered **0**. The result of all expressions are strings, which 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 which 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 filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries 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 then the command should be executed outside of an expression and the variable *status* examined.

Control of the Flow (one)

The shell contains a number of commands which 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 up 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

The shell 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. The shells 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;

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 backslashs or quotes, a newline preceded by a backslash gives a true newline character.

When the shell's input is not a terminal, the pound sign (#) 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 filename 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 filename 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 which 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 following characters which 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, then that value is used. Non–*csh* variables cannot be modified as shown below.

$variable_name[selector]
${variable_name[selector]}

This modification allows you to select only some of the words from the value of *variable_name*. The selector is subjected to variable substitution and may 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. This 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 filename 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** may be applied to the substitutions above, as may **:gh**, **:gt** and **:gr**. If curly braces { } appear in the command form then the modifiers must appear within the braces. *The current implementation allows only one* : *modifier on each* $ *expansion.*

The following substitutions may not be modified with **:** modifiers.

$?variable_name
${?variable_name}

Substitutes the string **1** if *variable_name* is set, **0** if it is not. *Variable_name* must be a boolean variable.

$?0 Substitutes **1** if the current input filename 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, *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. The HP–UX environment variable PATH could be set in the shell script **.login**, except that commands through *net*(1) would not see that value.

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 sub–directories in *chdir* commands.

cwd This variable contains the absolute pathname of your current working directory. Whenever you change 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 your standard output dev–ice just before being executed. Built–in commands are echoed before command and filename substitution, since these substitutions are then done selectively. For non–built–in commands, all expansions occur before echoing.

history This variable is used to create your command history buffer and to set its size. If this variable is not set, you have no command history and can do no history substitutions. Very large values of **history** may run your shell out of memory. Values of 10 or 20 are normal. All commands, executable or not, are saved in your command history buffer.

home This variable contains the absolute pathname to your home directory. **Home** is initialized from the HP–UX environment. The filename expan–sion of tilde (~) refers to this variable.

ignoreeof If set, *csh* ignores end–of–file characters from input devices which are terminals. This prevents your processes from accidentally being killed by control–D's.

mail This variable contains a list of the files where *csh* checks for your mail. *Csh* periodically (default is 10 minutes) checks this variable after a com–mand completion which results in a prompt. If the variable contains a filename 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, then 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 insure that files are not accidentally destroyed, and that commands using append redirection (>>) refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is no longer an error for a filename 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, i.e. **/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 then only full path names can be executed. When *path* is not set and when users do not specify full pathnames, *csh* searches for the command through the directories your 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 pound sign (#) 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 which 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, then 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 which controls the automatic timing of commands. If set, then *csh* prints, for any command which takes more than the specified number of cpu seconds, a line of information to your 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 exection.

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 Filename Substitution

The remaining substitutions, command and filename substitution, are applied selectively to the

arguments of built–in commands. This means that portions of expressions which 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.

Filename Substitution

If a word contains any of the characters *, ?, [, or {, or begins with the character ~, then that word is a candidate for filename substitution, also known as *globbing*. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters *, ?, and [imply pattern matching, while the characters ~ and { are more like abbreviations.

In matching filenames, the character **.** at the beginning of a filename or immediately following a /, as well as the character / itself, must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence [...] matches any one of the characters enclosed. Within the square brackets, a pair of characters separated by - matches any character lexically between and including the two.

The tilde character (~) at the beginning of a filename is used to refer to home directories. By itself, the tilde expands to your home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and - characters, the shell searches for a user with that name and substitutes their home directory; thus **~ken** might expand to **/users/ken** and **~ken/chmach** to **/usr/ken/chmach**. If the ~ is followed by a character other than a letter or /, or appears somewhere other than at the beginning of a word, it is left undisturbed.

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 may be redirected with the following syntax:

< *name*

Open file *name* (which is first variable, command and filename 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, filename 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 then it is created; if the file exists, it is truncated, and its previous contents are lost.
If the variable *noclobber* is set, then 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 filenames are.

>> *name*
>>& *name*
>>! *name*
>>&! *name*

Uses file *name* as standard output like >, but appends output to the end of the file. If the variable *noclobber* is set, then 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. Note that the default standard input for a command run detached is **not** modified to be the empty file */dev/null*; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user is notified.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form "|&" rather than just "|".

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 your ESCAPE key. The system then completes the filename for you, echoing the full name on your terminal. If the abbreviation doesn't match an available filename, 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 may 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 **Control–D** or **Control–F** at this point lists the files available. The files are listed in multicolumn format, sorted column-wise.

Directories and executable files are indicated with 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 **control–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 [Control]–[D]
```

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, then 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.

Sanity

The shell now 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 facility to save your history list between login sessions. If the shell variable *savehist* is set to a number, then that number of command events from your history list are 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.

FILES

/bin/sh	standard shell, for shell scripts not starting with a #;
/tmp/sh*	temporary file for <<;
/etc/passwd	source of home directories for ~name.
/etc/csh.login	a csh script executed when starting a csh login (analogous to */etc/profile* in the Bourne shell).

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 filename expansion is limited to 1/6'th 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* substititutions on a single line to 20.

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), umask(2), wait(2), tty(4), a.out(5), environ(7).

BUGS

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 suspend 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 ()'s 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.

If you do a kill job command on a pipeline, only the first process in the pipeline is killed.

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.

Your terminal type is only examined the first time you attempt recognition.

To list all commands on the system along PATH, enter [SHIFT]–[CNTL]–[D].

The csh metasequence !~ does not work.

NAME

ctags - create a tags file

SYNOPSIS

ctags [**-BFatuwvx**] files ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

Ctags makes a *tags* file for *ex*(1) (or *vi*(1)) from the specified C, Pascal and Fortran source *files*. A *tags* file gives the locations of specified objects (for C, functions, macros with argments, and typedefs; Pascal, procedures, programs and functions; FORTRAN, subroutines, progams 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. 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 blanks or tabs. Using the *tags* file, *ex* can quickly find these object's definitions.

-x causes *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 A page index is produced 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)). Since the output will be sorted into lexicographic order, it may be desired to run the output through **sort -f**. Sample use:

```
ctags -v files | sort -f > index
pr index files
```

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 suppressing warning diagnostics.

-u causing the specified files to be *updated* 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 prepending *M* to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

DIAGNOSTICS

Too many entries to sort.

An attempt to get additional heap space failed; the sort could not be performed.

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.

FILES

tags	output tags file
OTAGS	temporary used by –u

SEE ALSO

ex(1), vi(1).

BUGS

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded 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 will be 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.
It 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(1) 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 will be indexed.

NAME

cu - call another (HP–UX) system; terminal emulator

SYNOPSIS

cu [-sspeed] [-llline] [**-h**] [**-t**] [**-q**] [**-o**|**-e**] [**-d**] [**-m**] [**-n**] **telno | systemname | dir**
telno

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

Native Language Support:
8-bit data.

DESCRIPTION

Cu calls up another HP–UX system, a terminal, or possibly a non–HP–UX system. It manages an interactive conversation with possible transfers of ASCII files.

Cu accepts the following options and arguments:

-s*speed* Specifies the transmission speed (110, 134, 150, 300, 600, 1200, 2400, 3600, 7200, 4800, 9600, 19200); 300 is the default value. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

When using a direct–connect line, the **-s** option has no effect. The first line which matches the **-l** option is used, and its speed is taken from L–devices.

-l*line* Specifies 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 taken from the file **/usr/lib/uucp/L–devices**. When the –**l** and –**s** options are used simultaneously, *cu* will search the L–devices file to check whether the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made.

The specified device is generally a directly connected asynchronous line (e.g., **/dev/ttyab**); in this case a phone number is not required but the string **dir** may be use to specify a null ACU. If the specified device is associated with an auto–dialer, a phone number must be provided.

-h Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half–duplex mode.

-q Invokes the use of ENQ/ACK handshake.

-t Used when dialing an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage–return to carriage–return–line–feed pairs is set.

-d Causes diagnostic traces to be printed.

-e(**-o**) Designates that even (odd) parity is to be generated for data sent to the remote.

-m Designates a direct line which has modem control.

-n Requests the phone number to be dialed from the user rather than taking it from the command line.

telno When using an automatic dialer, this argument is the telephone number, with equal signs inserted to wait for secondary dial tones and minus signs inserted for any other delays.

systemname
A **uucp** system name may be used rather than a phone number; in this case, *cu* will obtain an appropriate direct line or phone number from **/usr/lib/uucp/L.sys** (the appropriate baud rate is also read along with phone numbers). *Cu* will try each phone

number or direct line for **systemname** in the **L.sys** file until either a connection is made or all the entries are tried.

dir Using **dir** insures that *cu* will use the line specified by the **–l** option.

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *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 so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following:

~. and **~..**	terminate the conversation. On a hardwired line (only), **~.** sends several EOF characters to log out the session; **~..** will suppress the EOF sequence. In general the remote hardwired machine will be unaware of the disconnect if **~..** is used. **~.** and **~..** do not differ for dialup connections.
~!	escape to an interactive shell on the local system.
~!*cmd...*	run *cmd* on the local system (via **sh -c**).
~&	just like ~! but kill the receive process, restoring it upon return from the shell. This is useful for invoking sub–processes which read from the communication line (i.e., *kermit*).
~&*cmd...*	run *cmd* on the local system (via **sh -c**) and kill the receive process, restoring it later.
~$*cmd...*	run *cmd* locally and send its output to the remote system.
~%cd	change the directory on the local system. **NOTE: ~!cd will cause the command to be run by a sub–shell, which is probably not what was intended.**
~%take *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%put *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* 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	toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which 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 prompt sequence and timeout are specified using ~%setps and ~%setpt, described below. While an input diversion is in progress, a limited amount of keyboard input is buffered, and not sent to the remote until the diversion ends. The diversion may be stopped prematurely by hitting the BREAK key.
~%setpt *n*	set the prompt timeout value to n seconds. The default is 2 seconds. A value of 0 will disable the timeout.
~%setps *xy*	set the handshake prompt to the characters *xy*. The default is DC1. The notation ^X may be used to specify a control character (i.e., ^J for line feed). The notation \^ may be used to enter the ^ character. Setting the prompt to ^@ (null) will cause any character to satisfy the prompt

sequence.

~%**set** this command will simply display the current value of the prompt timeout and prompt sequence.

~%>*file* Divert output from the remote system to the specified file until another ~%> command is given. When an output diversion is active, typing ~%> will terminate it, and ~%>*anotherfile* will terminate it and begin a new one. The output diversion (surprisingly) remains active through a ~& subshell, but unpredictable results may occur if input/output diversions are intermixed with ~%**take** or ~%**put**.

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 the existence of *echo*(1) and *cat*(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

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, uname can be executed on Z, X, and Y as follows:

uname
Z
X
~!uname
Y

In general, ~ causes the command to be executed on the original machine, and ~~ causes the command to be executed on the next machine in the chain.

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 login to 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:

```
cu YYYZZZ
```

FILES

/usr/lib/uucp/L.sys
/usr/lib/uucp/L-devices

/usr/spool/uucp/LCK..(tty-device)
/dev/null

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uname(1), uucp(1C).

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

BUGS

Cu buffers input internally.

There is an artificial slowing of transmission by *cu* during the ~**%put** operation so that loss of data is unlikely.

NAME

cut - cut out selected fields of each line of a file

SYNOPSIS

cut -clist [file1 file2 ...]
cut -flist [**-d** char] [**-s**] [file1 file2 ...]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (**-c** option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (**-f** option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

list A comma-separated list of integer field numbers (in increasing order), with optional **-** to indicate ranges as in the **-o** option of *nroff/troff* for page ranges; e.g., **1,4,7**; **1-3,8**; **-5,10** (short for **1-5,10**); or **3-** (short for third through last field).

-c*list* The *list* following **-c** (no space) specifies character positions (e.g., **-c1-72** would pass the first 72 characters of each line).

-f*list* The *list* following **-f** is a list of fields assumed to be separated in the file by a delimiter character (see **-d**); e.g., **-f1,7** 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.

-s Suppresses lines with no delimiter characters in case of **-f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the **-c** or **-f** option must be specified.

Hints

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLES

cut -d: -f1,5 /etc/passwd — mapping of user ID to names

name=`who am i | cut -f1 -d" "` — to set **name** to current login name.

DIAGNOSTICS

line too long A line can have no more than 1023 characters or fields.

bad list for c/f option Missing **-c** or **-f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

no fields The *list* is empty.

SEE ALSO

grep(1), paste(1).

BUGS

Cut(1) does not expand tabs; input should be piped through *expand*(1) if tab expansion is required.

NAME

cxref - generate C program cross–reference

SYNOPSIS

cxref [options] files

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

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 **#define**'d 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.

In addition to the **-D**, **-I**, **-U**, and **-Y** options (which are identical to their interpretation by *cc*(1)), the following *options* are interpreted by *cxref*:

-c Print a combined cross–reference of all input files.

-w*num* Width option, which formats 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; does not print input file names.

-t Format listing for 80–column width.

HARDWARE DEPENDENCIES

Series 200:

cxref utilizes a special version of the C compiler front end. The size of the internal compiler tables can be adjusted by using the **–W***c* and **–N** options, as described in the manual page for *cc*(1).

FILES

/usr/lib/xcpp special version of C–preprocessor.

/usr/lib/xpass special version of C compiler front end.

SEE ALSO

cc(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

BUGS

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**. Since *cxref* operates by running special versions of the C preprocessor and the C compiler front end, if a file cannot be compiled, it cannot be processed by *cxref*.

NAME

date – print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8–bit data, customs, messages

DESCRIPTION

If no argument is given, or an argument beginning with + is included, the current date and time are printed. If an argument that is not preceded by + is included **and** you are super-user, the current date and time are set to the value specified by the argument. *yy* is the last two digits of the year number; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (one 24-hour clock cycle per day); the second *mm* is the minute number. The year, month, and day can be omitted, in which case the current values are used as defaults. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM in the current year (HP-UX operates in GMT). *Date* handles conversions to and from local standard and daylight time.

Attempting to set the date backwards generates a warning, and requires an extra confirmation from the (super-)user.

If an argument is present that begins with +, the argument defines the output format from *date*. The argument is assumed to contain field descriptors from the list that follows, each preceded by the character %, and, optionally, other text to be included in the output. Output format is similar to that of the first argument to *printf*(3S). Numeric output fields are of fixed size, and include leading zeros when needed. Each field descriptor and the preceding % are replaced in the output by the corresponding value from the current date and time. All other characters in the format argument are copied to the output without alteration. To produce a % text character in the output string, use %% in the format argument. The output string is always terminated with a newline character.

Date writes an accounting record on the file /usr/adm/wtmp.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year – 01 to 12
d	day of month – 01 to 31
y	last 2 digits of year – 00 to 99
D	date as mm/dd/yy
H	hour – 00 to 23
M	minute – 00 to 59
S	second – 00 to 59
T	time as HH:MM:SS
j	Julian date – 001 to 366
w	day of week – Sunday = 0
a	abbreviated weekday name – Sun to Sat
W	full weekday name – Sunday to Saturday
h	abbreviated month name – Jan to Dec
F	Full month name – January to December
r	time in AM/PM notation
z	time zone name from TZ variable in user's environment

The full or abbreviated month name and full or abbreviated weekday name are spelled according to user's native language as defined by his LANG variable (see *environ*(7)).

If no format argument is present, the current date and time are printed according to the D_T_FMT string (see *langinfo*(3C)) which corresponds to the current value of the variable LANG in the user's environment. If the LANG variable is not set, *ctime*(3C) is used to format the date.

HARDWARE DEPENDENCIES

Series 500:

Do not change the date and/or time in the BASIC language system if your machine also runs HP-UX. The two operating systems' date and time are incompatible.

EXAMPLE

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

generates the output:

```
DATE: 08/01/76
TIME: 14:45:05
```

FILES

/usr/adm/wtmp

SEE ALSO

ctime(3C), langinfo(3C), environ(7).

DIAGNOSTICS

No permission	if you are not super-user and try to change the date;
bad conversion	if the date-setting parameter is syntactically incorrect;
bad format character	if the field descriptor is not valid.

NAME

dc - desk calculator

SYNOPSIS

dc [file]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

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 may be preceded by an underscore (_) to input a negative number. Numbers may 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.

s*x*
: The top of the stack is popped and stored into a register named *x*, where *x* may 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. The 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 may 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.

; : are used by *bc* for array operations.

f generates a dump of the stack.

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. Base 1 generates a string of **1**'s whose length is the value of the number. Base 0 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. Given the definition of output base, the command **Op** will always yield "10", regardless of the base; **O1–p** yields useful information about the output base.

EXAMPLE

This example prints the first ten values of n! (n factorial):

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented
where *x* is an octal number.

stack empty
when there are not enough elements on the stack to do what was asked.

Out of space
when the free list is exhausted (too many digits).

Out of headers
when there are too many numbers being kept around.

Out of pushdown
when there are too many items on the stack.

Nesting Depth
when there are too many levels of nested execution.

NAME

dd - convert, reblock, translate, and copy a (tape) file

SYNOPSIS

dd [option=value] ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
...,...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks are trimmed and a new-line is added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output block of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

EXAMPLE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the ASCII file **x**:

```
dd if=/dev/rmt/0m of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary block sizes.

SEE ALSO

cp(1), tr(1).

DIAGNOSTICS

f+p blocks in(out) numbers of full and partial blocks read(written)

WARNING

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

for output to a cartridge tape, and

tcio -ivS 256 /dev/rct | dd ... | ...

for input from a cartridge tape.

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256–character standard in the CACM 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–lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

delta - make a delta (change) to an SCCS file

SYNOPSIS

delta [**-r**SID] [**-s**] [**-n**] [**-g**list] [**-m**[mrlist]] [**-y**[comment]] [**-p**] files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, *delta* 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 (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 keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see **-m** and **-y** keyletters below).

Keyletter arguments apply independently to each named file.

-r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter 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** keyletter 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 the issue, 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)) then 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** keyletter).

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 taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from **MR** number validation program, *delta* terminates (it is assumed that the **MR** numbers were not all valid).

-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) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*–file are explained in the *Source Code Control System 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 "gotten" file and the *g–file*.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

Lines beginning with an **SOH** ASCII character (octal 001) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) 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** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4).

SCCS User's Guide in *HP-UX Concepts and Tutorials.*

NAME

deroff - remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [**-mx**] [**-w**] [files]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl*(1) descriptions, perhaps 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** *troff* 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 may be followed by an **m**, **s**, or **l**. The **-mm** option causes the macros be interpreted so that only running text is output (i.e., 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."

SEE ALSO

eqn(1), nroff(1), tbl(1).

BUGS

Deroff is not a complete *troff* interpreter, so 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.

NAME

diff, diffh – differential file comparator

SYNOPSIS

diff [**–efbh**] file1 file2

/usr/lib/diffh file1 file2

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8–bit data, customs, messages.

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is –, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

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 backward 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 >.

The options are:

–b causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

–e produces a script of *a, c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*.

–f produces a script similar to that of **–e**, only it is not useful with *ed*, and it is in the opposite order.

–h does a fast, half-hearted job. It works only when changed stretches of text are short and well-separated, but does work on files of unlimited length. Options **–e** and **–f** are unavailable with **–h**.

Diffh is equivalent to **diff –h**. It must be invoked as shown above in the synopsis, unless the **PATH** variable in your environment includes the directory **/usr/lib**.

In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo 1,$p) | ed – $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

FILES

/tmp/d?????
/usr/lib/diffh for **–h**

SEE ALSO

cmp(1), comm(1), bdiff(1), diff3(1), diffmk(1), dircmp(1), ed(1), sccsdiff(1), sdiff(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the **−e** or **−f** option are naive about creating lines consisting of a single period (.).

