HEWLETT PACKARD

HP-UX Reference

HP-UX

HP-UX

HP-UX

HP-UX

HP-UX

HP-UX

HP-UX Reference

Part No. 09000-90004

Hewlett-Packard Engineering Systems Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Hewlett-Packard Engineering Systems Division
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 note at the bottom of the page. A vertical bar in the margin indicates the technical changes on each page. Non-technical changes (such as typographical errors) have no vertical bar. Note that pages which are rearranged due to changes on a previous page are not considered updated.

Reprints incorporate all updates since the last edition.

The printing date changes on reprints and new editions.

First Edition...June 1983

Introduction

This manual describes the features of HP-UX in an alphabetical reference format. It is written for the user who is already familiar with UNIX or UNIX-like operating systems (UNIX is a trademark of Bell Telephone Laboratories, Inc.). The manual is intended for referencing specific details concerning the HP-UX operating system.

For a general overview of HP-UX, see the supplied tutorial text by Jean Yates. For details of the implementation and maintenance of the system, see the *HP-UX System Administrator* manual.

Manual Format

This manual is divided into nine sections, some containing sub-classes that are interspersed throughout the section:

1. Commands and Application Programs:
   - 1. General-Purpose Commands.
   - 1C. Communications Commands.
   - 1G. Graphics Commands.
2. System Calls.
3. Subroutines:
   - 3C. C Library Routines.
   - 3M. Mathematical Library Routines.
   - 3S. Standard I/O Library Routines.
   - 3X. Miscellaneous Routines.
4. Special Files.
5. File Formats.
6. Games.
7. Miscellaneous Facilities.
8. System Maintenance Procedures.
9. Glossary.

Section 1 (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to system calls (section 2) or subroutines (section 3), which are intended to be called by the user's programs. Commands generally reside in the directory */bin* (for *bin*ary programs). Some programs also 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)). Sub-class 1C contains communication programs such as *cu*, *fget*, etc. These entries may differ from system to system. A few commands are also located in */lib* and */usr/lib*.

Section 2 (*System Calls*) describes the entries into the HP-UX kernel, including the C language interface.

Section 3 (*Subroutines*) describes the available subroutines. 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 in which they are stored.

Section 4 (*Special Files*) discusses the characteristics of each special file (device driver) that actually refers to an input/output device. The names in this section generally refer to Hewlett-Packard's device names for the hardware, rather than to the names of the special files themselves.

Section 5 (*File Formats*) documents the structure of particular kinds of files. For example, the format of the output of the link editor is given in *a.out*(5). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories */usr/include* and */usr/include/sys.*

Section 6 (*Games*) describes the games and educational programs that, as a rule, reside in the directory */usr/games.* This section may or may not exist, depending on whether or not games are supported in each implementation of HP-UX.

Section 7 (*Miscellaneous Facilities*) contains a variety of things. Included are descriptions of character sets, macro packages, etc.

Section 8 (*System Maintenance Procedures*) discusses those commands which are useful for crash recovery and booting the system, plus commands used to perform system integrity checks and other maintenance procedures. Information in this section is mostly of interest to the super-user.

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

Each section (except 9) consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

NAME

gives the name(s) of the entry and briefly states its purpose.

SYNOPSIS

summarizes the use of the entry (program) being described. A few conventions are used:

Boldface strings are literals and are to be typed just as they appear.

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. When an argument name is given as "name" or "file", it always refers to a file name.

Ellipses (...) are used to show that the previous argument may be repeated. u .if t .nr CI -.8iu 3m

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 whose names begin with −, +, or =.

HP-UX COMPATIBILITY
shows the entry's HP-UX level and its origin, according to the HP-UX Compatibility Model (see *HP-UX Compatibility Model* following this introduction). This section also shows whether an optional HP software package is required.

DESCRIPTION
discusses the function and behavior of each entry.

HARDWARE DEPENDENCIES
points out variations from HP-UX standard due to the specific hardware involved.

EXAMPLE(S)
gives example(s) of usage, where appropriate.

FILES
gives the file names that are built into the program.

RETURN VALUE
discusses the meaning of values which are returned by the program.

SEE ALSO
gives pointers to related information.

DIAGNOSTICS
discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

WARNINGS
points out potential pitfalls.

BUGS
gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permuted index precede Section 1. On each index line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

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 by Jean Yates for a 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 manager of your system. Keep pressing the "break" or "del" key until the **login:** message appears.

When a connection has been established, the system types **login:** and you then type your user name followed by 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 your login name in lower case if possible; if you type upper-case letters, HP-UX will assume 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 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 a terminal, the character @ "kills" all the 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 passed as input 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 the Series 500/600/700). 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 your 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 manager gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is assumed to be in that directory by default. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. The permissions you have in other directories and files will have been granted or denied to you by their respective owners, or by the system manager. 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 your 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 a 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)), 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 the 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 logged in, *write*(1) is used; *mail*(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 two 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 Bell System III 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 University of California at Berkeley.

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 Bell, 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 Bell 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 standard HP-UX, or never will be.

NAME

intro – introduction to commands

SYNOPSIS

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.
HP	means written by HP.
UCB	means derived from U. C. Berkeley 4.2bsd.
V7	means included for UNIX Version 7 compatibility (and not in Bell System III).

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.

The **NAME** line is used to drive the permuted index. To make the permuted index more useful, some redundancy is introduced into the one-line summary. This means that some index entries are inaccurate on technicalities, but useful for indexing.

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), getopt(3C).
How to Get Started, 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 some 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.

NAME

admin – create and administer SCCS files

SYNOPSIS

admin [**–n**] [**–i**[name]] [**–r**rel] [**–t**[name]] [**–f**flag[flagval]] [**–d**flag[flagval]] [**–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 doesn't 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 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.

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.

–*elogin* 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.}f HH:MM:SS.}f* 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 check-sum. 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).
Source Code Control System User's Guide in *HP-UX: Selected Articles.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar – archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: System III

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 archive file format is consistent across all HP-UX implementations. It is only useful to port source archives. Individual files are inserted without conversion into the archive file.

Key must be present, and is one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcl**. *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 file. If the optional character **u** is used with **r**, then only those files with modified dates 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 files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end. *Ar* will create *afile* if it does not already exist. If there are no file *names*, *ar* may create an empty archive file whose only contents is its magic number.

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. 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, only those files are tabled.

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.

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. When used with **t**, it gives a long listing of all information about the files from the archive headers. 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. Normally *ar* places its temporary files in the directory /**tmp**. This option causes them to be placed in the current working directory. Only the **d**, **m**, and **r** options use temporary files.

Only the following combinations are meaningful:

d:	**v**, **l**
r:	**u**, **v**, **c**, **l**, and **a** ∣ **b** ∣ **i**
q:	**v**, **c**
t:	**v**

p:	**v**
m:	**v**, **l**, and **a** ǀ **b** ǀ **i**
x:	**v**

Other combinations have no effect.

FILES

/tmp/v* temporary files

SEE ALSO

arcv(1), ld(1), lorder(1), ranlib(1), ar(5).

WARNING

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

BUGS

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

Ar accepts the letter **n** as a modifier, but it has no effect.

NAME
asa – interpret ASA carriage control characters

SYNOPSIS
asa [files]

HP-UX COMPATIBILITY
Level: HP-UX/NUCLEUS

Origin: System V

Remarks: *Asa* is in a preliminary state, and could change in the future.

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

```
a.out | asa | lpr
```

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

```
asa file
```

SEE ALSO
fc(1).

NAME

at - execute commands at a later time

SYNOPSIS

at time [day] [file]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	Version 7

DESCRIPTION

At stores a copy of the named *file* (standard input default) to be used as input to *sh*(1) at a specified later time. *File* must be a shell script. A *cd*(1) command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the user and group ID of the creator of the copied file.

Time is one to four digits, with an optionally appended 'A', 'P', 'N', or 'M', standing for AM, PM, noon, or midnight, respectively. One and two digit numbers are taken to be hours; three and four digit numbers are taken to be hours and minutes. If none of the above-mentioned letters follow the digits, a 24 hour clock time is assumed.

Day is either a month name followed by a day number, or a day of the week. If the word 'week' follows *day*, invocation is moved seven days further off. Three-letter abbreviations for month and day names may be used. Examples of legal commands are

```
at 8am jan 24 scriptfile1
at 1530 fri week scriptfile2
```

At programs are executed by periodic execution of the command */usr/lib/atrun* from *cron*(8). The time interval accuracy of *at* depends upon how often *atrun* is executed.

Standard output or error output is lost unless redirected.

FILES

/usr/spool/at/*yy.ddd.hhhh.uu*
: activity to be performed at hour *hhhh* of day *ddd* of year *yy*. *uu* is a unique number.

/usr/spool/at/lasttimedone
: contains *hhhh* for last hour that *atrun* was executed.

/usr/spool/at/past
: directory of activities now in progress.

/usr/lib/atrun
: program that executes activites that are due.

pwd(1)

SEE ALSO

calendar(1), cron(8).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

BUGS

Due to the time interval accuracy of the execution of */usr/lib/atrun*, there may be bugs in scheduling things almost exactly 24 hours into the future.

NAME

awk – text pattern scanning and processing language

SYNOPSIS

awk [–Fc] [prog] [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.

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 (").

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 }
```

SEE ALSO

grep(1), lex(1), sed(1).
Awk–A Pattern Scanning and Processing Language in *HP-UX: Selected Articles.*

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:	System III Compatibility - HP-UX/STANDARD
Origin:	System III

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.

NAME

basename, dirname – extract portions of path names

SYNOPSIS

basename string [suffix]
dirname string

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

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.

Dirname delivers all but the last level of the path name in *string*.

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‘
```

SEE ALSO

sh(1).

NAME

bdiff – big diff

SYNOPSIS

bdiff file1 file2 [n] [–**s**]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

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

calendar – reminder service

SYNOPSIS

calendar [–]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

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 "Dec. 7," "december 7," "12/7," etc., are recognized, but not "7 December" or "7/12". On weekends, "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in his login directory and sends him any positive results by *mail*(1). Normally this is done daily in the early morning hours under control of *cron*(8).

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(8).

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**] file . . .

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

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. Output is buffered in 512-byte blocks unless the **–u** (unbuffered) option is specified in which case output is character-by-character.

The **–s** option makes *cat* silent about non-existent files, identical input and output, and write errors. No input file may be the same as the output file unless it is a special file.

SEE ALSO

cp(1), pr(1).

NAME

cb – C program beautifier, formatter

SYNOPSIS

cb [file]

HP-UX COMPATIBILITY

Level:	C-Compiler - HP-UX/EXTENDED
Origin:	System III

DESCRIPTION

Cb places a copy of the C program in *file* (standard input if *file* is not given) on the standard output with spacing and indentation that displays the structure of the program.

NAME

cc - C compiler

SYNOPSIS

cc [option] ... file ...

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: System III

DESCRIPTION

Cc is the HP-UX command file which invokes the 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**. The **.o** file is normally deleted, however, if a single C program is compiled and loaded all at one go.

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

The following options are interpreted by *cc*. Options may not be concatenated. See *ld*(1) for link editor options.

–c Suppress the link edit phase of the compilation, and force an object (**.o**) file to be produced even if only one program is compiled. Produces a **.o** file for each **.c** file.

–p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one which 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).

–g Cause the compiler to generate additional information needed for the use of a symbolic debugger.

–O Invoke an object-code optimizer.

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

–E Run only the macro preprocessor on the named C programs, and send the result to the standard output. The result is compatible with the */lib/ccom* step of *cc*.

–B*string* Find substitute compiler passes in the files named *string* with the suffixes **cpp**, **ccom**, **c1** and **c2**. *String* must be specified for **–B** to be meaningful.

–t[p012] Find only the designated compiler passes in the files whose names are constructed by a **–B** option. In the absence of a **–B** option, the *string* is taken to be /**lib**/**n**. Any or all of the pass designators **p**, **0**, **1**, or **2** may be specified, with the following meanings:

p – preprocessor;
0 – first pass of C compiler;
1 – second pass of C compiler (if any);
2 – optimizer.

–C
–D*name = def*
–D*name*
–I*dir*
–P
–U*name*

These options are passed through to the C preprocessor, *cpp*. Refer to *cpp*(1) for details.

Other arguments are taken to be either link editor option arguments, or 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.

char
The **char** type is treated as signed by default. It may be declared **unsigned**.

pointers
Dereference of a NULL (zero) pointer is illegal and may cause a SIGSEGV error. This applies whether the access is for reading or writing. Some implementations may not be able to detect this error, in this case the result of such an access is undefined. Programs which rely on being able to dereference a null pointer are not considered portable within HP-UX.

identifiers
Identifiers are significant up to (at least) 255 characters. Whether or not longer identifiers are handled is machine dependent. The assembler and loader must also support long identifiers to 255 characters.

HARDWARE DEPENDENCIES

Series 210:

The **–g** option is not currently supported.

An additional option, **–XE**, is supported. 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 –**p** and –**g** options are not currently supported.

An additional option, –**v**, is supported. This option enables verbose mode, which produces a step-by-step description of the compilation process.

The *ld* options **p** and **v** conflict with *cc* options, and thus cannot be accessed via *cc*.

The –**B** option is supported, but no substitute compiler passes are provided.

Cc currently supports 16-character identifiers, and 15-character external names.

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

If a C program compiles such that branches of greater than 256 Kbytes in either direction are generated in the assembler, "impossible reach" errors are given.

FILES

file.c	input file
file.o	object file
a.out	linked output
/tmp/ctm*	temporary
/lib/cpp	preprocessor
/lib/ccom	compiler
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	alternate startoff for profiling

/lib/libc.a	standard library, see section 3 of this manual
/usr/include	standard directory for **#include** files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978.
as(1), cpp(1), ld(1), prof(1), monitor(3C).

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.

NAME
cd – change working directory

SYNOPSIS
cd [directory]

HP-UX COMPATIBILITY
Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION
If specified, *directory* becomes the new working directory; otherwise, the value of the shell parameter **$HOME** is used. The process 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 executed by the shell.

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

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 *Source Code Control System User's Guide.* 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).
Source Code Control System User's Guide in *HP-UX: Selected Articles.*

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

chatr – change program's internal attributes

SYNOPSIS

/lbin/chatr [+**c**|**–c**] [+**h**|**–h**] [**–m**n] [+**n**|**–n**] [+**p**|**–p**] [**–s**] [+**z**|**–z**] file ...

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Chatr* is implemented on the Series 500/600/700 only.

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 the virtual bit for each code segment.
–c	clear the virtual bit for each code segment.
+**h**	set the virtual bit for the heap of a two data segment program.
–h	clear the virtual bit for the heap of a two data segment program.
–mn	change the maximum heap size to *n* bytes.
+**n**	mark code as shareable (magic number = SHARE_MAGIC).
–n	mark code as unshareable (magic number = EXEC_MAGIC).
+**p**	set the paged and virtual bits for the heap of a two data segment program.
–p	clear the paged and virtual bits for the heap of a two data segment program.
–s	perform action silently.
+**z**	set the demand load bit for each segment.
–z	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.

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** 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 III

DESCRIPTION

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**.

Only the owner of a file (or the super-user) may change its mode.

EXAMPLES

The first example denies write permission to others, the second makes a file executable:

```
chmod o–w file

chmod +x 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 III

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.

FILES

/etc/passwd
/etc/group

SEE ALSO

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

NAME

chroot – change root directory for a command

SYNOPSIS

chroot newroot command

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

```
chroot newroot command >x
```

will create the file **x** relative to the original root, not the new one.

Command includes both the command name and any arguments.

This command is restricted to the super-user.

Chroot does not search **PATH** for the location of *command*, so the absolute path name of *command* must be given.

The new root path name is always relative to the current root. Even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

SEE ALSO

chdir(2).

BUGS

Command cannot be in a shell script.

NAME

cmp – compare two files

SYNOPSIS

cmp [**–l**] [**–s**] file1 file2

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

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.

–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 III

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 will remove backspaces in favor of multiply overstruck lines. *Col* is particularly useful for filtering multi-column 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** (\017) and **SI** (\016) 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).

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/reject common lines of two files

SYNOPSIS
comm [– [**123**]] file1 file2

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

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

cp, ln, mv – copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target
ln [–f] file1 [file2 ...] target
mv [–f] file1 [file2 ...] target

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same. 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.

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.

Ln and *mv* will ask for permission if *target* already exists and is not writable. This is done by printing the mode (see *chmod*(2)), and reading one line from the standard input (if the standard input is a terminal). If the line (which you type in) begins with y, the operation will take place. Any other response will abort it. The –f option will force these operations to occur without your intervention.

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 file whose name ends in slash (/);
>
> 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(1), 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 name becomes that of the copying process and any linking relationship with other files is lost.

Ln will not link across file systems.

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio –o [**acBvx**]

cpio –i [**BcdmPrstuvx6**] [patterns]

cpio –p [**adlmuvx**] directory

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

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.

Cpio –i (copy in) extracts from the standard input (which is assumed to be the product of a previous **cpio –o**) the names of files selected by zero or more *patterns* given in the name-generating notation of *sh*(1). In *patterns*, metacharacters **?**, *****, and **[**...**]** match the slash / character. Multiple *patterns* may be specified. If no *patterns* are specified, the default 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.

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.

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 5120 bytes to the record (does not apply to the *pass* option; recommended only with data directed to or from /**dev**/**rmt?**).
d	*Directories* are to be created as needed.
c	Write *header* information in ASCII character form for portability.
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). Only bytes contained in the header are swapped. Non-ascii files will probably need further processing to be readable; this processing requires knowledge of the content of the file and thus cannot be done by this program. (**PDP–11** and **VAX** are registered trademarks of Digital Equipment Corporation).
r	Interactively *rename* files. If the user types a null line, the file is skipped.
s	Identical to the **P** option, except that all bytes in the file are swapped (including the header).
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. Usable only with the **–p** option.
m	Retain previous file modification time. This option is ineffective on directories that are being copied.
6	Process an old (i.e., UNIX *Sixth* Edition format) file. Only useful with **–i** (copy in).

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

Series 500/600/700:

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.

The **–B** option must be used when writing to the HP88140 (mag tape cartridge). Note that the **–B** option must not be used when performing raw I/O to an HP9130K (miniature flexible disc drive), if the I/O requires more than one volume.

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls  cpio –o >/dev/mt0

cd olddir
find . –print  cpio –pdl newdir
```

The trivial case "find . –print cpio –oB >/dev/rmt0" can be handled more efficiently by:

```
find . –cpio /dev/rmt0
```

SEE ALSO

ar(1), find(1), tar(1), cpio(5).

BUGS

Path names are restricted to 128 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.

NAME

cpp – C language preprocessor

SYNOPSIS

/lib/cpp [option ...] [ifile [ofile]]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. 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.

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 *option*s 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, vax
UNIX System variant:	RES, RT, TS, PWB

–D*name*
–D*name* = *def*
Define *name* as if by a **#define** directive. If no = *def* is given, *name* is defined as 1.

–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 *ifile* argument, then in directories named in **–I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in < >, the directory of the *ifile* argument is not searched.

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 #. 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*
Replace subsequent instances of *name*, followed by (, a list of comma separated tokens, and a), by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list. Note that there must be no space between *name* and (.

#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*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **–I** option above for more detail.

#line *integer-constant* "*filename*"
: Cause *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 from which it comes. If "*filename*" is not given, the current file name is unchanged.

#endif Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#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**, or if it is a currently defined reserved symbol.

#ifndef *name*
: The lines following will not appear in the output if and only if *name* has been the subject of 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.

FILES

/usr/include	standard directory for **#include** files

SEE ALSO

cc(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.

NAME
crypt – encode/decode files

SYNOPSIS
crypt [password]

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or clear text can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES
/dev/tty	for typed key

SEE ALSO
ed(1), makekey(8).

BUGS
If output is piped to *nroff*(1) and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)).

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut –**c**list [file1 file2 ...]
cut –**f**list [–**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.

SEE ALSO

grep(1), paste(1).

DIAGNOSTICS

line too long	A line can have no more than 511 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.

NAME

date – print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion 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 the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by **%** and will be replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output without change. The string is always terminated with a new-line 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 – Sun to Sat
h	abbreviated month – Jan to Dec
r	time in AM/PM notation

HARDWARE DEPENDENCIES

Series 500/600/700:

The file /dev/kmem is not used.

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'
```

would generate as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

FILES

/dev/kmem
/usr/adm/wtmp

SEE ALSO

ctime(3C).

DIAGNOSTICS

No permission	if you aren't the super-user and you try to change the date;
bad conversion	if the date set is syntactically incorrect;
bad format character	if the field descriptor is not recognizable.

NAME

dd – convert, reblock, translate, and copy a (tape) file

SYNOPSIS

dd [option = value] ...

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

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 records before starting copy.
seek = *n*	seek *n* records from beginning of output file before copying.
count = *n*	copy only *n* input records.
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 record 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 record 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 record into the ASCII file **x**:

```
dd if=/dev/rmt0 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 record sizes.

SEE ALSO

cp(1), tr(1).

DIAGNOSTICS

f + p records in(out)	numbers of full and partial records read(written)

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 [–rSID] [–**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.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
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*.

SEE ALSO

admin(1), bdiff(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide in *HP-UX: Selected Articles.*

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

Lines beginning with an **SOH** ASCII character (binary 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.

NAME

deroff – remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [–**w**] [–**m**x] [files]

HP-UX COMPATIBILITY

Level: Text processing - HP-UX/EXTENDED

Origin: System III

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, 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 resulting –**mm** or –**ms** option causes the **mm** or **ms** macros to 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), tbl(1), troff(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
df – report number of free disk blocks

SYNOPSIS
df [**–t**] [**–f**] [file-systems]

HP-UX COMPATIBILITY
Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION
Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name or by mounted directory name. If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

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, *df* will report on raw devices.

HARDWARE DEPENDENCIES
Series 500/600/700:
Df cannot report on unmounted raw devices.

FILES
/dev/HP79*
/dev/HP91*
/dev/HP98*
/dev/HP829*1
/etc/mnttab

SEE ALSO
fs(5), mnttab(5), fsck(8).

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 III

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 **–b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **–e** option produces a script of *a, c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **–f** option produces a similar script, not useful with *ed*, in the opposite order. 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.

Option **–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**.

FILES

/tmp/d?????
/usr/lib/diffh for **–h**

SEE ALSO

cmp(1), comm(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 (**.**).

The specified files must contain ASCII text in the same format as that produced by the HP-UX editors (*ed, ex, vi*). That is, all lines (including the last) must end with a new-line. *Diff* will not recognize a final line if the terminating new-line is not there.

NAME

diffmk – mark differences between files

SYNOPSIS

diffmk name1 name2 name3

HP-UX COMPATIBILITY

Level:	Text Processing - HP-UX/STANDARD
Origin:	System III

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, he can use *diffmk* 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).

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.

NAME

dircmp – directory difference comparison

SYNOPSIS

dircmp dir1 dir2

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

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 in addition to a list that indicates whether the files common to both directories have the same contents.

SEE ALSO

cmp(1), diff(1).

NAME

du – summarize disk usage

SYNOPSIS

du [–**ars**] [names]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Du gives the number of 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 III

DESCRIPTION
Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

b	backspace
c	print line without new-line
f	form-feed
n	new-line
r	carriage return
t	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.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO
sh(1).

NAME

ed – text editor

SYNOPSIS

ed [–] [**–x**] [file]

HP-UX COMPATIBILITY

Level:	HP-UX/DEVELOPMENT
Origin:	System III

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*. If **–x** is present, an *x* command is simulated first to handle an encrypted file. *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.

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 **p** or by **l**, in which case the current line is either printed or listed, respectively, as discussed below under the *p* and *l* 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.

(.)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* begins with **!**, 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.

(. , . + 1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If only 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* begins with **!**, 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.

(. , .)**s**/*RE*/*replacement*/ or
(. , .)**s**/*RE*/*replacement*/**g**

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. 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 *w*rite 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* begins with **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is written to the specified file, or to the currently-remembered file. Such a shell command is *not* remembered as the current file name.

X

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

(**$**)=

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. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

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?

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

SEE ALSO

awk(1), crypt(1), ex(1), grep(1), sed(1), sh(1), vi(1).
The ed Editor, in *HP-UX: Selected Articles.*

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 any character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.
Because 0 is an illegal address for the *w* command, it is not possible to create an empty file with *ed*.

NAME

env – set environment for command execution

SYNOPSIS

env [–] [name = value] ... [command args]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

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/STANDARD
Origin:	HP
Remarks:	*Err* is implemented on the Series 500/600/700 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, edit, e, expreserve, exrecover – text editor commands

SYNOPSIS

ex [–] [–**v**] [–**t** *tag*] [–**r**] [+ *command*] [–**l**] *name* ...
edit [ex options]
e [ex options]
/**usr**/**lib**/**expreserve**
/**usr**/**lib**/**exrecover**

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

Remarks: The *ex*/*vi* editor is based on software developed by the University of California at Berkeley Computer Science Division, Department of Electrical Engineering and Computer Science, and such software is owned and licensed by the Regents of the University of California.

DESCRIPTION

Ex is the root of a family of editors: *edit*, *ex*, *e*, *vi*, and *view*. *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 not used *ed*, or are a casual user, you will find that the editor *edit* is convenient for you. It avoids some of the complexities of *ex*, which is used mostly by systems programmers and persons very familiar with *ed*.

If you have a CRT terminal, you may wish to use a display based editor. If so, see *vi*(1), which is a command that focuses on the display editing portion of *ex*.

Expreserve saves the temporary files created during a *vi*, *ex*, *edit*, or *e* editing session that are left after a crash or editor abort. It looks in /**tmp** and moves all such files to /**usr**/**preserve**. Do not run *expreserve* while an editor is currently being used. *Expreserve* is normally invoked automatically by *rc*(8).

Exrecover is used by any of the above-mentioned editors when invoked with the –**r** option. It is rarely used in a stand-alone fashion.