WARNINGS

Missing newline at end of file X

indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

NAME

diff3 - 3–way differential file comparison

SYNOPSIS

diff3 [**-ex3**] file1 file2 file3

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

DESCRIPTION

Diff3 compares three versions of a file, and publishes 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 suffered in converting 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* to line *n2*. If *n1* = *n*2, the range may 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* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, 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 will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed - file1
```

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single **.** will defeat **-e**.
Files longer than 64K bytes will not work.

NAME

diffmk - mark differences between files

SYNOPSIS

diffmk name1 name2 name3

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff*(1) or *troff*(1). *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 | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, *diffmk* can 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
.nc `
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are 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 procedure).

SEE ALSO

diff(1), nroff(1), troff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

Although unlikely, certain combinations of formatting requests may cause change marks to either disappear or to mark too much. Manual intervention may be required as the subtleties of all the various formatting macro packages and preprocessors is beyond the scope of *diffmk*. The input to *tbl*(1) cannot tolerate **.mc** commands. Any **.mc** that would appear inside a **.TS** range will be silently deleted. The script can be changed if this action is inappropriate or *diffmk* can be run on the output from *tbl*(1).

Diffmk uses *diff*(1) and thus has whatever limitations on file size and performance that *diff* may impose. 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 *ed*(1), and 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.

NAME

dircmp - directory comparison

SYNOPSIS

dircmp [**-d**] [**-s**] [**-w***n*] dir1 dir2

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a 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.

SEE ALSO

cmp(1), diff(1).

NAME

du - summarize disk usage

SYNOPSIS

du [**-ars**] [names]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

DESCRIPTION

Du gives the number of 512-byte blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, **.** is used.

The optional argument **-s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **-a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. The **-r** option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

BUGS

If the **-a** option is not used, non-directories given as arguments are not listed.
If there are too many distinct linked files, *du* will count the excess files more than once.
Files with holes in them will get an incorrect block count.
If multiple links are involved, *du* can give different results, depending on the order of *names*.

NAME

echo - echo (print) arguments

SYNOPSIS

echo [arg] ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8-bit data

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. If *echo*'s arguments are not quoted, or are enclosed in double quotes (" ... "), all meta-characters are expanded according to the shell's interpretation. Thus, *echo* can be used to verify how a certain metacharacter pattern is going to be interpreted by the shell.

Echo also understands C-like escape conventions, which are listed below:

\b	backspace
\c	print line without 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- or 3-digit octal number *n*, which must start with a zero.

Note that these escape sequences are first interpreted by the shell before being passed to *echo*. Thus, if the arguments are unquoted, or quoted with double quotes, the backslash must be doubled to prevent premature interpretation by the shell. If the arguments are enclosed in single quotes, then the above escapes may be typed as shown.

To produce a literal backslash on the output, it must be doubled (if unquoted or quoted by double quotes). If quoted with single quotes, a single backslash suffices.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

NAME

ed, red - text editor

SYNOPSIS

ed [**-**] [**-p** string] [file]

red [**-**] [**-p** string] [file]

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: System V

Native Language Support:
8-bit and 16-bit data, customs, messages.

Remarks: The decryption facilities provided by this software are under control by the United States Government and cannot be exported without special licenses. These capabilities are considered an HP-UX/OPTIONAL feature, and can be sold only to domestic customers at this time.

DESCRIPTION

Ed is the standard (line-oriented) text editor. If the *file* argument is given, *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. The optional **-** suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the **!** prompt after a **!***shell command*. The **-p** option allows the user to specify a prompt string. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the 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*. It will only allow editing of files in the current directory. It prohibits executing shell commands via **!***shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(5) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty -tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be 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: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

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 may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character mentioned below is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, **[0-9]** is equivalent to **[0123456789]**. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., **[]a-f]** matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A currency symbol (**$**) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction **^** *entire RE***$** constrains the entire RE to match the entire line.

The null RE (e.g., **//**) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*–th line of the buffer.
4. **'***x* addresses the line marked with the mark name character *x*, which must be a lower–case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (**/**) addresses 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. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (**?**) 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. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (**+**) or a minus sign (**-**) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with **+** or **-**, the addition or subtraction is taken with respect to the current line; e.g, **-5** is understood to mean **.-5**.
9. If an address ends with **+** or **-**, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address **-** refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character **^** in addresses is entirely equivalent to **-**.) Moreover, trailing **+** and **-** characters have a cumulative effect, so **--** refers to the current line less 2.
10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the current line (**.**) is set to the first address, and only then is the second address 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 that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; **.** is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new–line character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; **.** is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; **.** is set to the last line of 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 typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *f*ile–name command changes the currently–remembered file name to *file*; otherwise, it prints the currently–remembered file name.

(1,$)g/*RE***/***command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with **.** 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 multi–line list except the last line must be ended with a ****; *a*, *i*, and *c* commands and associated input are permitted. The **.** terminating input mode may 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*. See also *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE***/**

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, **.** is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is

printed, and so on; a new–line acts as a null command; an **&** causes the 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*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; **.** is left at the last inserted line, or, if there were none, at 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. The maximum number of characters that may be entered from a terminal is 256 per line (including the new–line character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new–line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower–case letter. The address *'x* then addresses this line; **.** is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non–printing characters (e.g., *tab, backspace*) are represented by (hopefully) mnemonic overstrikes. All other non–printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes 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; **.** is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; **.** is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; **.** is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(\$)r *file*

The *r*ead command reads in the given file 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 causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; **.** is set to the last line read in. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "\$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/**g** or
(.,.)s/*RE*/*replacement*/**n** n = 1–512

The *s*ubstitute 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 the *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 may be used instead of / to delimit the RE and the *replacement*; **.** is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by \. 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 the *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 of more than one character or is preceded by a \.

A line may be split by substituting a new–line character into it. The new–line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); **.** is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,\$)v/*RE*/*command list*

This command is the same as the global command *g* except that the *command list* is executed with **.** initially set to every line that does *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 *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 your *umask* setting (see *sh*(1)) dictates otherwise. The currently–remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently–remembered file name, if any, is used (see *e* and *f* commands); **.** is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name. **($)=**

The line number of the addressed line is typed; **.** is unchanged by this command.

!*shell command*

The remainder of the line after the **!** is sent to the HP–UX shell (*sh*(1)) to be interpreted 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, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** 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**; it 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.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new–line.

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 may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

HARDWARE DEPENDENCIES

Series 500:

Certain older interface cards do not support **tty –tabs** or **stty tab3**. This precludes the use of the *fspec*(4) formatting capability.

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

SEE ALSO

awk(1), ex(1), grep(1), sed(1), sh(1), stty(1), vi(1), fspec(5), regexp(7).
The ed Editor, in *HP–UX Concepts and Tutorials.*

DIAGNOSTICS

? for command errors.

?*file* for an inaccessible file.
(use the *h*elp and *H*elp commands 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 *ed*'s buffer via the *e* or *q* commands. It prints **?**

and allows one to continue editing. A second *e* or *q* command at this point will take effect. The **-** command–line option inhibits this feature.

BUGS

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 RE does not match a new–line character.
The *l* command mishandles DEL.
Because 0 is an illegal address for the *w* command, it is not possible to create an empty file with
If the editor input is coming from a command file (i.e., ed file < ed–cmd–file), the editor will exit at the first failure of a command that is in the command file.

NAME

edit - text editor (variant of ex for casual users)

SYNOPSIS

edit [**-r**] name ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor. The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

BRIEF INTRODUCTION

To edit the contents of an existing file you begin with the command "edit name" to the shell. *Edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

Edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say "s/*old*/*new*/" where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No **write** since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "''". The **z** command can also be given other following characters "z-" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form **/text/** to search forward for *text* or **?text?** to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form **/^text/** which searches for *text* at the beginning of a line. Similarly **/text$/** searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$-5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. *Edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing "put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ex(1), vi(1).

NAME

enable, disable - enable/disable LP printers

SYNOPSIS

enable printers
disable [**-c**] [**-r**[reason]] printers

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

Native Language Support:
8-bit file names and data, customs, messages.

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) 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, then a default reason will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env - set environment for command execution

SYNOPSIS

env [-] [name=value] ... [command args]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8-bit filenames.

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 **-** flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1), exec(2), profile(5), environ(7).

NAME

err - report error information on last failure

SYNOPSIS

err

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: *Err* is implemented on the Series 500 only.

DESCRIPTION

Err produces error information on the standard output for the last command which failed. The *errno*, *errinfo*, and octal *trapno* values are listed.

Error information on the last child process which reported a failure is inherited across a *fork* and cleared by *exec*. The error values are also passed back from child to parent to grandparent as long as no errors were detected in the intermediate parent. Intervening commands which are executed successfully have no effect on the saved error information. If a command thinks it successfully completed, and returns an *exit* status of zero, no error information will be returned.

In general, the values reported are for a kernel intrinsic which failed, although values of *errno* or *errinfo* which are set by libraries or commands will also be reported.

SEE ALSO

errno(2), errinfo(2), trapno(2).

WARNING

This command may change in future releases of HP-UX. *Err* is intended for diagnostic purposes only.

BUGS

Information on a real error can be masked by "normal" errors caused by library routines or commands. For example, the library routine *isatty* will generate the error ENOTTY during normal operation.

NAME

ex - text editor

SYNOPSIS

ex [**-**] [**-v**] [**-t** *tag*] [**-r**] [**-R**] [+*command*] [**-l**] name ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

Remarks: The decryption facilities provided by this software are under control by the United States Government and cannot be exported without special licenses. These capabilities are considered an HP-UX/OPTIONAL feature, and can be sold only to domestic customers at this time.

DESCRIPTION

Ex is the root of a family of editors including: *ex, edit* and *vi. Ex* is a superset of *ed,* with the most notable extension being a display editing facility. Display based editing is the focus of *vi.*

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex.*

DOCUMENTATION

The *Ex Reference Manual* is a comprehensive and complete manual for the command mode features of *ex,* but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed;* the material in the introductory and advanced documents works also with *ex.*

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi.* The *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction.* The *vi*(1) manual page can also be used as reference.

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi.* Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base *terminfo*(4) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1).

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set

of files to be dealt with. In general, filenames in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming filenames and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join** (**j**) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

The following invocation options are interpreted by *ex*:

-	Suppress all interactive–user feedback. This is useful in processing editor scripts.
-v	Invokes *vi*
-t *tagfR*	Edit the file containing the *tag* and position the editor at its definition.
-r *file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
-R	*Readonly* mode set, prevents accidentally overwriting the file.
+*command*	Begin editing by executing the specified editor search or positioning *command*.
-l	**LISP** mode; indents appropriately for lisp code, the **()** **{}** [[and]] commands in *vi* are modified to have meaning for *lisp*.

The *name* argument indicates files to be edited.

Ex States

Command	Normal and initial state. Input prompted for by **:**. Your kill character cancels partial command.
Insert	Entered by **a** **i** and **c**. Arbitrary text may be entered. Insert is normally terminated by line having only **.** on it, or abnormally with an interrupt.
Visual	Entered by **vi**, terminates with **Q** or ^\.

Ex command names and abbreviations

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**			unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	window	**z**
insert	**i**	rewind	**rew**	escape	**!**
join	**j**	set	**se**	lshift	**<**
list	**l**	shell	**sh**	print next	CR
map		source	**so**	resubst	**&**
mark	**ma**	stop	**st**	rshift	**>**
move	**m**	substitute	**s**	scroll	**^D**

Ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	**?***pat*	previous with *pat*
$	last	*x*–*n*	*n* before *x*
+	next	*x*,*y*	*x* through *y*
-	previous	´*x*	marked with *x*
+*n*	*n* forward	´´	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment var.
$HOME/.exrc	editor initialization file
set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp		( ) { } are s-exp's
list		print ^I for tab, $ at end
magic		. [* special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in *vi*
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any char in *str*
[↑*str*]	... not in *str*
[*x*-*y*]	... between *x* and *y*
*	any number of preceding

AUTHOR

Vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/usr/lib/*/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ctags(1), ed(1), edit(1), grep(1), sed(1), vi(1), curses(3X), term(4), terminfo(4).

WARNINGS AND BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '-' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

expand, unexpand - expand tabs to spaces, and vice versa

SYNOPSIS

expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]
unexpand [**-a**] [file ...]

HP-UX COMPATIBILITY

Level: HP-UX/EXTENDED

Origin: UCB

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default only leading blanks and tabs are reconverted to maximal strings of tabs. If the **-a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

NAME

expr - evaluate arguments as an expression

SYNOPSIS

expr arguments

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8-bit filenames.

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may 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*
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 or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
multiplication, division, or 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. Regular expression syntax is the same as that of *ed*(1), 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 :.

EXAMPLES

1. a=`expr $a + 1`

   adds 1 to the shell variable **a**.

2. # 'For $a equal to either "/usr/abc/file" or just "file"'
   expr $a : '.*/\(.*\)' \| $a

   returns the last segment of a path name (i.e., file). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. # A better representation of example 2.
   expr //$a : '.*/\(.*\)'

   The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. expr $VAR : '.*'

   returns the number of characters in **$VAR**.

RETURN VALUE

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

SEE ALSO

ed(1), sh(1), test(1).

DIAGNOSTICS

syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string

BUGS

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 = '='

looks like:

expr = = =

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

expr X$a = X=

NAME

factor, primes - factor a number, generate large primes

SYNOPSIS

factor [number]

primes [start [stop]]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number, it factors the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and 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 *primes* is invoked without any arguments, it waits for two numbers to be typed in. The first number is interpreted as the lower bound, and 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, then all primes occurring in the inclusive range *"start - 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 until either overflow occurs or it is stopped by typing *interrupt.*

The largest number that can be dealt with by *primes* is 2,147,483,647.

DIAGNOSTICS

"Ouch" when the input is out of range, for garbage input, or when *start* is greater than *stop.*

NAME

fc, f77 - FORTRAN 77 compiler

SYNOPSIS

fc [options] files
f77 [options] files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

Native Language Support:
8-bit strings and comments.

Remarks: This manual page describes the FORTRAN 77 compiler as implemented on both the Series 200 and Series 500 computers. On both machines it is invoked as *f77* or *fc*.

DESCRIPTION

Fc is the HP-UX FORTRAN 77 compiler. It accepts several types of file arguments:

(1) Arguments whose names end with **.f** are taken to be FORTRAN 77 source files. They are compiled, and each object file is left in the current directory in a file whose name is that of the source, with **.o** substituted for **.f**. (The **.o** file will not be created for a single source which is compiled and loaded, nor for any source which fails to compile correctly.) *Series 200 Only:* In the same way, arguments whose names end with **.c** or **.s** are taken to be **C** or **assembly source** programs and are compiled or assembled, producing **.o** files.

(2) *Series 200 Only:* Arguments whose names end with **.r** are taken to be *ratfor*(1) source programs. These are first transformed by the *ratfor* preprocessor, and then compiled by *f77* producing **.o** files.

(3) Arguments whose names end with **.o** are passed on to the linker (*ld*(1)) to be linked into the final program.

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,