HARDWARE DEPENDENCIES

Series 500/600/700:

Under **FILES**, the following changes occur:

/**usr**/**lib**/**ex?.?strings** does not exist; error messages are compiled directly into the *ex* code;
the files /**usr**/**lib**/**ex?.?recover** and /**usr**/**lib**/**ex?.?preserve** are renamed /**usr**/**lib**/**exrecover** and /**usr**/**lib**/**expreserve**, respectively.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), vi(1), environ(5), termcap(5).
Edit: A Tutorial, in *HP-UX: Selected Articles*
Ex Reference Manual – Version 3.5, in *HP-UX: Selected Articles*
The ed Editor, in *HP-UX: Selected Articles*
The vi Editor, in *HP-UX: Selected Articles*

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 screenful 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/STANDARD
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 III

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 "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

fc – FORTRAN 77 compiler

SYNOPSIS

fc [options] files

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP

DESCRIPTION

Fc is the HP-UX FORTRAN 77 compiler. It accepts two types of file arguments:

(1) Arguments whose names end with **.f** are taken to be FORTRAN 77 source programs. They are compiled, and each object program 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.)

(2) Arguments whose names end with **.o** are passed on to the linker (*ld*(1)) to be linked into the final program.

The following options are recognized:

–c	suppress linking and produce object (**.o**) files from source files;
–C	enable range checking (same as **$OPTION RANGE ON**);
–e	write errors to *stderr*;
–I2	make default size of integers and logicals half a word (same as **$OPTION SHORT**);
–L	write a program listing to *stdout* during compilation;
–o outfile	name the output file from the linker *outfile* instead of *a.out*;
–onetrip	execute any DO loop at least once;
–Q dfile	specify *dfile* as the option file;
–s	issue warnings for non-ANSI features (same as **$OPTION ANSI ON**).
–u	force types of identifiers to be implicitly undeclared (same as specifying **IMPLICIT NONE**; no other **IMPLICIT** statements are permitted);
–U	use upper case for external names (default is lower case);
–v	write expanded compiler and linker runstrings to *stderr*;
–w	suppress warning messages (same as **$OPTION WARNINGS OFF**).

Fc also recognizes the following options, but they have no effect if they are encountered. This is for compatibility with other FORTRAN systems.

–m, **–f**, **–p**, **–w66**, **–S**, **–O**, **–E**, **–R**, **–F**

A warning is written to *stderr* if any of these options is used.

Any other options are taken to be arguments to *ld*, and are passed along to the linker.

HARDWARE DEPENDENCIES

Series 500/600/700:

Because the **–s** option is recognized by the compiler, it is not possible to pass the strip option to the linker. Programs which are to be stripped may be compiled with the **–c** option, then linked separately.

FILES

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/mainf.o	main program
/lib/libI77.a	FORTRAN I/O library
/lib/libF77.a	FORTRAN run-time library
/usr/tmp/*	temporary files used by the compiler; names are created by *tmpnam*(3S).

SEE ALSO

FORTRAN/9000 Language Reference Manual, HP Part No. 97081–90001; *Structured FORTRAN 77*, by Sidney 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*. 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.

NAME

find – find files

SYNOPSIS

find path-name-list expression

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

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**, 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 (512 bytes per block).
–atime *n*	True if the file has been accessed in *n* days.
–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*	Write the current file on *device* in *cpio* (5) format (5120 byte records).
–newer *file*	True if the current file has been modified more recently than the argument *file*.
(*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	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).

EXAMPLE

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), cpio(5), fs(5).

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 COMPATIBILITY

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 (stderr) 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".

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 UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX 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(5).
Source Code Control System User's Guide in *HP-UX: Selected Articles*.

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.

NAME

getopt – parse command options

SYNOPSIS

`**getopt** optstring args`

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

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. *Getopt* will place -- in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments ($1 $2 . . .) are reset so that each option is preceded by a – and placed in its own shell argument. Each option argument is also placed in its own shell argument.

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** and **b**, and 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; shift;;
        --)         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*.

BUGS

The output of *getopt* cannot be more than 256 characters.

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 III

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.
–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.

BUGS

Lines are limited to 256 characters; longer lines are truncated.
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.

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., **ge6**, for message 6 from the *get* command).
type 2	Does not contain numerics (as a command, such as **get**)
type 3	Is all numeric (e.g., **212**)

The response of the program 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.

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/RUN ONLY
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

gethostname(2), sethostname(2), uname(2).

NAME

id – print user, group IDs and names

SYNOPSIS

id

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

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

join – relational database operator

SYNOPSIS

join [options] file1 file2

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

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*.

Fields are normally separated by blank, tab or new-line. In this case, multiple separators count as one, and leading separators are discarded.

These 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.
–**o** *list*	Each output line comprises the fields specifed in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number.
–**t***c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

SEE ALSO

awk(1), comm(1), sort(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.

NAME
kill – terminate a process

SYNOPSIS
kill [–signo] processid ...

HP-UX COMPATIBILITY
Level: HP-UX/NUCLEUS

Origin: System III

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

ld – link editor

SYNOPSIS

ld [[option] ... [file] ...] ...

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	HP
Remarks:	This manual page describes *ld* as implemented on the Series 500/600/700. Refer to other *ld* manual pages for information valid for other implementations.

DESCRIPTION

Ld combines several object files into a final executable program, resolves external references (such as procedure calls and accesses of COMMON), and searches libraries (in the format generated by *ar*(1)). In the simplest case, *ld* combines several named object files (those ending in **.o**) to produce an object module loadable by the HP-UX program loader, *exec*. An alternative is to generate output that is suitable as input for future re-linking (see **–r** below). By default, *ld* leaves its output in the file **a.out**. The **–o** option overrides this. The linker will not mark the output file as executable if any errors occur during its operation.

Ld processes files (and libraries) in the same order that they appear on the command line. If an argument file is an archive, the linker searches it once, sequentially, at that point in the argument list. The linker keeps track of unresolved external references in the user's program, and includes a library element with the user's program if it defines a needed symbol. (The linker includes library code as it scans the library). Order is important here: the linker will only include a definition from a library if it has already seen an unresolved reference to that name.

Ld recognizes the special library format generated by *ranlib*(1). The first element of such a library is a directory of all the external symbols defined by the remaining archive elements. The linker normally scans a library only once, even if that scan produces new unresolved references to previously ignored elements. Once a library has been "randomized", however, *ld* iteratively searches the library as long as there are unsatisfied references to objects in that library. If a directory for a library is out of date, *ld* treats the library as a normal (sequential) one.

The linker recognizes several names as having special meanings. By default, **_main** (**main** in C) is the main entry point for a user program. The user can override this with the **–e** option. In addition, the names **_end**, **_edata**, and **_etext** (**end**, **edata**, and **etext** in C) are predefined (see section 3). User programs cannot define these names for external use. In addition, the name **_start** (**start** in C) is reserved if you link via *cc*. *Cc* links programs to the clean-up file **crt0.o** and arranges for the entry point to be **_start** in that file. Thus, if you use *cc* to link a program, you cannot define "start" as an external symbol in your C program.

Ld options may occur anywhere on the command line, as long as they follow the command name itself. Some options expect a modifier immediately following the option letter (e.g. ... **–o** *outname*). *Ld* recognizes modifiers either as part of the word containing the option letter, or as a separate word following the option letter (as with *getopt*(3)).

Ld recognizes the following options. Note that, in the following options, a colon indicates that an argument is accepted. The colon itself is *not* literal, and must not appear when specifying arguments.

–A put the D-data and I-data areas apart (in separate segments).

–T keep the D-data and I-data areas together in the same global data segment (GDS).

–V: assign a 32-bit integer version number to the resulting output file.

–N mark the program as not shareable.

–d force definition of COMMON storage (i.e. assign addresses and sizes), even for **–r** output.

–e: names an alternate entry point for the user program, other than **_main**. The loader calls this alternate entry point at run-time.

–l: is an abbreviation for a library name. *Ld* searches a default set of directories to locate the desired library. These directories are:

/lib
/usr/lib

The linker 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 *ld* searches a library when it encounters its name on the command line, the placement of the **–l** option is significant. Note that, for a sequential library, it may be necessary to specify it several times on the command line in order to resolve all references, including those stemming from use of code in the library. A **–l** with no modifier is the same as **–lc**, which causes *ld* to search the standard C library.

–n mark the program as shared (magic number of 108 hexadecimal).

–o: specify a name for the resulting output object file produced by the linker (the default is **a.out**).

–r arranges for the output file to be re-linkable. *Ld* preserves all symbol table and relocation information necessary for linking. In this case the linker does not make final definitions of COMMON, nor does it issue "undefined symbol" messages.

–s tells the linker to strip the final output file; that is, the symbol table and relocation information is discarded. Note that this impairs the effectiveness of your system's debugger (if any). The linker ignores this option if there are any undefined symbols. You can use *strip*(1) to remove this information from an existing executable file.

–u: specifies a name to enter in the linker's symbol table as undefined. This appears as an unresolved reference to the linker. Thus, you can force loading object information solely from a library.

–v causes the linker to output verbose status information showing the progress of the link. It prints the name of each linker input file and archive element that it scans as it encounters them. This information appears twice, once for each pass of linker processing. This option is not available via *cc*(1) because it conflicts with the compiler's verbose option.

–z mark each segment of the user program as demand-loadable. The loader sets up the program at run-time so that segments are loaded only as they are referenced.

SEE ALSO

ar(1), chatr(1), getopt(1), nm(1), ranlib(1), strip(1), end(3), a.out(5).

DIAGNOSTICS

Ld returns the following exit codes:

0 – no errors;
1 – abort (killed by signal);
2 – error during link;
3 – denotes a debugging condition;

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* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

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.

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;
                }
        }
```

The external names generated by *lex* all begin with the prefix **yy** or **YY**.

The flags must appear before any files. The flag **–r** indicates RATFOR actions, **–c** indicates C actions and is the default, **–t** causes the **lex.yy.c** program to be written instead to standard output, **–v** provides a one-line summary of statistics of the machine generated, **–n** will not print out the – summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n*	number of positions is *n* (default 2000)
%n *n*	number of states is *n* (500)
%t *n*	number of parse tree nodes is *n* (1000)
%a *n*	number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **–v** option, unless the **–n** option is used.

SEE ALSO

yacc(1).
LEX – Lexical Analyzer Generator, in *HP-UX: Selected Articles.*

BUGS

The **–r** option is not yet fully operational.

NAME

LIF – Logical Interchange Format description

DESCRIPTION

LIF (Logical Interchange Format) is a Hewlett-Packard standard disc format that may be used for interchange of files among various HP computer systems. A LIF volume contains a header (identifying it as a LIF volume) and a directory that defines the contents (i.e. files) of the volume. The size of the directory is fixed when the volume is initialized (see *lifinit*(1)) and sets an upper bound on the number of files that may be created on the volume.

HP-UX contains a set of utilities (referred to hereafter as *lif**(1)) that may be used to initialize a LIF volume (i.e. create a header and an empty directory), copy files to and from LIF volumes, list the contents of LIF volumes, remove LIF files, and rename LIF files.

The *lif**(1) utilities are the only utilities within HP-UX where the internal contents of a LIF volume are known. To the rest of HP-UX a LIF volume is simply a file containing some unspecified data. The term 'LIF volume' should in no way be confused with the HP-UX notion of a file system volume or mountable volume.

A LIF volume may be created on any HP-UX file (either regular disc file or device special file) that supports random access via *lseek*(2). Note that you should **not** mount the special file before using the *lif**(1) routines. See *lifinit*(1) for details. Within a LIF volume, individual files are identified by 1 to 10 character file names. File names may consist of upper-case alphanumeric characters (A through Z, 0 through 9) and the underscore character (_). The first character of a LIF file name must be a letter. The *lif**(1) utilities will accept any file name, including illegal file names generated on other systems, but will only create legal names. For example, file names containing lower-case letters will be read but not created.

LIF file names are specified to the *lif**(1) utilities by concatenating the HP-UX path name for the LIF volume with the LIF file name, separating the two with a colon (:). For example,

/dev/floppy1:ABC	specifies LIF file ABC within HP-UX device special file /dev/floppy1.
myfile:ABC	specifies LIF file ABC within HP-UX disc file 'myfile'.

Note that this file naming convention is applicable only for use as arguments to the *lif**(1) utilities and do not constitute legal path names for any other use within HP-UX.

HARDWARE DEPENDENCIES

Series 500/600/700:

You **must** use a character special file to access the media.

SEE ALSO

lifcp(1), lifinit(1), lifls(1), lifrename(1), lifrm(1).

NAME

lifcp – copy to or from LIF files

SYNOPSIS

lifcp [–**a**] [–**b**] [–**t**] file1 file2
lifcp [–**a**] [–**b**] [–**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.

The –**a** option will cause an ASCII copy. In case of HP-UX to LIF copy, a LIF ASCII file is created. The –**b** option will force binary copy, creating LIF BINARY files. If neither of the –**a** or –**b** options are given, **lifcp** will guess whether to do an ASCII or a binary copy based on the magic number of the file. Text files are ASCII and object files are treated as binary.

If the –**t** option is given, the HP-UX file names will be translated to a name acceptable by a LIF utility. That is, all the lower-case letters will be substituted by upper-case letters and all other characters except numeric will be substituted by 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.

A LIF file name is recognized by the embedded colon (:) delimiter (see *lif*(1) 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 in or standard out, depending on its position in the argument list.

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 before using *lifcp*.

HARDWARE DEPENDENCIES

Series 500/600/700:
You **must** use a character special file to access the media.

EXAMPLES

lifcp abc liffile:CDE
copy file abc to LIF file CDE within HP-UX file liffile.

lifcp abc\\: liffile:CDE
copy file abc: to LIF file CDE in liffile.

lifcp –t abc def liffile:
copy files abc and def to lif files ABC and DEF within liffile.

lifcp liffile:ABC .
copy LIF file ABC within liffile to file ABC within current directory.

lifcp – /dev/floppy:A_FILE
copy standard input to LIF file A_FILE on LIF volume /dev/floppy.

lifcp liffile:ABC /dev/floppy:CDE
copy LIF file ABC in liffile to LIF file CDE in /dev/floppy.

lifcp liffile:ABC –
copy LIF file ABC in liffile to standard out.

lifcp * liffile:
copy all files within current directory to LIF files of the same name on LIF volume liffile (may cause errors if file names in the current directory do not obey LIF naming conventions!).

SEE ALSO

lif(1), 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. *Option*s may appear in any order. Their meanings are:

–**v***nnn* Sets the volume size to *nnn* bytes. If *nnn* is not a multiple of 1024, 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 omitted, the volume name is set to the last component of the path name specified by *file*. If *string* is longer than 6 characters, it is truncated.

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 500/600/700:

You **must** use a character special file to access the media.

EXAMPLES

lifinit -v500000 -d10 x
lifinit /dev/rmf0

SEE ALSO

lif(1), 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.

BUGS

If your media has never been initialized, it must be initialized (using *sdfinit*(8)) before *lifinit* can be used.

NAME

lifls – list contents of 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. There are two options:

–l List in long format, giving volume name, volume size, directory start, directory size, file type, file size, file start, extension, date created, last volume and volume number.

–C Force multiple column output format regardless of STDOUT type.

Note that you should **not** mount the special file before using *lifls.*

HARDWARE DEPENDENCIES

Series 500/600/700:

You **must** use a character special file to access the media.

EXAMPLES

lifls -l ../TEST/header
lifls /dev/rmf0

SEE ALSO

lif(1), 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*(1) 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 500/600/700:

You **must** use a character special file to access the media.

EXAMPLES

lifrename liffile:A_FILE B_FILE
lifrename /dev/floppy:ABC CDE

SEE ALSO

lif(1), 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*(1).

Note that you should **not** mount the special file before using *lifrm.*

HARDWARE DEPENDENCIES

Series 500/600/700:

You **must** use a character special file to access the media.

EXAMPLES

lifrm liffile:MAN
lifrm /dev/rmf0:F

SEE ALSO

lif(1), 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 III

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

link, unlink – exercise link and unlink system calls

SYNOPSIS

/**etc**/**link** file1 file2
/**etc**/**unlink** file

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Link and *unlink* perform their respective system calls on their arguments, abandoning most error checking. These commands may only be executed by the super-user.

RETURN VALUE

0 - successful *link*.
1 - input syntax error.
2 - *link* call failed (*unlink* will never report failure).

SEE ALSO

rm(1), link(2), unlink(2).

NAME

lint – a C program checker/verifier

SYNOPSIS

lint [**–abchnpuvxDUI**] file ...

HP-UX COMPATIBILITY

Level:	C Compiler – HP-UX/STANDARD
Origin:	System III

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 which 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 which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

It is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. By default, *lint* uses function definitions from the standard lint library **llib–lc.ln**; function definitions from the portable lint library **llib–port.ln** are used when *lint* is invoked with the **–p** option.

Any number of *lint* options may be used, in any order. 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 a large number of such complaints.)

–c Suppress complaints about casts that have questionable portability.

–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:

–n Do not check compatibility against either the standard or the portable lint library.

–p Attempt to check portability to other dialects (IBM and GCOS) of C.

The **–D**, **–U**, and **–I** options of *cc*(1) are also recognized as separate arguments.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*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 in this file.

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, 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.

FILES

/usr/lib/lint[12]	programs
/usr/lib/llib–lc.ln	declarations for standard functions (binary format; source is in **/usr/lib/llib–lc**)
/usr/lib/llib–port.ln	declarations for portable functions (binary format; source is in **/usr/lib/llib–port**)
/usr/tmp/*lint*	temporaries

SEE ALSO

cc(1).

BUGS

Exit(2) and other functions which do not return are not understood.

NAME

login – sign on

SYNOPSIS

/etc/login [name [hangup]]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It cannot be invoked explicitly (except by the super-user), but is invoked by the system when a connection is first established, or after the previous user has logged out by sending an "end-of-file" (control–D) to his or her initial shell. (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

Login asks for your user name (preferably lower-case), 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. Note that, if you have a password, you will be asked for it whether your user name is valid or not. This is done to make it more difficult for an unauthorized user to log in on the system by trial and error.

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 are likely to be silently disconnected.

After a successful login, the accounting files are updated, and the user and group id's and the working directory are initialized. *Login* then executes a command interpreter (usually *sh*(1)), according to specifications found in the **/etc/passwd** file. If the shell is executed, the profile files **/etc/profile** and **$HOME/.profile** are executed, 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. Argument 0 of the command interpreter is – followed by the last component of the interpreter's path name. The *environment* (see *environ*(7)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
LOGNAME=*your-login-name*

For the super-user, PATH is augmented to include /**etc**.

The presence of *name* suppresses the **login:** prompt, and uses *name* as the login name. *Hangup* is the time, in seconds, before hanging up if the login is unsuccessful. The default is 60 seconds. Zero (0) seconds indicates an indefinite wait.

FILES

/etc/utmp	users currently logged in
/usr/adm/wtmp	history of logins, logouts, and date changes
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file – defines users, passwords, and groups
/etc/profile	system profile (initialization for all users)
$HOME/.profile	personal profile (individual user initialization)

SEE ALSO

mail(1), newgrp(1), passwd(1), sh(1), su(1), passwd(5), profile(5), environ(7), getty(8).

DIAGNOSTICS

Login incorrect
: if the user name or the password is incorrect.

No shell, cannot open password file, no directory:
: consult your system manager.

Your password has expired. Choose a new one.
: if password aging is implemented.

No entry in utmp:
: *utmp* file exists but your terminal is not listed there. Caused by serious trouble in file system or *ttyname*(3).

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 login in /etc or /bin on root:
: same as above except sub-root login command not found.

Invalid ID:
: *setuid* or *setgid* failed.

No directory:
: 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.

NAME
logname – get login name

SYNOPSIS
logname

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

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 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).

This one-line example intends to build a new library from existing **.o** files.

```
ar cr library `lorder *.o | tsort`
```

Ranlib(1) serves a similar purpose and is more efficient for libraries.

FILES

*symref, *symdef	temp files

SEE ALSO
ar(1), ld(1), tsort(1), ranlib(1).

BUGS
Object files whose name 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

lpd – line printer daemon

SYNOPSIS

/usr/lib/lpd [printername]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Lpd is the daemon for the line printer. It is automatically initiated by the line printer spooling command, *lpr*.

Printername is the name of a printer device file, without the initial "/dev/" (i.e. **lp8**). If *printername* is not specified, the default printer **lp** is used.

Lpd searches the directory /**usr**/**spool** for a directory of the same name as the specified *printername*. Thus, /**usr**/**spool**/**lp** is used by default. To be able to use other printers, a directory for each printer must be created in /**usr**/**spool** by the super-user. (Other spool directories can be specified by *lpr* with the **–d** option.)

The file **lock** is used to prevent two daemons from becoming active on the same spool directory. Several daemons can be active simultaneously, as long as they are working on different spool directories. After the program has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with "df". Each such file is submitted as a job. Each line of a job file must begin with a key character to specify what to do with the remainder of the line. The key characters are:

L	specifies that the remainder of the line is to be sent as a literal.
I	is the same as **L**, but signals the $IDENT card which is to be mailed back by the mail option.
B	specifies that the rest of the line is a file name. That file is to be printed.
F	is the same as **B** except a form-feed is prepended to the file.
U	specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.
M	is followed by a user ID; after the job is sent, a message is mailed to the user via the *mail*(1) command to verify the sending of the job.
D	specifies that the remainder of the line is a pathname for a specific printer.

Any error encountered will cause the daemon to give up, wait 10 seconds, and start over. This means that an improperly constructed "df" file may cause the same job to be submitted every 10 seconds.

To restart *lpd* (in the case of hardware or software malfunction), it is necessary to first kill the old daemon (if it is still alive), and remove the lock file (if present), before initiating the new daemon. This is done automatically by /**etc**/**rc** when the system is brought up, in case there were any jobs left in the spooling directory when the system last went down.

Lpd will pass ASCII escape sequences to the output device. This enables users to access special capabilities like expanded type fonts, alternate character sets, etc.

FILES

/usr/spool/lp/*	default spool area for line printer daemon.
/usr/spool/*printername*/*	spool area for additional printers
/etc/passwd	used to get the user's name.
/dev/lp	default line printer device.
/dev/lp*	additional printer devices.

SEE ALSO

lpr(1).

NAME

lpr – line printer spooler

SYNOPSIS

lpr [option ...] [name ...]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Lpr causes the named files to be queued for printing on a line printer. If no *name*s appear, the standard input is assumed; thus *lpr* may be used as a filter. If the **–d** option is absent, the printer /**dev**/**lp** is assumed.

The following *option*s may be given (each as a separate argument and in any order) before any file name arguments:

–c	Makes a copy of the file to be sent before returning to the user.
–r	Removes the file after sending it.
–m	When printing is complete, reports that fact by *mail*(1).
–n	Does not report the completion of printing by *mail*(1). This is the default option.
–d	Is followed by a printer name, and causes output to be sent to that printer.

Note that *lpr* does not force a page-eject at the end of a printing. Thus, if printing stops in the middle of a page, the upcoming header will not begin at the appropriate place. The most common solution to this is to pipe your printing through *pr*(1).

FILES

/etc/passwd	user's identification and accounting data.
/usr/lib/lpd	line printer daemon.
/usr/spool/*printername*/*	spool area.

SEE ALSO

lpd(1), pr(1).

NAME

ls, l, ll, lsf, lsr, lsx – list contents of directories

SYNOPSIS

ls [**–abcdfgilmnoqrstux1ACFR**] [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 III and UCB

DESCRIPTION

For each directory named, *ls* lists the contents of that directory; for each file named, *ls* repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

If you are the super-user, *ls* defaults to listing all files except **.** and **..**.

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. Finally, there is a stream output format in which files are listed across the page, separated by "," characters. The **–m** flag enables this format.

There are several options:

–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 contain the major device number in decimal and the minor device number in hexadecimal, rather than a size.

–o The same as **–l**, except that the group is not printed.

–n The same as **–l**, except that the owner number is printed, and all group information is omitted.

–g The same as **–l**, except that the owner is not printed (overrides **–o** if both options are specified).

–t Sort by time of last modification (latest first) instead of by name.

–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**.

–s Give size in blocks (including indirect blocks) for each entry.

–d If argument is a directory, network, or SRM file, list only its name; often used with **–l** to get the status of a directory or directory-like file.

–r Reverse the order of sort to get reverse alphabetic or oldest first, as appropriate.

–u Use time of last access instead of last modification for sorting (with the **–t** option) and/or printing (with the **–l** option).

–c Use time of last modification of the inode (mode, etc.) instead of last modification of the file for sorting (**–t**) and/or printing (**–l**).

–i	For each file, print the i-number in the first column of the report.
–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.
–m	Force stream output format, i.e. a comma-separated list.
–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.
–C	Force multi-column output to a file or a pipe.
–q	Force printing of non-graphic characters in file names as the character "?"; this normally happens only if the output device is a teletype.
–b	Force printing of non-graphic characters to be in *ddd* notation, in octal.
–x	Force columnar printing to be sorted across rather than down the page.
–F	Cause directories to be marked with a trailing "/" and executable files to be marked with a trailing "*".
–R	Recursively list subdirectories encountered. Network 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.

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 11 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;
s	if the entry is a Shared Resource Manager (SRM) special file;
–	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 bit of the mode are capitalized 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 500/600/700:

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**.

SEE ALSO

chmod(1), find(1).

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

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* error-prone.

Column widths choices are poor for terminals which can tab.

NAME

lsdev – list device drivers in the system

SYNOPSIS

/etc/lsdev [major...]

HP-UX COMPATIBILITY

Level: HP-UX/EXTENDED

Origin: HP

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.

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

mail [**–rpqe**] [**–f** file]

mail persons

rmail persons

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input to determine the disposition of the message:

<new-line>	Go on to next message.
+	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
–	Go back to previous message.
s [*files*]	Save message in the named *files* (**mbox** is default).
w [*files*]	Save message, without its header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons* (yourself is default).
q	Put undeleted mail back in the *mailfile* and stop.
EOT (control-d)	Same as **q**.
x	Put all mail back in the *mailfile* unchanged and stop.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

The optional arguments alter the printing of the mail:

–r	causes messages to be printed in first-in, first-out order.
–p	causes all mail to be printed without prompting for disposition.
–q	causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the printing of the current message.
–f*file*	causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.
–e	The mail is simply tested for existence and the exit code returned. 0 = mail present 1 = no mail 2 = other error

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a **.**) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From . . .") are preceded with a >. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the **dead.letter** will be saved to allow editing and resending.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will interpret that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, he is informed of the presence of mail, if any.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*	incoming mail for user *; *mailfile*
$HOME/mbox	saved mail
$MAIL	*mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), uucp(1C), write(1).

BUGS

Race conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

make – maintain, update, recompile programs

SYNOPSIS

make [**–f** makefile] [**–p**] [**–i**] [**–k**] [**–s**] [**–r**] [**–n**] [**–b**] [**–e**] [**–t**] [**–q**] [**–d**] [names]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

The following is a brief description of all options and some special names. Options may occur in any order.

–f *makefile*	Description file name. *Makefile* is assumed to be the name of a description file. A file name of – denotes the standard input. The contents of *makefile* override the built-in rules if they are present.
–p	Print out the complete set of macro definitions and target descriptions.
–i	Ignore error codes returned by invoked commands. This mode is also entered if the fake target name **.IGNORE** appears in the description file.
–k	Abandon work on the current entry, but continue on other branches that do not depend on that entry.
–s	Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.
–r	Do not use the built-in rules.
–n	No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.
–b	Compatibility mode for old (Version 7) makefiles.
–e	Environment variables override assignments within makefiles.
–t	Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
–d	Debug mode. Print out detailed information on files and times examined. (This is intended for debugging the *make* command itself.)
–q	Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

The "built-in" dependency targets are:

.DEFAULT
If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS
Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT
Same effect as the **–s** option.

.IGNORE
Same effect as the **–i** option.

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If *makefile* is –, the standard input is taken. If no **–f** option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. More than one **–f** makefile argument pair may appear.

Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, followed by a colon (:), followed by a (possibly null) list of prerequisite files or dependencies. Text following a **;** and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o –o pgm
a.o: incl.h a.c
        cc –c a.c
b.o: incl.h b.c
        cc –c b.c
```