```
FCOPTS=-v
export FCOPTS
fc -L prog.f
```

is equivalent to

```
fc -v -L prog.f
```

The following options are recognized:

-c suppress linking and produce object (**.o**) files from source files.

-C enable range checking (same as **$OPTION RANGE ON**).

-D compile debug lines (source lines with a "D" or "d" in column 1 are treated as comments by default).

-g causes the compiler to generate additional information needed for the use of a symbolic debugger. (This option may be incompatible with optimization.)

-I2 make default size of integers and logicals **INTEGER*2** and **LOGICAL*2** (same as **$OPTION SHORT**).

-I4 make default size of integers and logicals **INTEGER*4** and **LOGICAL*4**. This is the compiler's default.

-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 which were written to depend on static allocation of memory (i.e. variables retaining their values between invocations of the respective program units).

-l*x* causes the linker to search the library named by either **/lib/lib***x***.a** (tried first) or **/usr/lib/lib***x***.a**. (See *ld*(1).)

-L write a program listing to *stdout* during compilation.

-n causes the output file from the linker to be marked *shared.*

-N causes the output file from the linker to not be marked *shared.*

-o *outfile* name the output file from the linker *outfile* instead of *a.out.*

-onetrip execute any DO loop at least once.

-O invoke the assembly code optimizer.

-p prepare object files for profiling (see *prof*(1)).

-q causes the output file from the linker to be marked *demand load.*

-Q causes the output file from the linker to not be marked *demand load.*

-s causes 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 of an 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.

For the Series 200, *c* can take one or more of the values:

r ratfor preprocessor (standard suffix is *ratfor*)
c compiler body (standard suffix is *f77pass1*)
0 same as *c*
1 compiler code generator (suffix is *f1*)
2 optimizer (standard suffix is *c2*)
a assembler (standard suffix is *as*)
l linker (standard suffix is *ld*)

For the Series 500, *c* can take one or more of the values:

c compiler body (standard suffix is *f77comp*)
0 same as *c*
l linker (standard suffix is *ld*)

-u force types of identifiers to be implicitly undeclared (same as specifying **IMPLICIT NONE**; no other **IMPLICIT** statements are permitted).

-U use upper case for external names (default is lower case).

-v enable the verbose mode, producing a step–by–step description of the compilation process on *stderr.*

-w suppress warning messages (same as **$OPTION WARNINGS OFF**).

-w66 suppress warnings about FORTRAN 66 features used.

-W *c,arg1[,arg2,...,argN]*
causes *arg1* through *argN* to be handed off to subprocess *c*. 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 *c* can assume are those listed under the **-t** option, as well as *d* (driver program) which has a special meaning explained below.

-Y enable 8- and 16-bit NLS support in strings and comments. In the default case, NLS is not enabled.

The **-W d** option specification allows additional, implementation-specific options to be recognized and passed through the compiler driver to the appropriate subprocesses (see **-W** above). For example, on the Series 500,

-W d,-Q,dfile,-e

will send the options *-Q dfile* and *-e* through the compiler driver. Furthermore, a shorthand notation for this mechanism can be used by prepending + to the option name; as in

+Q dfile +e

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**).

The implementation-specific options on the Series 200 are:

+b causes the compiler to generate code for floating point operations that will use floating point hardware if it is installed in the computer at run-time.

+f causes the compiler to generate code for floating point operations that will use floating point hardware. This code does not run unless floating point hardware is installed.

+k this option forces dynamic storage for local arrays. If specified, arrays are subject to the 32K byte limitation for local data space.

+N< *secondary*><*n*>
This option adjusts the size of internal compiler tables. The compiler uses fixed size arrays for certain internal tables. *Secondary* is one of the letters from the set {**qsxcnaet**}, and *n* is an integer value. *Secondary* and *n* are **not** optional. The table sizes can be re-specified using one of the secondary letters and the number *n* as follows:

- **q** maximum size of equivalence table (default = 150 table entries).
- **s** maximum size of statement label table (default = 201 table entries).
- **x** maximum size of external symbol table (default = 200 table entries).
- **c** maximum size of control statements table (default = 20 table entries).
- **n** maximum size of the hash table of symbols (default = 401 table entries).
- **a** maximum size of external label name storage table (default = 10000 bytes).
- **e** maximum number of expression tree nodes (default = 1000 entries).
- **t** maximum size of external symbol storage table (default = 40000 bytes).

+s issue warnings for non-ANSI features (same as **$OPTION ANSI ON**).

+U upper and lower case are distinguished (case is significant). Keywords are only recognized in lower case.

The implementation–specific options on the Series 500 are:

+e write errors to *stderr*.

+F causes the compiler to generate information used by various program analysis programs.

+Q *dfile* specify *dfile* as the option file.

+s issue warnings for non–ANSI features (same as **$OPTION ANSI ON**).

+T causes the running program to issue a procedure traceback for runtime errors.

+Vc put all COMMONs in the virtual data area.

+Vd put all SAVE'd and initialized (DATA statement) variables in the virtual data area.

+Vf put all FORMAT strings in the virtual data area.

Any other options encountered will generate a warning to *stderr*.

HARDWARE DEPENDENCIES

Series 200:

The following options are not implemented:
-D, –L, –U

Series 500:

The following options are not implemented:
-O, –p, –S, –w66

FILES

Series 200:

file.r	input file (*ratfor* source file)
file.f	input file (FORTRAN 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
/usr/bin/f77	mother program (linked to /usr/bin/fc)
/usr/lib/f77pass1	compiler pass 1
/lib/f1	compiler pass 2
/lib/c2	assembly code optimizer
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	FORTRAN I/O library
/lib/libc.a	C library; See Section 3 of this manual
/lib/libm.a	math library
/lib/frt0.o	run–time startoff routine
/lib/mfrt0.o	startoff with profiling
/usr/lib/end.o	symbolic debugger string buffer

Series 500:

file.f	input file (FORTRAN source file)
file.o	object file
a.out	linked executable output file
/bin/fc	mother program
/usr/lib/f77comp	compiler
/lib/frt0.o	runtime startup
/usr/lib/end.o	symbolic debugger string buffer
/lib/libI77.a	FORTRAN I/O library
/lib/libF77.a	FORTRAN math library
/lib/libc.a	C library; See Section 3 of this manual

/lib/libm.a	math library
/usr/tmp/*	temporary files used by the compiler; names are created by *tmpnam*(3S).

SEE ALSO

as(1), asa(1), cc(1), ld(1), strip(1).

FORTRAN 77 programming and reference manuals for your HP-UX system and *Structured FORTRAN 77* by Seymour Pollack.

DIAGNOSTICS

The diagnostics produced by *fc* are intended to be self-explanatory. If a listing is requested (**-L** option), errors are written to the listing file. If no listing is being generated, errors are written to *stderr*. *Series 500 Only:* Errors will be written to both the listing file and *stderr* if the **-L** and **+e** options are both specified. Occasional messages may be produced by the linker.

BUGS

The **-s** option has a new meaning; use **+s** for non-ANSI warnings.
Series 200: The **-U** option has a new meaning; use **+U** for case sensitivity.
Series 500: The **-Q dfile** option has a new meaning; use **+Q dfile** to specify an option file.

NAME

file - determine file type

SYNOPSIS

file [**-c**] [**-f** ffile] [**-m** mfile] arg ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable *a.out* file, *file* will print 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 its format.

The options are as follows:

-c causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file classification is done under **-c**.

-f*ffile* specifies that *ffile* is a file containing a list of the files which are to be examined. *File* then classifies each file whose name appears in *ffile*.

-m*mfile* instructs *file* to use an alternate magic file.

SEE ALSO

ld(1).

NAME

find – find files

SYNOPSIS

find path-name-list expression

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8–bit filenames.

DESCRIPTION

Find 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. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, −*n* means less than *n* and *n* means exactly *n*.

−name *string* True if *string* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for [, ? and *).

−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**, **f**, or **l** for block special file, character special file, directory, fifo (a.k.a named pipe), plain file, or symbolic link respectively.

−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*[**c**] True if the file is *n* blocks long. If *n* is followed by a **c**, the size is in characters.

−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 has been changed in *n* days.

−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.

−cpio *device* Always true; write the current file on *device* in *cpio*(5) format (5120-byte records). By default, *find* will not follow symbolic links that point to directories when this option is specified. The **−follow** option may be used to follow symbolic links that point to directories.

−follow Always true; causes *find* to recursively descend symbolic links that point to directories. (This is the default when the **−cpio** option is not specified.) Not

all HP-UX systems support symbolic links.

-nofollow Always true; causes *find* to not recursively descend symbolic links that point to directories.

-newer *file* True if the current file has been modified more recently than the argument *file*.

-depth Always true; 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.

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

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

-ncpio*device* Same as **-cpio** but adds the -c option to **cpio**.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (**!** is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**-o** is the *or* operator).

HARDWARE DEPENDENCIES

Series 200/300/500:

Symbolic links are not supported on Series 200, 300, and 500 at this time.

EXAMPLES

To 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.

FILES

/etc/passwd, /etc/group

SEE ALSO

cpio(1), sh(1), test(1), stat(2), lstat(2), cpio(5), fs(5).

find / -mtime +7 -print

NAME

findmsg, dumpmsg – create message catalog file for modification

SYNOPSIS

findmsg file ...
dumpmsg file ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

DESCRIPTION

Findmsg extracts messages from C program source *file* and writes them to the standard output, along with set information. The source file lines from which the string literals are to be extracted must have *nl_msg* and " in the same line. There are four cases to be handled:

```
printf(nl_msg(1, "message"));

#define NLSMESS "message"    /* nl_msg 1 */

char nlsmess[] = "message"       /* nl_msg 1 */

char *nlsmess[] = {
        "message 1",    /* nl_msg 1 */
        "message 2",    /* nl_msg 2 */
        0
};
```

In each of the latter three cases, there are executable lines elsewhere which contain nl_msg in an executable form, along with the necessary reference.

Findmsg derives message catalog set numbers from source lines which appear as:

#define NL_SETN 1

Typically a single such line will appear toward the beginning of the source file.

Dumpmsg dumps out messages which are stored in a message catalog file which was generated by the *gencat(1)* command.

The output of either command is in the form:

```
$set 1
1 message1\n
2 message two\n
```

Each message can then be changed as necessary, then processed by the *gencat(1)* command.

SEE ALSO

findstr(1), gencat(1), insertmsg(1), getmsg(3C).

BUGS

For use with *gencat*(1), the output of *findmsg* must have the **$set** line appear first in its output. Thus the

#define NL_SETN 1

must appear before any messages.

Only one message may appear on each physical line. Each message must appear completely on one line along with the *nl_msg* token.

NAME

findstr – find strings for inclusion in message catalogs

SYNOPSIS

findstr file ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

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 will be used by *insertmsg*.

SEE ALSO

insertmsg(1).

NAME

fixman - fix manual pages for faster viewing with man(1)

SYNOPSIS

fixman

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

This shell script processes all ordinary files under /usr/man/cat* to unexpand all possible spaces to tabs and remove all {character, backspace} pairs. Such pairs usually exist to cause overstriking or underscoring for printer output. They only slow down man(1), and use up significant amounts of disc space. The script should be run after running catman(1) to rebuild all cat-able manual entries from pre-nroff forms.

The script does not remove duplicate blank lines, so all files remain a multiple of one page (66 lines) long and can still be passed directly to lp(1). (Note that man(1) normally uses rmnl(1) to accomplish this removal.)

To insure success, the script should be run by the super-user. It can take two to three hours to complete. As a side-effect, file ownerships and permissions may be changed.

FILES

/usr/man/cat*

Directories containing post-nroff versions of manual entries.

SEE ALSO

catman(1), chmod(1), expand(1), lp(1), man(1), mv(1), rmnl(1), sed(1).

NAME

fold - fold long lines for finite width output device

SYNOPSIS

fold [-width] [file ...]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

DESCRIPTION

Fold is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1) before coming to *fold*.

SEE ALSO

expand(1)

BUGS

If underlining is present it may be messed up by folding.

NAME

gencat - generate a formatted message catalog file

SYNOPSIS

gencat catfile file ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

Native Language Support:
8-bit and 16-bit data, customs, messages

DESCRIPTION

Gencat merges message source *files* into a formatted *catfile* which can be accessed by *getmsg*(3C). If *catfile* does not exist it will be created. If *catfile* does exist its messages will be included in the new *catfile* unless set and message numbers collide, in which case the new supersedes the old. The *files* consist of sets of messages along with comments.

The format for message source in the *files* has been designed to include compatibility with MPE and RTE. A line which begins with a dollar sign followed by a blank denotes a comment and may appear anywhere in a file.

A message set consists of a line of the form

$set*n* [*comment*]

followed by lines of the form

m message-text

where *n* denotes the set number (1–255) and *m* the message number (1–32767). Typically the set number will be used to identify the language, while the message number denotes which string from a given program is wanted. *Message-text* is a C string, including white space and '\' escapes, without the surrounding quotes. A **$set** line may optionally contain comment text following the set number. Set numbers and message numbers must be in ascending order but need not be contiguous.

If a message source line has a number but no text then the existing message with this number is deleted from the catalog.

To delete an entire message set the directive

$DELSET *set_name*

may be placed at the beginning of a line between sets.

SEE ALSO

findmsg(1), insertmsg(1), getmsg(3C).

NAME

get - get a version of an SCCS file

SYNOPSIS

get [**-r**SID] [**-c**cutoff] [**-i**list] [**-x**list] [**-a**seq–no.] [**-k**] [**-e**] [**-l**[p]] [**-p**] [**-m**] [**-n**] [**-s**] [**-b**] [**-g**] [**-t**] file ...

HP–UX COMPATIBILTY

Level: HP–UX/STANDARD

Origin: System III

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with **-**. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* 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. 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 leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter 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*(1) if the **-e** keyletter is also used), as a function of the SID specified.

-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 may separate the various 2 digit pieces of the *cutoff* date–time. This feature allows one to specify a *cutoff* date in the form: *"***-c77/2/2 9:22:25***"*. Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

~!get "-c%E% %U%" s.file

-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*(1). The **-e** keyletter 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.

If the *g–file* generated by *get* with an **-e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re–executing the *get* command with the **-k** keyletter in place of the **-e** keyletter.

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** keyletter is used.

-b Used with the **-e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter 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* may 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, may 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.

-x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **-i** keyletter for the *list* format.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** keyletter is implied by the **-e** keyletter.

-l[p] Causes a delta summary to be written into an *l–file*. If **-lp** is used then 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*.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No *g–file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **-s** keyletter 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** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **-m** keyletter 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**).

-a*seq–no.* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the **-r** and **-a** keyletters are specified, the **-a** keyletter is used. Care should be taken when using the **-a** keyletter in conjunction with the **-e** keyletter, as the SID of the delta to be created may not be what one expects. The **-r** keyletter can be used with the **-a** and **-e** keyletters 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** keyletter 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** keyletter is used included deltas are listed following the notation "Included"; if the **-x**

keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**-b** Keyletter 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** keyletter 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 may 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 shouldn't 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*(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%

FILES

Several auxiliary files may 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** keyletter 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** keyletter 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** keyletter 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:

a. A blank character if the delta was applied;
   * otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   * if the delta wasn't applied and wasn't ignored.
c. A code indicating a "special" reason why the delta was or was not applied:
   "I": Included.
   "X": Excluded.
   "C": Cut off (by a **-c** keyletter).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. 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** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **-e** keyletter 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** keyletter argument if it was present, followed by a blank and the **-x** keyletter 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.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(4).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the **-e** keyletter is used. Some list sequences will not work properly with the –i and/or –x options.

NAME

getopt - parse command options

SYNOPSIS

getopt optstring args

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

Native Language Support:
8-bit filenames.

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 special option **--** is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. 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.

The most common use of *getopt* is in the shell's **set** command (see the example below). There, *getopt* converts the command line to a more easily parsed form. *Getopt* writes the modified command line to the standard output.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)        FLAG=$i; shift;;
        -o)             OARG=$2; shift 2;;
        --)             shift; break;;
        esac
done
```

This code will accept 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
```

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

getprivgrp - get special attributes for group

SYNOPSIS

getprivgrp [group-name]

HP-UX COMPATIBILITY

Level: HP-UX

Origin: HP

DESCRIPTION

Getprivgrp lists the access privileges of privileged groups set by *setprivgrp(1m)*. When a group name is supplied access privileges are listed for that group only. Otherwise, access privileges are listed for all privileged groups of which the caller is a member. The super-user is considered to be a member of all groups. Access privileges include *rtprio* and *mlock*.

SEE ALSO

getprivgrp(2), setprivgrp(1m), privgrp(5).

NAME

grep, egrep, fgrep - search an ASCII file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

-v All lines but those matching are printed.
-x (Exact) only lines matched in their entirety are printed (*fgrep* only).
-c Only a count of matching lines is printed.
-i Ignore upper/lower case distinction during comparisons (*grep* only).
-l Only the names of files with matching lines are listed (once), separated by new-lines.
-n Each line is preceded by its relative line number in the file.
-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.
-s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).
-e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a **-** (does not work with *grep*).
-f *file* The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, 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 **'...'**.

Fgrep searches for lines that contain one of the *strings*, each of which is separated from the next by a new-line.

Egrep accepts regular expressions as in *ed*(1), except for \(and \), with the addition of:

1. A regular expression followed by + matches one or more occurrences of the regular expression.
2. A regular expression followed by **?** matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by | or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses **( )** for grouping.

The order of precedence of operators is [ ], then * ? +, then concatenation, then | and new-line.

EXAMPLES

The following example searches two files, finding all lines containing occurrences of any of four strings:

```
fgrep 'if
then
else
```

fi´ script1 script2

Note that the single quotes are necessary to tell *fgrep* when the strings have ended and the file names have begun.

This example searches for a new–line in a file:

grep -v ´\.´ file1

The **-v** option causes *grep* to print those lines that do not match the expression. Since a new–line cannot be matched with dot, only lines containing a new–line are printed.

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we do not know a single algorithm that spans a wide enough range of space–time tradeoffs.
Lines are limited to BUFSIZ characters; longer lines are truncated. (BUFSIZ is defined in **/usr/include/stdio.h**.)
Egrep does not recognize ranges, such as **[a-z]**, in character classes.
Grep finds lines in the input file by searching for a new–line. Thus, if there is no new–line at the end of the file, *grep* will ignore the last line of the file.
If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

groups – show group memberships

SYNOPSIS

groups [[-p] [-g] [-l] user]

HP-UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: UCB

DESCRIPTION

The *groups* command shows the groups to which you or the optionally specified user belong. If invoked with no arguments, *groups* prints the current access list returned by *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(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 list which is printed to only those groups specified in /etc/passwd, /etc/group, and /etc/logingroup, respectively.

SEE ALSO

id(1), newgrp(1), getgroups(2), initgroups(3c), group(5)

FILES

/etc/passwd, /etc/group, /etc/logingroup

NAME

head - give first few lines

SYNOPSIS

head [-count] [file ...]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

SEE ALSO

tail(1).

NAME

help - ask for help

SYNOPSIS

help [args]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may 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., **ge5**, for message 5 from the *get* command).

type 2 Does not contain numerics (as a command, such as **get**)

type 3 Is all numeric (e.g., **26**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

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**.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Only SCCS and a very few other commands currently use *help*.

NAME

hostname - set or print name of current host system

SYNOPSIS

hostname [nameofhost]

HP-UX COMPATABILITY

Level: HP-UX/STANDARD

Origin: UCB

DESCRIPTION

The *hostname* command prints the name of the current host, as given in the *uname* system call. The super-user can set the hostname by giving an argument; this is usually done in the startup script */etc/rc*.

SEE ALSO

uname(1), gethostname(2), sethostname(2), uname(2).

NAME

hp - handle special functions of HP 2640 and 2621–series terminals

SYNOPSIS

hp [**-e**] [**-m**]

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System III

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

Hp supports special functions of the Hewlett–Packard 2640– and 2621– series of terminals, with the primary purpose of producing accurate representations of most *nroff*(1) output. A typical use is:

nroff -h files ... | hp

Regardless of the hardware options on your 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.

The options are as follows:

-e It is assumed that your terminal has the "display enhancements" feature, and so maximal use is made 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, i.e., dark–on–light, rather than the usual light–on–dark.

-m Requests minimization of output by removal of new–lines. Any contiguous sequence of 3 or more new–lines is converted into a sequence of only 2 new–lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set as does *300*(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*(1).

DIAGNOSTICS

"line too long" if the representation of a line exceeds 1,024 characters.
The exit codes are **0** for normal termination, and **2** for all errors.

SEE ALSO

300(1), col(1), neqn(1), greek(1), nroff(1), tbl(1).

BUGS

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*(1) 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*(1).
Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hyphen - find hyphenated words

SYNOPSIS

hyphen [files]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

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* may be used as a filter.

EXAMPLE

The following will allow the proofreading of *nroff* hyphenation in *textfile*.