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the **–s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the first character of the command is @. The **–n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). Note that this feature does not work if **MAKE** is enclosed in braces, as in **${MAKE}**. The **–t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **–i** option is present, or the entry **.IGNORE:** appears in *makefile*, or if the line specifying the command begins with <tab><hyphen>, the error is ignored. If the **–k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **–b** option allows old makefiles (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null) command associated with them. The previous version of *make* assumed if no command was specified explicitly that the command was null.

Interrupt and quit cause the target to be deleted unless the target depends on the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **–e** option causes the environment to override the macro assignments in a makefile.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **–f**, **–p**, and **–d**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **–n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make –n** recursively on a whole software system to see what would have been executed. This is because the **–n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form *string1* = *string2* are macro definitions. Subsequent appearances of $(*string1*[:*subst1* = [*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1* = *subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out of date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
        .c.o:
                cc –c –O $*.c
```

or:

```
        .c.o:
                cc –c –O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, ./ is generated. The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make –p –f – 2>/dev/null </dev/null
```

The only peculiarity in this output is the (**null**) string which *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see *sccsfile*(5)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix it is incompatible with *make*'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e. **.c:**) is the definition of how to build *x* from *x*.**c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite.

The default list is:

```
    .SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly:

```
    pgm: a.o b.o
            cc a.o b.o –o pgm
    a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1) respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parenthesis, it is assumed to be an archive library, the string within parenthesis referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form ***.XX*.a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
    lib:    lib(file1.o) lib(file2.o) lib(file3.o)
            @echo lib is now up to date
    .c.a:
            $(CC) –c $(CFLAGS) $<
            ar rv $@ $*.o
            rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
    lib:    lib(file1.o) lib(file2.o) lib(file3.o)
            $(CC) –c $(CFLAGS) $(?:.o=.c)
            ar rv lib $?
            rm $?  @echo lib is now up to date
    .c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out of date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile
s.[Mm]akefile

SEE ALSO

sh(1), mk(8).

BUGS

Some commands return non-zero status inappropriately; use **–i** to overcome the difficulty.

Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*.

The syntax **(lib(file1.o file2.o file3.o)** is illegal.

You cannot build **lib(file.o)** from **file.o**.

The macro **$(a:.o = .c˜)** doesn't work.

There is a limit of 2500 characters, including the terminating new-line, for expanded dependency lines.

NAME

man – on-line manual command

SYNOPSIS

man –k keyword ...
man –f file ...
man [–] [section] title ...

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Man is a program which gives information from the programmers manual. It can be asked to form one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. It can also provide on-line access to the sections of the printed manual.

When given the option **–k** and a set of keywords, *man* prints out a one line synopsis of each manual section whose listing in the table of contents contains that keyword.

When given the option **–f** and a list of file names, *man* attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither **–k** nor **–f** is specified, *man* formats a specified set of manual pages. If a section specifier is given *man* looks in that section of the manual for the given *titles*. *Section* is an arabic section number, i.e. 3, which may be followed by a single letter classifier, i.e. 1g indicating a graphics program in section 1. If *section* is omitted, *man* searches all sections of the manual, giving preference to commands over subroutines in system libraries, and printing the first section it finds, if any.

If the standard output is a teletype, or if the flag – is given, then *man* pipes its output through *rmnl*(1) to delete useless blank lines, *ul*(1) to create proper underlines for different terminals, and through *more*(1) to stop after each page. Hit a space to continue.

If the /usr/man/cat? directory is present and the file is not in it, but the file exists in /usr/man/man?, then the page is formatted and installed in /usr/man/cat? on first access. If only the /usr/man/cat? directories are present and/or nroff is not installed then only those pages which have been preformatted are displayable.

FILES

/usr/man/man?/*
/usr/man/cat?/*

SEE ALSO

rmnl(1), ul(1), more(1), whereis(1), catman(8).

BUGS

The manual is supposed to be reproducible either on the phototypesetter or on a typewriter. However, on a typewriter some information is necessarily lost.

NAME

mesg – permit or deny messages to terminal

SYNOPSIS

mesg [**n**] [**y**]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME
mkdir – make a directory

SYNOPSIS
mkdir dirname ...

HP-UX COMPATIBILITY
Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION
Mkdir creates specified directories in mode 777, masked by the current value of ***umask***. Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO
rm(1), umask(1).

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

NAME

mm – print documents formatted with MM macros

SYNOPSIS

mm [options] [files]

HP-UX COMPATIBILITY

Level:	Text Processing - HP-UX/EXTENDED
Origin:	System III

DESCRIPTION

Mm can be used to type out documents using *nroff*(1) and the MM text formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn*(1) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff*(1) and MM are generated, depending on the options selected.

Options for *mm* are given below. Any other arguments or flags (e.g., **–rC3**) are passed to *nroff*(1) or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

–T*term*	Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **$TERM** from the environment (see *profile*(5) and *environ*(7)) as the value of *term*, if **$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.
–12	Indicates that the document is to be produced in 12-pitch. May be used when **$TERM** is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)
–c	Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000A**, **382**, **4014**, **tek**, **1620**, and **X**.
–e	Causes *mm* to invoke *neqn*(1).
–t	Causes *mm* to invoke *tbl*(1).
–E	Invokes the **–e** option of *nroff*(1).
–y	Causes *mm* to use the non-compacted version of the macros (see *mm*(7)).

As an example (assuming that the shell variable **$TERM** is set in the environment to **450**), the two command lines below are equivalent:

```
mm –t –rC3 –12 ghh*
tbl ghh* | nroff –cm –T450 –12 –h –rC3
```

Mm reads the standard input when – is specified instead of any file names. (Mentioning other files together with – leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws  mm –
```

The following helpful hints should aid you in using these macros:

1. *Mm* invokes *nroff*(1) with the **–h** flag. With this flag, *nroff*(1) assumes that the terminal has tabs set every 8 character positions.
2. Use the **–o***list* option of *nroff*(1) to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **–e**, **–t**, and – options, *together* with the **–o***list* option of *nroff*(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the **–s** option of *nroff*(1) (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The **–s** option of *nroff*(1) does not work with the **–c** option of *mm*, or if *mm* automatically invokes *col*(1) (see **–c** option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you'll get (often subtle) garbage; however, if you are redirecting output into a file, use the **–T37** option, and then use the appropriate terminal filter when you actually print that file.

HARDWARE DEPENDENCIES

Series 500/600/700:

Eqn(1), *greek*(1), and *mmt*(1) are not currently supported.

SEE ALSO

col(1), env(1), eqn(1), greek(1), mmt(1), nroff(1), tbl(1), profile(5), mm(7), term(7).
MM–Memorandum Macros in *HP-UX: Selected Articles.*

DIAGNOSTICS

"mm: no input file" if none of the arguments is a readable file and *mm* is not used as a filter.

NAME

more, page – file perusal filter for crt viewing

SYNOPSIS

more [**–cdflsu**] [–n] [+linenumber] [+/pattern] [name ...]

page [more options]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	UCB

DESCRIPTION

More is a filter which allows examination of continuous text, one screenful at a time, on a soft-copy terminal. It normally pauses after each screenful, printing ––**More**–– at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilites are enumerated later.

The command line options are:

–n An integer which is the size (in lines) of the window which *more* will use instead of the default.

–c *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

–d *More* will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

–f This causes *more* to count logical lines, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen postions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

–l Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

–s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

–u Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The –**u** option suppresses this processing.

+ linenumber
Start up at *linenumber.*

+ /pattern
Start up two lines before the line containing the regular expression *pattern.*

If the program is invoked as *page,* then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and *k* – 1 rather than *k* – 2 lines are printed in each screenful, where *k* is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the **–c** mode of operation, the shell command sequence *MORE = ´-c´ ; export MORE* would cause all invocations of *more*, including invocations by programs such as *man* and *msgs*, to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the **– –More– –** prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i <space>
: display *i* more lines, (or another screenful if no argument is given).

^D
: display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d
: same as ^D (control-D).

i z
: same as typing a space except that *i*, if present, becomes the new window size.

i s
: skip *i* lines and print a screenful of lines.

i f
: skip *i* screenfuls and print a screenful of lines.

q or Q
: Exit from *more*.

=
: Display the current line number.

v
: Start up the editor *vi* at the current line.

h
: Help command; give a description of all the *more* commands.

i /expr
: search for the *i* -th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

i n
: search for the *i* -th occurrence of the last regular expression entered.

: (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
: invoke a shell with *command*. The characters "%" and "!" in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, "%" is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i :n
: skip to the *i* -th next file given in the command line (skips to last file if n doesn't make sense).

i :p
: skip to the *i* -th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f
: display the current file name and line number.

:q or :Q
: exit from *more* (same as q or Q).

.
: (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the **– –More– –(*xx*%)**.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control–\). *More* will stop sending output, and will display the usual **--More--** prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat,* except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff –ms +2 doc.n | more -s
```

FILES

/etc/termcap	terminal data base
/usr/lib/more.help	help file

SEE ALSO

man(1), msgs(1), sh(1), termcap(5).

NAME

mount, umount – mount and unmount file system

SYNOPSIS

/**etc**/**mount** [special directory [**–r**]]

/**etc**/**umount** special

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The *directory* must exist already; it becomes the name of the root of the newly mounted file system. *Directory* must be given as an absolute path name.

These commands maintain a table of mounted devices. If invoked with no arguments, *mount* prints the table.

The optional last argument indicates that the file is to be mounted read-only. Physically write-protected and magnetic tape file systems must be mounted in this way or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

FILES

/etc/mnttab	mount table

SEE ALSO

mount(2), mnttab(5).

DIAGNOSTICS

Attempts to mount a currently-mounted volume under another name will result in an error [EBUSY].

If an attempt to read and (partially) verify the disc label information fails, the mount will fail.

Umount complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory.

WARNING

If virtual memory is brought up on a volume other than the root volume, and if that volume is then mounted, it cannot be unmounted.

BUGS

Some degree of validation is done on the file system, however it is generally unwise to mount garbage file systems.
The third parameter may be anything which has the effect of **–r**.
An error will occur if *mnttab* does not exist.
Names are truncated to MNTLEN bytes (see *mnttab*(5)).

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**–t** tapename] command [count]

HP-UX COMPATABILITY

Level:	Magnetic Tape Support – HP-UX/STANDARD
Origin:	UCB

DESCRIPTION

Mt is used to give commands to the tape drive. If *tapename* is not specified, /dev/rmt12 is used. If *count* is not specified, 1 is assumed.

Here are the commands:

eof	write *count* end-of-file marks
fsf	space forward *count* files
fsr	space forward *count* records
bsf	space backward *count* files
bsr	space backward *count* records
rew	rewind tape
offl	rewind tape and go offline.

FILES

/dev/mt*	Magnetic tape interface
/dev/rmt*	Raw magnetic tape interface

/dev/rmt12 (or whatever drive is used) must be described as a Berkeley compatibility-mode tape drive without rewind for *mt* to operate as expected.

SEE ALSO

mt(4).

NAME

mvdir – move a directory

SYNOPSIS

/etc/mvdir dirname name

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Mvdir moves and/or renames directories within a file system. *Dirname* must be a directory. If *name* exists, then the directory *dirname* is moved. If *name* does not exist, *dirname* is simply renamed.

Name cannot be a subdirectory of *dirname*. *Dirname* may be a subdirectory of *name*, but the shorthand notations **.** and **..** must be used in naming the directories, because *mvdir* does not allow explicit directory names to be used when one directory is a subdirectory of the other.

Only the super-user can use *mvdir*.

SEE ALSO

mkdir(1).

BUGS

The restriction on names is intended to prevent creation of a (cyclic) sub-tree that cannot be reached from the root. The test is strictly by name, thus creating such a sub-tree is still possible. The super-user is cautioned to be very careful in his use of the names **.** and **..** while moving directories.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [group]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Newgrp changes the group identification of its caller. The same user remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

Newgrp without an argument changes the group identification to the group in the password file; in effect it changes the group identification back to the caller's original group.

A password is demanded if the group has a password and the user himself does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

Newgrp is recognized by the shell, and causes the *newgrp* program to execute on top of the shell (instead of as the usual sub-process). If *newgrp* fails, you cannot be returned to your old shell, and you are thus logged out.

FILES

/etc/group
/etc/passwd

SEE ALSO

login(1), group(5).

DIAGNOSTICS

Sorry:	You didn't qualify as a group member.
Unknown group:	The group name was not in /etc/group.
Permission denied:	If a password must be given, it can only come from a teletype port. If the *stdin* is a non-tty file, this message is given.
You have no shell:	*Exec* of the shell failed.

BUGS

There is no convenient way to enter a password into **/etc/group**.
Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.
Shell variables which are not exported are lost.

NAME

news – print news items

SYNOPSIS

news [**–a**] [**–n**] [**–s**] [items]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *News* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

The **–a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The **–n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The **–s** option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If an interrupt is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

mail(1), profile(5), environ(7).

NAME

nice – run a command at low priority

SYNOPSIS

nice [–increment] command [arguments]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., −−**10**.

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nm – print name list (symbol table) of object file

SYNOPSIS

nm [**–gnopru**] [file ...]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III
Remarks:	This manual page describes *nm* as implemented on the Series 500/600/700. Refer to other *nm* manual pages for information valid for other implementations.

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced, preceded by the member name on a separate line. If no *file* is given, the symbols in **a.out** are listed.

Options are:

–g	Print only global (external) symbols.
–n	Sort numerically rather than alphabetically.
–o	Prepend file or archive element name to each output line rather than only once. This option can be used to make piping to *grep*(1) more meaningful.
–p	Don't sort; print in symbol-table order.
–r	Sort in reverse order.
–u	Print only undefined symbols.

The output from *nm* consists of five columns of data. The following is a portion of a typical output:

```
.  X  IDATA      00000108                A_iob
.  X  IDATA      000002a0                A_sctab
.  X  ICOMM      00000400 0 00000440     A_sibuf
.  X  ICOMM      00000400 0 00000840     A_sobuf
.  .  UDATA      00000c40                Aallocs
.  X  FUNC       EDS c04 002a8 00000003  __cleanup
.  X  DDATA      DR 00000098             __ctype
.  X  FUNC       EDS c0c 00000 00000001  __doscan
.  X  SYSTEM     EPP 004 0000e           __exit
.  X  DDATA      DR 00000038             __iob
.  X  DCOMM      00000004 000000b0       __pfile
.  X  DDATA      DR 00000090             __sctab
.  X  PTR        1 00000a 000000b4       __sibuf
.  X  PTR        1 00000c 000000b8       __sobuf
.  .  FILENAME   0000000a                _exit.o
.  .  FILENAME   0000000f                _print.o
```

From left to right, the first column specifies whether the symbol is defined (.) or undefined (U). The second column specifies whether the symbol is non-external (.) or external (X). The third column gives the linker symbol type (as defined in *a.out.h* and described below). The fourth column lists the data associated with the specified symbol type. The fifth column gives the name of the system call, file, variable, array, common, etc., described by that entry in the symbol table.

Up to four data elements are reported in the fourth column. If they are not symbolic values (such as 'EDS' or 'DR'), then they are hexadecimal values. The number of data elements reported depends on the symbol type. Each symbol type has one to four parameters associated with it, whose values are given by the data elements in the fourth column. The symbol types and associated parameters are discussed below.

The following symbol types are supported:

ABS	not currently generated; reserved for future use.
FUNC or **ENTRY**	specifies that the entry refers to a function or procedure call. Four numbers, *ptr_type*, *segment*, *offset*, and *stt_index*, are associated. Their values are given in order, from left to right, by the data elements. *Ptr_type* consists of a single bit that is always cleared. It is symbolically represented by 'EDS'. *Ptr_type* is meaningful to the linker (see *ld*(1)), and specifies the storage format of the call in the symbol table. *Segment* specifies the code segment number (a number in the range 3073 to 4095, that indicates which code segment in the user's program space contains the desired code). *Offset* specifies the number of bytes from the beginning of the code segment where the function or procedure code begins. *Stt_index* is an indirect reference to the beginning of the function or procedure code.
SYSTEM	specifies that the entry refers to a procedure call directly into the system kernel. Three numbers, *entry_type*, *segment*, and *stt*, are associated. Their values are given by the data elements. *Entry_type* consists of a single bit that is always set. Its value is symbolically represented by 'EPP'. *Entry_type* is meaningful to the linker, and specifies the storage format of the call in the symbol table. *Segment* specifies the system code segment number (the number of a code segment among those set aside for system use; typically in the range 0 to 64). *Stt* is an indirect pointer to the beginning of the procedure code.
LABEL	specifies that the entry is the destination address for a branch instruction. Three numbers, *ptr_type*, *segment*, and *offset*, are associated. Their values are given by the data elements. *Ptr_type* consists of a single bit which is always cleared. Its value is symbolically represented by 'EDS'. *Ptr_type* is meaningful to the linker, and specifies the storage format of the address in the symbol table. *Segment* specifies the user code segment number. *Offset* specifies the number of bytes from the beginning of the code segment where the label begins.
DDATA	specifies that the entry is a directly-addressable, initialized data structure (a variable, or the beginning of an array, common, structure, etc.). Two numbers, *base_reg* and *displacement*, are associated. Their values are given by the data elements. *Base_reg* is assigned one of nine possible symbolic values which describe the addressing scheme used to find the data structure. It is meaningful to the linker. The possible symbolic values are P+, P–, DB, DL, Q+, Q–, SB, S–, and DR. *Displacement* specifies the byte offset where the data structure is located. Note that this offset is measured relative to the beginning of the data space of the file for which the *nm* listing is made. The actual byte offset of the data structure in the executable **a.out** file could change.
IDATA or **UDATA**	specifies that the entry refers to an indirectly-addressable, uninitialized array, or an indirectly-addressable, initialized common block. One number, *displacement*, is associated. Its value is given by the data element. It is identical to the *displacement* described above under **DDATA**.
DCOMM or **ICOMM**	specifies that the entry is treated as a common block. Three numbers, *blocksize*, *needs_length_word*, and *displacement*, are associated. Their values are given by the data elements. *Blocksize* is the size, in bytes, of the common block. *Needs_length_word* is a boolean value which appears in a print-out as either 0 or 1. If its value is 1, the linker places the value of (*blocksize* – 4) in the first four

bytes of the common block. This information is necessary when linking FORTRAN programs. *Displacement* is identical to that described under **DDATA** above.

PTR specifies that the entry is a pointer to an indirectly-addressable data structure (variable, array, common block, etc.). Three numbers, *ptr_to_common*, *target*, and *address*, are associated. Their values are given by the data elements. *Ptr_to_common* is an eight-bit boolean expression. Its value is given as 1 (true) or 0 (false). If true, then the entry is a pointer to a common block. If false, the entry is a pointer to some other type of data structure. *Target* is an index into the symbol table to the entry that describes the target of the data structure pointer. *Address* is a pointer to the data structure pointer; that is, an indirect pointer to the data structure.

SEGMENT not currently generated; reserved for future use.

FILENAME specifies that the entry is a file name. One number, *sequence*, is associated. Its value is given by the data element. *Sequence* reflects the order in which the linker encountered each file name.

SEE ALSO

ar(1), a.out(5), ar(5).

DIAGNOSTICS

Nm generates an error message for the following conditions:

invalid option
cannot open *file*
bad magic number
read error

NAME

nohup – run a command immune to hangups, logouts, and quits

SYNOPSIS

nohup command [arguments]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, it will be sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**; otherwise, *nohup* will fail.

If output from *nohup* is redirected to a terminal, or is not redirected at all, the output is sent to **nohup.out**.

SEE ALSO

nice(1), signal(2).

NAME

nroff – format text

SYNOPSIS

nroff [options] [files]

HP-UX COMPATIBILITY

Level:	*Nroff* - HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers. Its capabilities are described in the *NROFF/TROFF User's Manual* cited below.

An argument consisting of a minus (–) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

–o*list*	Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N*– means from *N* to the end. (See *BUGS* below.)
–n*N*	Number first generated page *N*.
–s*N*	Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N* = 1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). When *nroff* halts between pages, an ASCII **BEL** is sent to the terminal.
–r*aN*	Set register *a* (which must have a one-character name) to *N*.
–i	Read standard input after *files* are exhausted.
–q	Invoke the simultaneous input-output mode of the **.rd** request.
–z	Print only messages generated by **.tm** (terminal message) requests.
–m*name*	Precede the input *files* with the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.
–c*name*	Precede the input *files* with the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***na*
–k*name*	Compact the macros used in this invocation of *nroff*, placing the output in files **[dt].***name* in the current directory (see the May 1979 Addendum to the *NROFF/TROFF User's Manual* for details of compacting macro files).
–T*name*	Prepare output for specified terminal. Known *names* are **37** for the (default) TELETYPE® Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, and **2631** for the Hewlett Packard 2631 line printer.
–e	Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.
–h	Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.
–u*n*	Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

HARDWARE DEPENDENCIES

Series 500/600/700:

The **–c** and **–k** options are not currently supported.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file

/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

mm(1).
NROFF/TROFF User's Manual in *HP-UX: Selected Articles.*

BUGS

Nroff is keyed to Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff* generates may be off by one day from your current date.
When *nroff* is used with the **–o***list* option inside a pipeline, it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

od, xd – octal and hexadecimal dump

SYNOPSIS

od [**–bcdox**] [file] [[+]offset[**.**][**b**]]
xd [**–bcdox**] [file] [[+]offset[**.**][**b**]]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III and HP

DESCRIPTION

Od (*xd*) dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **–o** (**–x**) is the default. Each line is prepended with an offset field. For *od*, the offset is in octal, for *xd* the offset is in hexadecimal. The meanings of the format options are:

–b Interpret bytes in octal (hexadecimal).

–c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null = **0**, backspace = **b**, form-feed = **f**, new-line = **n**, return = **r**, tab = **t**; others appear as 3-digit octal numbers.

–d Interpret 16-bit words in decimal.

–o Interpret 16-bit words in octal.

–x Interpret 16-bit words in hexidecimal.

The *file* argument specifies which file is to be dumped. If no *file* argument is specified, the standard input is used.

The *offset* argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, *offset* is interpreted in decimal. If **0x** is prepended, *offset* is interpreted in hexadecimal. If **b** is appended, *offset* is interpreted in blocks of 512 bytes. If the file argument is omitted, *offset* must be preceded by +.

Dumping continues until end-of-file.

HARDWARE DEPENDENCIES

Series 500/600/700:
Xd is not currently implemented.

SEE ALSO

adb(1).

NAME

passwd – change login password

SYNOPSIS

passwd [name]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

This command changes (or installs) a password associated with the login *name*. If *name* is omitted, it defaults to *getlogin*(3) name.

The program prompts for the old password (if any) and then for the new one (twice). The caller must supply these. New passwords should be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. Only the first eight characters of the password are significant.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password, if any. The super-user need not supply the old password when he changes another user's password. Only the super-user can create a null password.

The password file is not changed if the new password is the same as the old password, or if the password has not "aged" sufficiently; see *passwd*(5)).

FILES

/etc/passwd

SEE ALSO

login(1), crypt(3C), passwd(5).

DIAGNOSTICS

Permission denied.
name is not in password file, or you are not user *name* or the super-user.

Sorry. the old password does not match.

Sorry: <x weeks since the last change
password aging is in effect, and it is too soon to change yours.

You may not change this password
the super-user has made it impossible to change your password.

Too short
passwords must be at least 4 characters long.

Please use at least one non-numeric character
your new password does not utilize a sufficiently varied selection of characters. You can override this rule by re-entering your new password 2 more times.

Please use a longer password.
your new password is not long enough to be sufficiently secure. You can override this rule by re-entering your new password 2 more times.

They don't match, try again.
the two entries of your new password are not identical.

Temporary file busy; try again later
only one user can change his password at a time.

Cannot create temporary file
see the super-user.

Cannot unlink 'filename'
see the super-user.

cannot link 'file' to 'file'.
: see the super-user.

cannot recover 'file'.
: see the super-user.

Password unchanged.
: the new and old passwords are the same.

NAME

paste – merge lines in one or more files

SYNOPSIS

paste file1 file2 . . .
paste –d list file1 file2 . . .
paste –s [**–d** list] file1 file2 . . .

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

The meanings of the options are:

–d Without this option, the new-line characters of each but the last file (or last line in case of the **–s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **–d** replace the default *tab* as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no **–s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **n** (new-line), **t** (tab), \\\\ (backslash), and **0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use *–d*"\\\\\\\\").

–s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **–d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

– May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls paste –d" " –	list directory in one column
ls paste – – – –	list directory in four columns
paste –s –d"\t\n" file	combine pairs of lines into lines

SEE ALSO

grep(1), cut(1),
pr(1): **pr –t –m**. . . works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for **–s** option, no more than 12 input files may be specified.

NAME

pc – Pascal compiler

SYNOPSIS

pc [options] files

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP

DESCRIPTION

Pc is the HP standard Pascal compiler. It accepts two types of file arguments:

(1) Arguments whose names end with **.p** are taken to be Pascal source programs. They are compiled, and each object program is left in the current directory in a file whose name is that of the source, with **.o** substituted for **.p**. (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.)

(2) Arguments whose names end with **.o** are passed on to the linker (*ld*(1)) to be linked into the final program.

The following options are recognized:

–c	Suppress linking and produce object (**.o**) files from source files;
–w	Suppress warning messages (same as **$WARN OFF$**);
–C	Enable range checking (same as **$RANGE ON$**);
–L	Write a program listing to *stdout* during compilation;
–s	Accept standard Pascal only (same as **$STANDARD_LEVEL ANSI$**);
–o outfile	Name the output file from the linker *outfile* instead of *a.out*;
–Q optfile	Specify *optfile* as the option file;
–U	Cause the names of level 1 routines and external routines to be upshifted;
–e	Write lines containing errors to *stderr*;
–v	Write expanded compiler and linker runstrings to *stderr*.

Pc also recognizes the following options, but they have no effect if they are encountered. This is for compatibility with other Pascal systems.

–g, **–p**, **–O**, **–S**, **–b**, **–i**, **–z**

A warning is written to *stderr* if any of the above options are used.

Any other options are taken to be arguments to *ld*, and are passed along to the linker. However, because the **–s** option is recognized by *pc*, it is not possible to pass the strip option to the linker. Thus, programs which are to be stripped must be run through *strip*(1) after they are compiled.

HARDWARE DEPENDENCIES

Series 500/600/700:

Strip(1) is not currently implemented, so you must suppress linking after compilation and manually link the program with *ld*(1) to be able to strip the program.

FILES

file.p	input file (Pascal source file)
file.o	object file
a.out	linked executable output file
/bin/pc	mother program
/usr/lib/pascomp	compiler
/usr/lib/paserrs	error message file
/lib/mainp.o	main program

/lib/libpc.a	Pascal run-time library
/usr/tmp/*	temporary files used by the compiler; names are created by *tmpnam*(3S).

SEE ALSO

Pascal/9000 Language Reference Manual, HP Part No. 97082–90001; *Programming in Pascal with Hewlett-Packard Pascal*, by Peter Grogono.

DIAGNOSTICS

The diagnostics produced by *pc* 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*. 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.

A list of all errors may be found in /usr/lib/paserrs.

NAME

pr – print files

SYNOPSIS

pr [options] [files]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Pr prints the named files on the standard output. If *file* is –, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **–s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

Options may appear singly or be combined in any order. Their meanings are:

$+k$	Begin printing with page k (default is 1).
–*k*	Produce k-column output (default is 1). The options **–e** and **–i** are assumed for multi-column output.
–a	Print multi-column output across the page.
–m	Merge and print all files simultaneously, one per column (overrides the **–***k*, and **–a** options).
–d	Double-space the output.
–e*ck*	Expand *input* tabs to character positions $k+1$, $2*k+1$, $3*k+1$, etc. If k is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If c (any non-digit character) is given, it is treated as the input tab character (default for c is the tab character).
–i*ck*	In *output*, replace white space wherever possible by inserting tabs to character positions $k+1$, $2*k+1$, $3*k+1$, etc. If k is 0 or is omitted, default tab settings at every eighth position are assumed. If c (any non-digit character) is given, it is treated as the output tab character (default for c is the tab character).
–n*ck*	Provide k-digit line numbering (default for k is 5). The number occupies the first $k+1$ character positions of each column of normal output or each line of **–m** output. If c (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for c is a tab).
–w*k*	Set the width of a line to k character positions (default is 72 for equal-width multi-column output, no limit otherwise).
–o*k*	Offset each line by k character positions (default is 0). The number of character positions per line is the sum of the width and offset.
–l*k*	Set the length of a page to k lines (default is 66).
–h	Use the next argument as the header to be printed instead of the file name.
–p	Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).
–f	Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
–r	Print no diagnostic reports on failure to open files.

–t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

–s*c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr –3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr –e9 –t <file1 >file2
```