```
mm textfile | hyphen
```

SEE ALSO

mm(1), nroff(1).

BUGS

Hyphen cannot cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

id - print user and group IDs and names

SYNOPSIS

id

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

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.

SEE ALSO

logname(1), getgid(2), getuid(2).

NAME

insertmsg - use *findstring* output to insert calls to *getmsg*

SYNOPSIS

insertmsg stringlist

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

DESCRIPTION

Insertmsg examines the file *stringlist*, which is assumed to be the output of *findstr* minus the strings which do not need to be localized and have been removed by editing. *Insertmsg* first places the line

```
#include <msgbuf.h>
```

at the beginning of each file named in *stringlist.* Then for each line in *stringlist ,* it surrounds the string with an expression of the form

```
(*getmsg(nl_fd, nl_set_num, nl_msg_num, nl_msg_buf,
 NL_MBUFLN) == '\0' ? "saved string" : nl_msg_buf)
```

which evaluates to the original string if the translation cannot be retrieved. The string buffer and other "nl_" variables and constants are defined in *<msgbuf.h>*. *Insertmsg* places the modified source on a file *nl_xx.c* where the original file name was *xx.c*. The user must then hand edit the file to insert a call

```
nl_catopen("/*appropriate message catalog*/");
```

and assign the proper value to *nl_set_num*.

Insertmsg also places on the standard output a file which can be used as input to *gencat*. Again, hand editing is required to define the *$set* number to match *nl_set_num*. Messages will automatically be numbered from 1 upward, in the order that they appear in *stringlist*. The same number will also be placed in the call to *getmsg*(3C), as the parameter *msg_num*.

DIAGNOSTICS

If *insertmsg* doesn't find the opening or closing double quote where it expects it in the strings file, it prints "insertmsg exiting : lost in strings file" and dies. If this happens check the strings file to make sure that the lines that have been kept there haven't been altered.

SEE ALSO

findstr(1), gencat(1), getmsg(3C).

BUGS

Inserts a pointer to a static area which is overwritten on each call.

NAME

ipcrm - remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

-q *msqid* removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

-m *shmid* removes 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* removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

-Q *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

-M *shmkey* removes 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* removes the semaphore identifier, created with key *semkey*, from the system and destroys 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 may be found by using *ipcs*(1).

SEE ALSO

ipcs(1).

msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

NAME

ipcs - report inter–process communication facilities status

SYNOPSIS

ipcs [options]

HP–UX COMPATIBILITY

Level: HP–UX/STANDARD

Origin: System V

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 that is displayed is controlled by the following *options*:

585.sp0u

-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 biggest 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 meaning 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 the 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*(2) on semaphores.) See below.
-a Use all print *options*. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)
-C *corefile*
Use the file *corefile* in place of **/dev/kmem**. This option is not supported on the Series 500.
-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) Type of the facility:
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.

FILES

/hp-ux	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2).

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

join - relational database operator

SYNOPSIS

join [options] file1 file2

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is **-**, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, 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 blank.

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. The following options are recognized:

-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*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.

-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.

EXAMPLE

The following command line will join 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 ASCII collating sequence on the group ID fields.

```
join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

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*(1) are incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

NAME

kill - terminate a process

SYNOPSIS

kill [-signo] PID ...

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill 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 *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by - is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill -9 ..." is a sure kill.

SEE ALSO

ps(1), sh(1), kill(2), signal(2).

BUGS

If a process becomes hung during some operation (such as I/O) so that it is never scheduled, that process will not die until it is allowed to run. Thus, such a process may never go away after the kill.

NAME

last - indicate last logins of users and teletypes

SYNOPSIS

last, lastb [-N] [name ...] [tty ...]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

Last will look back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. *Last* will print the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

Last with no arguments prints a record of all logins and logouts, in reverse order. The **-N** option limits the report to N lines.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) *last* indicates how far the search has progressed so far, and the search continues.

Lastb will look back in the *btmp* database to display bad login information.

FILES

/usr/adm/wtmp	login data base
/usr/adm/btmp	bad login data base

SEE ALSO

login(1), wtmp(5)

NAME

ld - link editor

SYNOPSIS

ld [[option] ... [file] ...] ...

HP–UX COMPATIBILITY

Level: HP–UX/NUCLEUS

Origin: HP

DESCRIPTION

Ld takes one or more object files 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 it is present in the file). By default, *ld* processes one or more object files to produce an executable file that can be run by the HP–UX loader *exec*(2). Alternatively, the linker can generate a relocatable file – one suitable for further processing by *ld* (see **-r** below). *Ld* will not generate an output file if any errors occur during its operation.

Ld recognizes two kinds of input files: object files created by the compilers or assembler (also known as ".o" files) and archives of such object files (called libraries). A library contains an index of all the externally–visible symbols from its component object files. (The archiver command *ar* 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 a 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.

OPTIONS

Ld options may occur anywhere on the command line following the command name itself. Some options require a modifier following the option letter (e.g. ... **-o** *outputname*). *Ld* recognizes modifiers either as part of the word containing the option letter, or as a separate word following the option letter. This is the same convention as used by *getopt*(3).

Ld recognizes the following options. In the descriptions below, a colon following an option letter indicates that it takes an additional modifier. The colon itself is *not* literal, and must not appear on the command line.

–N generate an (executable) output file that is not shareable.

–Q ensure that the output file is **not** marked as demand–loadable. (This is the complement of the **–q** option.)

–V : specifies a version stamp (in decimal) to identify the output file. This is not the same as the version information reported by the SCCS *what*(1) command.

–X : indicates the initial size for the linker's global symbol table. Thus you can reduce link time for very large programs (i.e. those with very many external symbols). Note: the linker expands its data structures as necessary no matter what their starting size. provides the user with a means of controlling the size of the linker's symbol table. It is useful for dealing with very large applications that contain a great many externally visible symbols.

–d forces definition of "common" storage (i.e. assign addresses and sizes), even for **–r** output.

–e : names an alternate entry point for the output file. The loader will then call the alternate entry point to initiate the program. (Thus this option only applies to executable files.)

–h : 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*. (Of course, this only makes sense with the **-r** option.)

–l : abbreviation for a library name. *Ld* searches for a library called **lib***x***.a**, where *x* is a string of up to nine ASCII characters specified as the modifier to the **–l** option. A null string for *x* is the same as specifying **–lc**.

In this case, the linker searches specific directories for libraries. By default, it first looks in **/lib**, and then in **/usr/lib**. Since *ld* searches files (including libraries) in the same order that they are named on the command line, the placement of **–l** options is important.

–n generate an (executable) output file with code to be shared by all users. Compare with **–N**.

–o : specifies a name for the output file that the linker generates. (The default name is **a.out**.)

–q generate an (executable) output file that is demand–loadable. Compare with **–Q**.

–r retains relocation information in the output file for subsequent re–linking. *Ld* will not report undefined symbols.

–s "strip" the output file so that it does not contain symbol table, relocation, and debug support information. This may 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. The file names appear once for **each** pass over the input (usually two).

–u : indicates a name to enter as an undefined symbol in the linker's symbol table. The resulting unresolved reference is useful for linking a program entirely from object files in a library.

DEFAULTS

Unless otherwise directed, *ld* names its output **a.out**. The **-o** option overrides this. Executable output files are marked as shareable.

EXAMPLES

The following command line links part of a C program for later processing by *ld*. It also specifies 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 –V2 –r file1.o file2.o –o prog.o
```

The next example links a simple FORTRAN program for use with the *cdb*(1) symbolic debugger. The output file name will be *a.out* since there is no **–o** option in the command line.

```
ld –e start /lib/frt0.o ftn.o –lI77 –lF77 –lm –lc /usr/lib/end.o
```

Finally, this command will link a PASCAL program.

```
ld –e start /lib/prt0.o main.o –lpccat –lpc –lm –lc
```

HARDWARE DEPENDENCIES

Series 200:

The default entry point is taken to be text location 0x0 (which is also the default origin of the program text). This corresponds to the first procedure in the first input file that the linker reads. Use the **-e** option to select a different entry point.

The version number specified with the **–V** option must be in the range 0 – 32,767.

These options are specific to the Series 200 linker:

–R : specifies an alternate origin (in hexadecimal) for the text (i.e. code) segment.

–x partially "strip" the output file: i.e. 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** may impact the use of a debugger.

Series 500:

The linker searches for **_main** (written as *main* in C) as the main entry point for a user program. Use the **-e** option to select a different entry point.

The special names **etext** and **edata** are not supported.

The linker marks output files with the following memory management attributes by default: virtual code, virtual data (both D–data and I–data), and paged I–data. Executable output files are not shareable if they contain symbolic debug information.

The **–t** option displays file names twice, once for **each** pass over the input.

These options are specific to the Series 500 linker:

–A forces the linker to put D–data and I–data in separate segments.

–M :
enables merging of code segments. The integer argument specifies a target upper bound on the size of output code segments. (The actual size may vary from this.)

–T forces the linker to put D–data and I–data into the same segment.

–v display verbose messages. This option may have little or no effect. It is useful for obtaining more information about an error that occurs while linking.

Unless the user specifies a **–A** or a **–T** option, the linker puts all data in a single segment (GDS) when the total data size is less than or equal to 16,384 bytes.

FILES

/lib/crt0.o	run–time start–up for C
/lib/frt0.o	run–time start–up for FORTRAN
/lib/prt0.o	run–time start–up for Pascal.
/usr/lib/end.o	for use with *cdb/fdb/pdb*(1)
/lib/lib*x*.a	libraries
/usr/lib/lib*x*.a	libraries
a.out	output file

SEE ALSO

ar(1), cc(1), cdb(1), fc(1), nm(1), pc(1), strip(1), exec(2), end(3), a.out(5), ar(5).

DIAGNOSTICS

Ld returns a zero when the link is successful. A non–zero return code indicates that an error occurred.

WARNINGS

Ld recognizes several names as having special meanings. The names **_end**, **_edata**, and **_etext** (**end**, **edata**, and **etext** in C) are reserved. (See *end*(3) for details.) Users must not write alternative (externally–visible) definitions for these names.

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 insure 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 the user calls *ld*

directly, then he or she must insure that the program **always** calls *exit*(~) rather than falling through the end of the entry routine.

When linking for use with the symbolic debugger *cdb*, the user must ensure that the program contains a routine called *main* and furthermore that *main* is the initial entry point for executing the program. (Thus the **–e** option is compatible with debugging.) 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 libraries and include them in the final program in the same relative order that they occur within the library.

NAME

leave - remind you when you have to leave

SYNOPSIS

leave [hhmm]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

DESCRIPTION

Leave waits until the specified time, then reminds you that you have 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 just before it would have printed the next message.

The time of day is in the form hhmm where hh is a time in hours (on a 12 or 24 hour clock). All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, *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*.

Leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use "kill -9" giving its process id.

SEE ALSO

calendar(1)

NAME

lex - generate programs for lexical analysis of text

SYNOPSIS

lex [**-rctvn**] [file] ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input *files* 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 may 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. 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 within " symbols 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 you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. 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 **stdin** and **stdout**, 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. Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with **{}**. Note that curly brackets do not imply parentheses; only string substitution is done.

The flags, which must appear before any *file*s, are as follows:

-r indicates *ratfor*(1) actions;

-c indicates C actions - this is the default;

-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;

-n suppresses printing of the - summary.

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 2000);

%n *n* number of states is *n* (default is 500);

%t *n* number of parse tree nodes is *n* (default is 1000);

%a *n* number of transitions is *n* (default is 3000).

The use of one or more of the preceding table options automatically implies **-v**, unless **-n** is specified.

External names generated by *lex* all begin with the prefix **yy** or **YY**.

EXAMPLE