FILES

/dev/tty*	to suspend messages

SEE ALSO

cat(1), lpr(1), ul(1).

NAME

prs – print and summarize an SCCS file

SYNOPSIS

prs [**–d** [dataspec]] [**–r** [SID]] [**–e**] [**–l**] [**–a**] files

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

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

All the described *keyletter* arguments apply independently to each named file:

–d[*dataspec*]	Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *Data Keywords*) interspersed with optional user supplied text.
–r[*SID*]	Used to specify the SCCS IDentification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.
–e	Requests information for all deltas created *earlier* than and including the delta designated via the **–r** keyletter.
–l	Requests information for all deltas created *later* than and including the delta designated via the **–r** keyletter.
–a	Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, i.e., delta type = *D*, deltas. If the **–a** keyletter is not specified, information for existing deltas only is provided.

Data Keywords

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

Keyword	Data Item	File Selection	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	″	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	″	nnnnn	S
:Ld:	Lines deleted by Delta	″	nnnnn	S
:Lu:	Lines unchanged by Delta	″	nnnnn	S
:DT:	Delta type	″	D or R	S
:I:	SCCS ID string (SID)	″	:R:.:L:.:B:.:S:	S
:R:	Release number	″	nnnn	S
:L:	Level number	″	nnnn	S
:B:	Branch number	″	nnnn	S
:S:	Sequence number	″	nnnn	S
:D:	Date Delta created	″	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	″	nn	S
:Dm:	Month Delta created	″	nn	S
:Dd:	Day Delta created	″	nn	S
:T:	Time Delta created	″	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	″	nn	S
:Tm:	Minutes Delta created	″	nn	S
:Ts:	Seconds Delta created	″	nn	S
:P:	Programmer who created Delta	″	logname	S
:DS:	Delta sequence number	″	nnnn	S
:DP:	Predecessor Delta seq-no.	″	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	″	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq#)	″	:DS: :DS:...	S
:Dx:	Deltas excluded (seq#)	″	:DS: :DS:...	S
:Dg:	Deltas ignored (seq#)	″	:DS: :DS:...	S
:MR:	MR numbers for delta	″	text	M
:C:	Comments for delta	″	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	″	text	S
:MF:	MR validation flag	″	*yes or no*	S
:MP:	MR validation pgm name	″	text	S
:KF:	Keyword error/warning flag	″	*yes or no*	S
:BF:	Branch flag	″	*yes or no*	S
:J:	Joint edit flag	″	*yes or no*	S
:LK:	Locked releases	″	:R:...	S
:Q:	User defined keyword	″	text	S
:M:	Module name	″	text	S
:FB:	Floor boundary	″	:R:	S
:CB:	Ceiling boundary	″	:R:	S
:Ds:	Default SID	″	:I:	S
:ND:	Null delta flag	″	*yes or no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	″	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

User supplied text is any text other than recognized data keywords. Escapes may be used as follows:

tab	\t
new-line	\n
colon	\:
backspace	\b
carriage-return	\r
formfeed	\f
backslash	\\
single quote	\'

The default *dataspec* is:

```
:Dt::DL:0MRs:0MR:COMMENTS:0C:
```

If no option letters (or only **–a**) are given, *prs* prints the file name, using the default *dataspec*, and the **–e** option; thus, information on all deltas is produced.

EXAMPLES

```
prs –d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file
```

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a special case (when no **–d** keyletter is given):

```
prs s.file
```

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(5).
Source Code Control System User's Guide in *HP-UX: Selected Articles.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – report process status

SYNOPSIS

ps [**–edafl**] [**–c** corefile] [**–s** swapdev] [**–n** namelist] [**–t** tlist] [**–p** plist] [**–u** ulist] [**–g** glist]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Ps prints certain information about active processes. With no options, information is printed about processes associated with the current terminal. Otherwise, the information that is displayed is controlled by the following options:

–e	Print information about all processes.
–d	Print information about all processes, except process group leaders.
–a	Print information about all processes, except process group leaders and processes not associated with a terminal.
–f	Generate a *full* listing. (Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.
–l	Generate a *long* listing. See below.
–c *corefile*	Use the file *corefile* in place of /**dev**/**mem**.
–s *swapdev*	Use the file *swapdev* in place of /**dev**/**swap**. This is useful when examining a *corefile*; a *swapdev* of /**dev**/**null** will cause the user block to be zeroed out.
–n *namelist*	The argument will be taken as the name of an alternate *namelist* (/**unix** is the default).
–t *tlist*	Restrict listing to data about the processes associated with the terminals given in *tlist*, where *tlist* can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.
–p *plist*	Restrict listing to data about processes whose process ID numbers are given in *plist*, where *plist* is in the same format as *tlist*.
–u *ulist*	Restrict listing to data about processes whose user ID numbers or login names are given in *ulist*, where *ulist* is in the same format as *tlist*. In the listing, the numerical user ID will be printed unless the **–f** option is used, in which case the login name will be printed.
–g *glist*	Restrict listing to data about processes whose process groups are given in *glist*, where *glist* is a list of process group leaders and is in the same format as *tlist*.

The column headings and the meaning of the columns in a *ps* listing are given below. The letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear. **All** means that the heading always appears. Note that these two options only determine what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:

01	in core;
02	system process;
04	locked in core (e.g., for physical I/O);
10	being swapped;
20	being traced by another process.

S (l) The state of the process:

O	non-existent;
S	sleeping;
W	waiting;
R	running;
I	intermediate;
Z	terminated;
T	stopped.

UID	(f,l)	The user ID number of the process owner; the login name is printed under the **–f** option.
PID	(all)	The process ID of the process; it is possible to kill a process if you know this datum.
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
STIME	(f)	Starting time of the process.
PRI	(l)	The priority of the process; higher numbers mean lower priority.
NI	(l)	Nice value; used in priority computation.
ADDR	(l)	The memory address of the process, if resident; otherwise, the disk address.
SZ	(l)	The size in blocks of the core image of the process.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
TTY	(all)	The controlling terminal for the process (without the initial "tty", if any).
TIME	(all)	The cumulative execution time for the process (reported in the form "min:sec").
CMD	(all)	The command name; the full command name and its arguments are printed under the **–f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked <**defunct**> (see "zombie process" in *exit*(2)).

Under the **–f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **–f** option, is printed in square brackets.

HARDWARE DEPENDENCIES

Series 500/600/700:

The **F** field is always 01.
In the **S** field, **I** means "waiting for input from terminal".
In the **S** field, the **P** (paused) state is added.
In the **S** field, the **T** state is not currently supported.
The **C** field is always zero.
The **ADDR** field reports the partition number.
In the **SZ** field, the block size is 1K bytes.
The **WCHAN** field is always blank.
The **CMD** field is renamed **COMMAND** except when the **–fl** option is specified.

The definition of **STIME** is as follows:

The time when the process was forked, *not* the time when it was modified by *exec*; the date is included only if the elapsed time is greater than 24 hours.

The **s**, **n**, and **c** options are not currently supported. A diagnostic is printed if they are used.

FILES

/unix	system namelist
/dev/mem	memory
/dev	searched to find swap device and terminal (tty) names

SEE ALSO

kill(1), nice(1), exec(2), exit(2).

BUGS

Things can change while *ps* is running; the picture it gives is only a snapshot in time. Some data printed for defunct processes are irrelevant.

If two special files for terminals are located at the same select code, they are reported in the order in which they appear in /**dev**, not in alphabetical order.

NAME
pwd – working directory name

SYNOPSIS
pwd

HP-UX COMPATIBILITY
Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION
Pwd prints the path name of the working (current) directory.

SEE ALSO
cd(1).

DIAGNOSTICS
"Cannot open .. " and "Read error in .. " indicate possible file system trouble and should be referred to the super-user.

NAME

ranlib - convert archives to random libraries

SYNOPSIS

ranlib archive ...

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	Version 7

DESCRIPTION

Ranlib converts each named *archive* to a form which can be loaded more rapidly by the link editor (*ld*). It does this by adding a table of contents named **__.SYMDEF** to the beginning of the *archive*. It uses *ar*(1) to reconstruct the *archive*, so that sufficient temporary file space must be available in the file system containing the current directory. A library's table of contents must be updated each time that library is modified, or *ld* refuses to use the table of contents. The table of contents is updated by running the library through *ranlib* again.

SEE ALSO

ar(1), ld(1), ranlib(5).

BUGS

Because generation of a library by *ar* and creation of the table of contents by *ranlib* are separate, phase errors are possible. The link editor *ld* warns when the modification date of a library is more recent than the creation of its table of contents, but this means that you get the warning even if you only copy the library.

NAME
revision – get HP-UX revision information

SYNOPSIS
/lbin/revision

HP-UX COMPATIBILITY
Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Revision* is implemented on the Series 500/600/700 only.

DESCRIPTION
This command prints four lines to standard output. Those four lines consist of the four data items output by *uname*(2), which give information on the kernel.

Typical output looks like this:

```
System:     HP-UX
Version:    9030/40
Release:    02.00
Nodename:   hp-esd
```

SEE ALSO
uname(2).

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [**–fri**] file ...

rmdir dir ...

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with **y** the file is deleted, otherwise the file remains. No questions are asked when the **–f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **–r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **–i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **–r**, whether to examine each directory.