```
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: go to loop;
                        }
                }
```

SEE ALSO

yacc(1), malloc(3X).

LEX - Lexical Analyzer Generator, in *HP-UX Concepts and Tutorials.*

BUGS

The **-r** option is not yet fully operational.

NAME

lifcp - copy to or from LIF files

SYNOPSIS

lifcp [**-T**xxx] [**-L**xxx] [**-v**xxx] [**-b**] [**-i**xxx] [**-r**] [**-t**] file1 file2
lifcp [**-T**xxx] [**-L**xxx] [**-v**xxx] [**-b**] [**-i**xxx] [**-r**] [**-t**] file1 [file2 ...] directory

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

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 may appear singly or be 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 will force the file type of the LIF directory entry to be set to the argument given, which may be decimal, octal or hex in 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 one.

-v xxx
Used only when copying files to a LIF volume. This option will set the "volume number" to **xxx.** The default "volume number" is one.

-b This option will force 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(5)). This option is a no-op 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, which may be decimal, octal or hex in 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).

-r This option will force a RAW mode of copying regardless of the file type. When copying in RAW mode from HP-UX to LIF the default file type is BIN(-23951). **-T** option will override the default file type. (various modes of copying are explained in **LIF(5).)** This option is a no-op in case of LIF to LIF copying.

-t will cause the HP-UX file names to be translated to a name acceptable by a LIF utility. That is, all the lower-case letters will be up-shifted and all other characters except numeric will be changed to an underscore (_). If the HP-UX file name starts with a non-letter, the file name will be preceded by the capital letter (X). Note that if there are two files named colon (:) and semicolon (;), both of them will be translated to X_. File names will be truncated to a maximum of 10 characters. When copying a LIF file to (HP-UX/LIF) file **-t** is a no-op. Omitting **-t** will cause 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 content of the file are duplicated from source to destination.

A LIF file name is recognized by the embedded colon (:) delimiter (see *lif*(5) 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) will be 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.

The LIF file naming conventions are known only by the LIF utilities. Since file name expansion is done by the shell, this mechanism cannot be used for expansion of LIF file names.

Note that the media should **not** be mounted while using *lifcp*.

HARDWARE DEPENDENCIES

Series 500:

You **must** use a character special file to access the media.

EXAMPLES

lifcp abc lifvol:CDE

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 –t * ../lifvol:

will 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 –r –T –5555 –t *.o lifvol:

will 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 –b *.o lifvol:

All the object files in the current directory are copied to the LIF volume lifvol. Copying mode is BINARY and LIF BINARY files are created.

lifcp –r –t * /lifvol:

All the files in the current directory are copied to the LIF volume lifvol in root directory. Copying mode is RAW and LIF file types are set to BIN.

lifcp abc\\: lifvol:CDE

copy file abc: to LIF file CDE in lifvol.

lifcp –t abc def lifvol:

copy files abc and def to lif files ABC and DEF within lifvol.

lifcp lifvol:ABC .

copy LIF file ABC within lifvol to file ABC within current directory.

lifcp - /dev/fd.0:A_FILE

copy standard input to LIF file A_FILE on LIF volume /dev/fd.0.

lifcp lifvol:ABC /dev/fd.0:CDE
copy LIF file ABC in lifvol to LIF file CDE on /dev/fd.0.

pr abc | lifcp - lifvol:ABC
copy the output of pr to the LIF file ABC.

pr abc | lifcp - lifvol:
copy the output of pr to the LIF volume lifvol. LIF file STDIN is crated since no files names are specified.

lifcp lifvol:ABC -
copy LIF file ABC in lifvol to standard out.

lifcp * ../lifvol:
copy all files within 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!).

SEE ALSO
lif(5), lifinit(1), lifls(1), lifrename(1), lifrm(1).

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

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Lifinit writes a LIF volume header on a volume or file. *Options* may appear in any order. Their meanings are:

-v*nnn* Sets the volume size to *nnn* bytes. If *nnn* is not a multiple of 256, it will be rounded down to the next such multiple.

-d*nnn* Sets the directory size to *nnn* file entries. If *nnn* is not a multiple of 8, it will be 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 upper case letters (A–Z), digits (0–9) and the underscore character (_). The first character (if any) must be a letter. The utility will automatically perform translation to create legal LIF volume names. Therefore, all lower-case letters are up-shifted and all other characters except numeric and underscore will be replaced with capital letter (X). If the volume name does not start with a letter, the volume name will be preceded by the capital letter (X). The volume name will also be right padded with blanks or truncated as needed to be 6 characters long. If **–n** is used with no *string*, the default volume name is set to 6 blanks.

If *file* does not exist, a regular HP–UX disc file is created and initialized.

The default values for volume size are 256K bytes for regular files, and the actual capacity of the device (as determined by *volsize*(3)) 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.

Note that you should **not** mount the special file before using *lifinit*.

HARDWARE DEPENDENCIES

Series 200:

If your media has never been initialized, it must be initialized using *mediainit* before *lifinit* can be used. (Refer to the System Administrator Manual for details concerning *mediainit*.)

Series 500:

You **must** use a character special file to access the media.

If your media has never been initialized, it must be initialized using *sdfinit*(8) before *lifinit* can be used.

EXAMPLES

lifinit -v500000 -d10 x
lifinit /dev/rfd.0

SEE ALSO

lif(5), lifcp(1), lifls(1), lifrename(1), lifrm(1), sdfinit(8).

DIAGNOSTICS

Lifinit returns exit code 0 if the volume is initialized successfully. Otherwise it prints a diagnostic and returns non–zero.

WARNING

Do not terminate *lifinit* once it has started executing. Otherwise, your media could become corrupted.

NAME

lifls - list contents of a LIF directory

SYNOPSIS

lifls [option] name

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Lifls lists the contents of a LIF directory on STDOUT. The default output format calls for the file names to be listed in multiple columns (as is done by *ls*(1), except unsorted) if STDOUT is a character special file. If STDOUT is not a teletype, 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*, then only the file is listed. Following options are available and only one option should be specified at any one time.

-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 STDOUT type.

-L Will return the content of the "last volume flag" in decimal.

-i Will return the content of the "implementation" field in hex.

-v Will return the content of the "volume number" in decimal.

Note that you should **not** mount the special file before using *lifls*.

HARDWARE DEPENDENCIES

Series 500:

You **must** use a character special file to access the media.

EXAMPLES

lifls -l ../TEST/header
lifls -C /dev/rfd.0

SEE ALSO

lif(5), lifcp(1), lifinit(1), lifrename(1), lifrm(1).

DIAGNOSTICS

Lifls returns exit code 0 if the directory was listed successfully. Otherwise it prints a diagnostic and returns non-zero.

NAME

lifrename - rename LIF files

SYNOPSIS

lifrename oldfile newfile

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Oldfile is a full LIF file specifier (see *lif*(5) 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.

Note that you should **not** mount the special file before using *lifrename*.

HARDWARE DEPENDENCIES

Series 500:

You **must** use a character special file to access the media.

EXAMPLES

lifrename liffile:A_FILE B_FILE
lifrename /dev/fd.0:ABC CDE

SEE ALSO

lif(5), lifcp(1), lifinit(1), lifls(1), lifrm(1).

DIAGNOSTICS

Lifrename returns exit code 0 if the file name is changed successfully. Otherwise it prints a diagnostic and returns non-zero.

NAME

lifrm - remove a LIF file

SYNOPSIS

lifrm file1 ... filen

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: HP

DESCRIPTION

Lifrm removes one or more entries from a LIF volume. File name specifiers are as described in *lif*(5).

Note that you should **not** mount the special file before using *lifrm*.

HARDWARE DEPENDENCIES

Series 500:

You **must** use a character special file to access the media.

EXAMPLES

lifrm liffile:MAN
lifrm /dev/rfd.0:F

SEE ALSO

lif(5), lifcp(1), lifinit(1), lifls(1), lifrename(1).

DIAGNOSTICS

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

NAME

line - read one line from user input

SYNOPSIS

line

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

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.

SEE ALSO

read (documented under sh(1)), read(2).

NAME

linkinfo - object file link information utility

SYNOPSIS

linkinfo [[option] ... [file] ...] ...

HP-UX COMPATIBILITY

Level: HP-UX/OPTIONAL

Origin: HP

Remarks: *Linkinfo* is implemented on the Series 500 only.

DESCRIPTION

Linkinfo examines the object files that are part of a program and prints statistics about sizes of the various data areas and symbol table information. *Linkinfo* searches libraries and examines object files in the same fashion as the link editor *ld*. Thus your command line should reflect the same ordering of object files and libraries as it does for the corresponding link.

Linkinfo is intended for developers of large FORTRAN applications who want information about data sizes in order to tune their application for the Series 500 architecture. It prints a file-by-file summary of sizes for code segments and for the D-data and I-data areas (both initialized and uninitialized). There are options for including information about COMMON blocks, linker-generated pointers, and linker symbol entries (again, file-by-file). There is also provision for generating a crude cross-reference of COMMON block usage by file.

Linkinfo options may occur anywhere on the command line after the command name itself. Some options take a modifier immediately following the option letter (e.g. ... **-e** *entryname*). The space between the option and the modifier is optional.

This utility recognizes the following options. Note that a colon indicates that the option takes an argument; the colon itself is **not** a literal, and must not appear when specifying arguments.

-c requests the name and size of COMMON blocks in the input files.

-e: names an alternate entry point for the user program, other than **_main**. The loader calls this alternate entry point at run-time.

-l: abbreviates a library name. *Linkinfo* searches a default set of directories to locate the desired library. These directories are */lib* and */usr/lib*.

The utility searches these directories in the above order, looking for the library **lib***xxx***.a**, where *xxx* is a string of one or more ASCII characters specified as the modifier for the **-l** option. Since *linkinfo* searches a library immediately upon encountering the library's name on the command line, the placement of the **-l** option is significant. A **-l** with no modifier is the same as **-lc**, which causes *linkinfo* to search the standard C library.

-p requests size information on linker-generated pointers.

-s forces inclusion of symbol table size information for each input file.

-u: specifies a name to enter in the symbol table as undefined. This entry appears as an unresolved reference to the command name. You can use it to force loading object information solely from a library.

-x produces a cross-referenced listing of COMMON block usage. This information is saved in the file *xref.out*.

SEE ALSO

ld(1), getopt(1).

DIAGNOSTICS

Linkinfo returns the following exit codes:

0 - no errors
1 - abort (killed by signal)
2 - error during link

NAME

lint - a C program checker/verifier

SYNOPSIS

lint [option] file ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the 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 taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **-c** or the **-o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

Lint will take all the **.c**,**.ln**, and **llib-l***x***.ln** files (specified by **-l***x*) and process 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, the **.ln** and the **llib-l***x***.ln** files are ignored.

Any number of *lint* options may 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* will 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, you can include a lint version of the Math Library **llib-lm.ln** by inserting **-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 multi-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.

-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 lint library produced is the input that is given 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 of the **-x** option is suggested. 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 through the use of "lint comments" (see below).

The **-D**, **-U**, and **-I** options of *cpp*(1) and the **-g**, **-O**, and **-Y** options of *cc*(1) 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*(1) command. Other options are warned about and ignored. The pre–processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like *exit*(2)).

/*VARARGS*n*/
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
turns on 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 will be printed followed by a question mark.

The behavior of the **-c** and the **-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 will print all the inter–file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

HARDWARE DEPENDENCIES

Series 200:

Lint utilizes a special version of the C compiler front end. The size of the internal compiler tables can be adjusted by using the **–N** option. The syntax for this option is described in the Series 200 HARDWARE DEPENDENCIES section of the manual page for *cc*(1).

FILES

cc(1), cpp(1), make(1).

BUGS

exit(2), *longjmp*(3C), and other functions that do not return are not understood; this causes various lies.

NAME

lock - reserve a terminal

SYNOPSIS

lock

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

DESCRIPTION

Lock requests a password from the user, then prints a "locked" message on the terminal and refuses to relinquish the terminal until the password is repeated. If the user forgets the password, he has no other recourse but to login elsewhere and kill the lock process.

NAME

login – sign on

SYNOPSIS

login [name [env-var ...]]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file." (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session. An invalid login name will cause a request for 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.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login. See *dialups*(5) for details on dialup security.

If you have defined no password, you will be diverted into *passwd*(1) to define a password, after which you may attempt to login again. 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 may attempt to login again.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you will be silently disconnected.

After a successful login, the accounting files are updated, the command interpreter (usually *sh*(1)) is determined, and the user and group IDs, group access list, and working directory are initialized. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter as passed to it is – followed by the last component of the interpreter's pathname (i.e., **–sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used. The command interpreter performs its own initialization, and does login initialization if the name by which it is called starts with –.

If *sh*(1) is the command interpreter, it executes the profile files **/etc/profile** and **$HOME/.profile** if they exist. Depending on what these profile files contain, you are notified of mail in your mail file or any messages you may have received since your last login.

If the command name field is "*", then a *chroot*(2) is done to the directory named in the directory field of the entry. At that point *login* is re-executed at the new level which must have its own root structure, including **/etc/login** and **/etc/passwd**.

The basic *environment* (see *environ*(7)) 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 the super-user, PATH is augmented to include **/etc**.

The environment may be expanded or modified by supplying additional arguments to *login,* either at execution time or when *login* requests your login name. The arguments may 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 they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which 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.

The presence of *name* suppresses the **login:** prompt, and uses *name* as the login name.

If **/usr/adm/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 may be viewed using **lastb** (see *last* (1))

If **/etc/securetty** is present, login security is in effect and the super-user may only login successfully on the ttys listed in this file. 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.**

FILES

/etc/utmp	users currently logged in
/etc/wtmp	history of logins, logouts, and date changes
/usr/adm/btmp	history of bad login attempts
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file – defines users, passwords, and primary groups
/etc/profile	system profile (initialization for all users)
$HOME/.profile	personal profile (individual user initialization)
/etc/dialups	lines which require dialup security
/etc/d_passwd	dialup security encrypted passwords
/etc/securetty	list of valid ttys for root login

VARIABLES

HOME The users home directory.

PATH The path to be searched for commands.

SHELL Which command interpreter is being used.

MAIL Where to look for mail.

TZ The current timezone.

xxx User specified named variables.

L*xxx* User specified unnamed variables.

HARDWARE DEPENDENCIES

Multiple groups are not currently supported on Series 500 and Integral PC.

SEE ALSO

last(1), mail(1), newgrp(1), passwd(1), sh(1), su(1), getty(1m), initgroups(3C), dialups(5), group(5), passwd(5), profile(5), utmp(5), environ(7).

DIAGNOSTICS

Login incorrect
if the user name or the password cannot be matched.

No shell, cannot open password file, or *no directory*:
consult your system manager.

Your password has expired. Choose a new one.
if password aging is implemented.

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:
your shell (or /bin/sh if your 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".
if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

You don't have a password. Choose one.
This is the first time you have logged in, and you havn't established a password. See *passwd*(1) for the formation and entry of the password.

NAME

logname - get login name

SYNOPSIS

logname

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

DESCRIPTION

Logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3X), environ(7).

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). 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. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1). Note that the link editor *ld*(1) is capable of multiple passes over an archive in the portable archive format and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar cr library 'lorder *.o | tsort'
```

FILES

*symref, *symdef temporary files

SEE ALSO

ar(1), ld(1), tsort(1).

BUGS

Object files whose names do not end with **.o**, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

lp, cancel - send/cancel requests to an LP line printer

SYNOPSIS

lp [**-c**] [**-d**dest] [**-m**] [**-n**number] [**-o**option] [**-s**] [**-t**title] [**-w**] files
cancel [ids] [printers]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Native Language Support:
8-bit file names, 8-bit and 16-bit data, customs, messages

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name **-** stands for the standard input and may 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.

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*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

-c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will be 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 then *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.

-d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then 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 may 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* may be collected by specifying the **-o** keyletter more than once. For more information about what is valid as *options* for printers supported on your hardware, see the *mklp*(1M) script.

-s Suppress messages from *lp*(1) 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, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently

printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

EXAMPLES

1. Assuming there is an existing Hewlett–Packard 2934A line printer named *lp2*, configured with the **hp2934a** model interface program. This model has the **-c** option which will cause the printer to print in a compressed mode. To obtain compressed print on *lp2*, use the command:

```
lp -dlp2 -oc files
```

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1). accept(1M), lpadmin(1M), lpsched(1M), mklp(1M).

NAME

lpstat - print LP status information

SYNOPSIS

lpstat [options]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System V

Remarks: Not supported on the Integral Personal Computer.

Native Language Support:
8-bit file names and data, customs, messages.

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may 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. For example:

```
-u"user1, user2, user3"
```

The 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*.

-o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

-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.

-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 pathnames of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls, l, ll, lsf, lsr, lsx – list contents of directories

SYNOPSIS

ls [**–RadCxmlLnogrtucpFbqisf1A**] [names]
l [ls options] [names]
ll [ls options] [names]
lsf [ls options] [names]
lsr [ls options] [names]
lsx [ls options] [names]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System V and UCB

Native Language Support:
8–bit filenames.

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 alphabetically by default. 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, network special files, SRM special files, and the contents of these directories and special files.

If you are the super-user, 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, and may also be controlled by option flags. The default format for a teletype is to list the contents of directories in multi-column format, with the entries sorted down the columns. (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. This is because the individual file names may be arbitrarily long.) If the standard output is not a teletype, 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 one 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 are assumed.

There is an unbelievable number of options:

–R Recursively list subdirectories encountered. Network special files and SRM files are treated as directories only up to a fixed recursion limit. This is to prevent infinite loops from developing between two network nodes which are mutually linked. This recursion limit is currently set to 1.

–a List all entries; usually 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 the super-user, this flag defaults to ON, and is turned off by **–A**.

–d If an argument is a directory, network special file, or SRM file, list only its name (not its contents); often used with **–l** to get the status of a directory.

–L If the argument is a symbolic link, list the file or directory link references rather than the link itself. Not all HP-UX systems support symbolic links. See *symlink*(2).

–C Multi-column output with entries sorted down the columns.

–x Multi-column output with entries sorted across rather than down the page.

–m Stream output format.

–l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '->'.

–n The same as **–l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

–o The same as **–l**, except that the group is not printed. (If both **-l** and **-o** are specified, the group is not printed.)

–g The same as **–l**, except that the owner is not printed. (If both **-l** and **-o** are specified, the owner is not printed.)

–1 The file names will be listed in single column format regardless of the output device. This will force single column format to the user's terminal.

–r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

–t Sort by time modified (latest first) instead of by name.

–u Use time of last access instead of last modification for sorting (with the **–t** option) or printing (with the **–l** option).

–c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**–t**) or printing (**–l**).

–p Put a slash (/) after each filename if that file is a directory.

–F Put a slash (/) after each filename if that file is a directory, network special file, or SRM file, and put an asterisk (*) after each filename if that file is executable.

–b Force printing of non-graphic characters to be in the octal \ddd notation.

–q Force printing of non-graphic characters in file names as the character (?).

–i For each file, print the i-number in the first column of the report.

–s Give size in blocks, including indirect blocks, for each entry.

–f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **–l**, **–t**, **–s**, and **–r**, and turns on **–a**; the order is the order in which entries appear in the directory.

Ls normally is known by several names which provide shorthands for the various formats:

l is equivalent to **ls –m**.
ll is equivalent to **ls –l**.
lsf is equivalent to **ls –F**.
lsr is equivalent to **ls –R**.
lsx is 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.

The mode printed under the **–l** option consists of 10 characters that are interpreted as follows:

The first character is:

d if the entry is a directory;
b if the entry is a block special file;
c if the entry is a character special file;

n if the entry is a network special file (LAN);
p if the entry is a fifo (a.k.a. "named pipe") special file; l if the entry is a symbolic link;
– if the entry is an ordinary file.

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 file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
– if the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or –) is **t** if the 1000 (octal) bit of the mode is on; see *chmod*(1) for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

HARDWARE DEPENDENCIES

Series 200:

Network and SRM files are not implemented.

Series 500:

The **–a** and **–A** options perform the same function.

FILES

/etc/passwd	to get user IDs for **ls –l** and **ls –o**.
/etc/group	to get group IDs for **ls –l** and **ls –g**.
/usr/lib/terminfo/?/*	to get terminal information.

SEE ALSO

chmod(1), find(1), symlink(2).

BUGS

Newline and tab are considered printing characters in file names.

The option setting based on whether the output is a teletype is undesirable as **ls –s** is much different than **ls –s | lpr**. On the other hand, not using this setting would make old shell scripts which used *ls* almost inevitably fail.

Unprintable characters in file names may confuse the columnar output options.

NAME

lsdev – list device drivers in the system

SYNOPSIS

/etc/lsdev [major...]

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: This manual page describes *Lsdev* as implemented on the Series 200.

Not supported on the Integral Personal Computer.

DESCRIPTION

With no arguments, *lsdev* lists the major device numbers, for block and character files, and driver names of all device drivers configured into the system and available for invocation via special files. A "-1" in either the block or character column means that a major number does not exist for that type.

If there are any arguments, they must represent major device numbers. The corresponding driver name, if any, will be printed for each argument. Some numbers will return two driver names, one for the block and one for the character.

Lsdev is simply a quick-reference aid. In some implementations, it may only read an internal list of device drivers, not the actual list in the operating system.

SEE ALSO

Section 4.

DIAGNOSTICS

Lists the drivername as "no such driver" when appropriate.

NAME

lsdev – list device drivers in the system

SYNOPSIS

/etc/lsdev [major...]

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: This manual page describes *lsdev* as implemented on the Series 500. not supported on the Integral Personal Computer.

DESCRIPTION

With no arguments, *lsdev* lists, one pair per line, the major device numbers and driver names of all device drivers configured into the system and available for invocation via special files.

If there are any arguments, they must represent major device numbers. For each, *lsdev* lists the corresponding driver name (if any).

Lsdev is simply a quick-reference aid. In some implementations, it may only read an internal list of device drivers, not the actual list in the operating system.

SEE ALSO

Section 4.

DIAGNOSTICS

Lists the drivername as "no such driver" when appropriate.

Permuted Index

HP Part Number
09000-90008
Printed in U.S.A. 6/86

For Internal Use Only