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

SEE ALSO

unlink(2).

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file **..** merely to avoid the consequences of inadvertently doing something like:

```
rm –r .*
```

NAME
rmdel – remove a delta from an SCCS file

SYNOPSIS
rmdel **–r**SID files

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the *SID* specified must ***not*** be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get*(1)) exists for the named SCCS file, the *SID* specified must ***not*** appear in any entry of the *p-file*).

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide* Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES
x-file	(see *delta*(1))
z-file	(see *delta*(1))

SEE ALSO
delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide in *HP-UX: Selected Articles.*

DIAGNOSTICS
Use *help*(1) for explanations.

NAME

rmnl - remove extra new-line characters from file

SYNOPSIS

rmnl

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	UCB

DESCRIPTION

Rmnl is useful for removing excess white space from files for display on a crt terminal. Groups of more than one \n character are compressed to one \n character. This is used by the *man* command. *Rmnl* provides the same functionality as UCB *cat –s*.

SEE ALSO

man(1).

NAME

rsh – restricted shell (command interpreter)

SYNOPSIS

rsh [flags] [name [arg1 ...]]

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Rsh is a restricted version of the standard command interpreter *sh*(1). It is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

cd
setting the value of **$PATH**
command names containing /
> and >>

When invoked with the name **–rsh**, *rsh* reads the user's **.profile** (from **$HOME/.profile**). It acts as the standard *sh* while doing this, except that an interrupt causes an immediate exit, instead of causing a return to command level. The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end user shell procedures that have access to the full power of the standard shell, while restricting him to a limited menu of commands; this scheme assumes that the end user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions, then leaving the user in an appropriate directory (probably *not* the login directory).

Rsh is actually just a link to *sh* and any *flags* arguments are the same as for *sh*(1).

The system administrator often sets up a directory of commands that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

SEE ALSO

sh(1), profile(5).

BUGS

An illegal command (such as *cd*) included in a trap 0 (logout) will cause a memory fault in the shell.

NAME

sact – print current SCCS file editing activity

SYNOPSIS

sact files

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **–e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. Non-SCCS files and unreadable files are ignored. The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (i.e. executed a *get* for editing).
Field 4	contains the date that **get –e** was executed.
Field 5	contains the time that **get –e** was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sccsdiff – compare two versions of SCCS file

SYNOPSIS

sccsdiff –**r**SID1 –**r**SID2 [–**p**] [–**s**n] files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

–**r***SID?*	*SID*1 and *SID*2 specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.
–**p**	pipe output for each file through *pr*(1).
–**s***n*	*n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), diff(1), get(1), help(1), pr(1).
Source Code Control System User's Guide in *HP-UX: Selected Articles.*

DIAGNOSTICS

"*File*: No differences" if the two versions are the same.
Use *help*(1) for explanations.

NAME
sed – stream text editor

SYNOPSIS
sed [**–n**] [**–e** script] [**–f** sfile] [files]

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION
Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **–f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **–e** option and no **–f** options, the flag **–e** may be omitted. The **–n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed*(1) modified thus:

In a context address, the construction *?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line *embedded* in the pattern space.

A period **.** matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**\	
text	Append. Place *text* on the output before reading the next input line.
(2) **b** *label*	Branch to the **:** command bearing the *label*. If *label* is empty, branch to the end of the script.
(2) **c**\	
text	Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.
(2) **d**	Delete the pattern space. Start the next cycle.
(2) **D**	Delete the initial segment of the pattern space through the first new-line. Start the next

cycle.

(2) **g**	Replace the contents of the pattern space by the contents of the hold space.
(2) **G**	Append the contents of the hold space to the pattern space.
(2) **h**	Replace the contents of the hold space by the contents of the pattern space.
(2) **H**	Append the contents of the pattern space to the hold space.
(1) **i**\	
text	Insert. Place *text* on the standard output.
(2) **l**	List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.
(2) **n**	Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
(2) **N**	Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)
(2) **p**	Print. Copy the pattern space to the standard output.
(2) **P**	Copy the initial segment of the pattern space through the first new-line to the standard output.
(1) **q**	Quit. Branch to the end of the script. Do not start a new cycle.
(2) **r** *rfile*	Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

g	Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
p	Print the pattern space if a replacement was made.
w *wfile*	Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label*	Test. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.
(2) **w** *wfile*	Write. Append the pattern space to *wfile*.
(2) **x**	Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/

Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function*

Don't. Apply the *function* (or group, if *function* is {) only to lines *not* selected by the address(es).

(0) **:** *label*	This command does nothing; it bears a *label* for **b** and **t** commands to branch to.
(1) =	Place the current line number on the standard output as a line.
(2) {	Execute the following commands through a matching } only when the pattern space is selected. The syntax is:

```
{ cmd1
cmd2
cmd3
.
.
.
}
```

(0) An empty command is ignored.

SEE ALSO

awk(1), ed(1), grep(1).

BUGS

There is a limit of 100 commands in the script.

NAME

sh – shell, the standard command programming language

SYNOPSIS

sh [**–ceiknrstuvx**] [args]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See *Invocation* below for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non-blank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200 + *status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by vertical bars (|). The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word* . . .] **do** *list* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* . . . is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [*pattern*] . . .) *list* **;;**] . . . **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation* below).

if *list* **then** *list* [**elif** *list* **then** *list*] . . . [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

list is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments.
A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution.
The standard output from a command enclosed in a pair of grave accents (\`\`) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution.
The character **$** is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

name = *value* [*name* = *value*] . . .

Pattern-matching is not performed on *value*.

${*parameter*}
A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *****, **@**, **#**, **?**, **–**, **$**, and **!**. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is ***** or **@**, then all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:–*word*}
If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

${*parameter*: = *word*}
If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, then the message "parameter null or not set" is printed.

${*parameter*: + *word*}
If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, *pwd* is executed only if **d** is not set or is null:

```
echo ${d:–`pwd`}
```

If the colon (**:**) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
–	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME	The default argument (home directory) for the *cd* command.
PATH	The search path for commands (see *Execution* below).
MAIL	If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.
SHELL	a string specifying which shell to run.
PS1	Primary prompt string, by default "**$** ".
PS2	Secondary prompt string, by default "> ".

IFS	Internal field separators, normally **space**, **tab**, and **new-line**.

The shell gives default values to **PATH**, **PS1**, **PS2**, and **IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** *is* set by *login*(1)).

Blank Interpretation.
After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation.
Following substitution, each command *word* is scanned for the characters *****, **?**, and **[**. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character **.** at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. A NOT operator, **!**, can be specified immediately following the left bracket to match any single character *not* enclosed in the brackets.

Quoting.
The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **new-line** is ignored. All characters enclosed between a pair of single quote marks (´´) except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, `, ", and **$**. "**$***" is equivalent to "**$1 $2** ...", whereas "**$@**" is equivalent to "**$1**" "**$2**" ....

Prompting.
When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output.
Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<word	Use file *word* as standard input (file descriptor 0).
>word	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
>>word	Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<[–] word	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **new-line** is ignored, and \ must be used to quote the characters \, **$**, `, and the first character of *word*. If – is appended to <<, then all leading tabs are stripped from *word* and from the document.
<**&**digit	The standard input is duplicated from file descriptor *digit* (see *dup*(2)). Similarly for

	the standard output using >.
<&–	The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by **&** then the default standard input for the command is the empty file /**dev**/**null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment.

The *environment* (see *environ*(7)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd args                      and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the –**k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c:**

```
echo a=b c
set –k
echo a=b c
```

Signals.

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution.

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying the current directory, /**bin**, and /**usr**/**bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

Special Commands.

The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*. Note that this command does not spawn another shell to execute *file*, and thus differs in behavior and output from executing *file* as a shell script.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*.

eval [*arg* . . .]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* . . .]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* . . .]
The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, then a list of all names that are exported in this shell is printed.

newgrp [*arg* . . .]
Equivalent to **exec newgrp** *arg* . . . .

read [*name* . . .]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* . . .]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, then a list of all *readonly* names is printed.

set [**–ekntuvx** [*arg* . . .]]

–e	If the shell is non-interactive then exit immediately if a command exits with a non-zero exit status.
–k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
–n	Read commands but do not execute them.
–t	Exit after reading and executing one command.
–u	Treat unset variables as an error when substituting.
–v	Print shell input lines as they are read.
–x	Print commands and their arguments as they are executed.
––	Do not change any of the flags; useful in setting $1 to –.

Using + rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given then the values of all names are printed.

shift
The positional parameters from **$2** . . . are renamed **$1** . . . .

test
Evaluate conditional expressions. See *test*(1) for usage and description.

times
Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] . . .
: *arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *n* is 0 then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

umask [*nnn*]
: The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait
: Wait for all child processes to terminate, and report the termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is always zero.

Invocation.
If the shell is invoked through *exec*(2) and the first character of argument zero is –, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; Note that unless the **–c** or **–s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

–c *string* If the **–c** flag is present then commands are read from *string*.

–s If the **–s** flag is present or if no arguments remain then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

–i If the **–i** flag is present or if the shell input and output are attached to a terminal, then this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

–r If the **–r** flag is present the shell is a restricted shell (see *rsh*(1)).

The remaining flags and arguments are described under the **set** command above.

If the **SHELL** parameter appears in the environment, and the first character to the right of the right-most slash is an "r", then the shell becomes restricted. The following lines should be included in the **.profile** files of restricted login names:

```
SHELL=/usr/rsh
export SHELL
```

This causes whoever logs in under the restricted login names to be given a restricted shell (see *rsh*(1)).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

RETURN VALUE

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

SEE ALSO

cd(1), env(1), login(1), newgrp(1), rsh(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(5), profile(5), environ(7).

BUGS

The command *readonly* (without arguments) produces the same output as the command *export*.

If << is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

^ is a synonym for |; the use of ^ is discouraged.

The shell assumes it is talking to terminals that only process the least significant seven bits of a character. If your terminal uses all eight bits, you may see some strange strings.

When the shell encounters >>, it does not open the file in append mode. Instead, it opens the file for writing and seeks to the end.

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(2), sleep(3C).

BUGS

Time must be less than 2^32 seconds.

NAME

sort – sort and/or merge files

SYNOPSIS

sort [–**cmubdfinrTt**x] [+pos1 [–pos2]] . . . [–**o** output] [file ...]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name – means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d "Dictionary" order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b**.

r Reverse the sense of comparisons.

t*x* "Tab character" separating fields is *x*.

The notation +*pos1* –*pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first non-blank in the field; **b** is attached independently to *pos2*. A missing *.n* means .0; a missing –*pos2* means the end of the line. Under the –**t***x* option, fields are strings separated by *x*; otherwise fields are non-empty non-blank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

T The next argument is the name of a directory where *sort* should put its temporary files.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words (capitalized words differ from uncapitalized):

```
sort –u  +0f  +0 list
```

Print the password file (*passwd*(5)) sorted by user ID (the third colon-separated field):

```
sort –t:  +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month-day) entries (the options **–um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort –um +0 –1 dates
```

FILES

/usr/tmp/stm*	default temporary files
/tmp/stm*	

SEE ALSO

comm(1), join(1), uniq(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions and for disorder discovered under option **–c**.

BUGS

Lines that exceed 512 characters (including the new-line) are silently truncated (i.e., the excess characters are ignored).

Sort will not recognize the line in a file if it is not terminated with a new-line character.

Sort uses signed characters for comparison (unlike *strcmp*(3)), so you may get unexpected results when sorting a file that contains ASCII characters with the most significant bit set (characters 128 – 255).

Sort does not understand "missing" fields. For example, if you have a file with the following contents:

```
Doe, John     mailman    17550   8
Spencer, Joe  plumber            4
Johns, Ann    secretary  15950
Malley, Dean  engineer   26750   4
```

you may get unexpected results if you try to sort on the third or fourth fields (all names and associated data are fictitious).

Sort does not expand tabs when counting characters to locate a field.

NAME

spell, spellin, spellout – find spelling errors

SYNOPSIS

spell [options] [files]

/**usr**/**lib**/**spell**/**spellin** [list]

/**usr**/**lib**/**spell**/**spellout** [**–d**] list

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the **–v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **–b** option, British spelling is checked. Besides preferring *centre*, *colour*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the **–x** option, every plausible stem is printed with = for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier = thy–y + ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell* (both expect a list of words, one per line, from the standard input): *spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word read from the standard input, and prints on the standard output those that are missing from (or, with the **–d** option, present in) the hash list.

FILES

D_SPELL = /usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL = /usr/lib/spell/hstop	hashed stop list
H_SPELL = /usr/lib/spell/spellhist	history file
/tmp/spell.$$	temporary
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1), typo(1).

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local dictionary that is added to the hashed *list* via *spellin*.
British spelling was done by an American.

NAME

split – split a file into pieces

SYNOPSIS

split [*–n*] [file [name]]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if – is given instead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

stty – set the options for a terminal port

SYNOPSIS

stty [**–a**] [**–g**] [options]

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **–a** option, it reports all of the option settings; with the **–g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *tty*(4). Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sense checking is performed. The options are selected from the following:

Control Modes

parenb (**–parenb**)	enable (disable) parity generation and detection.
parodd (**–parodd**)	select odd (even) parity.
cs5 cs6 cs7 cs8	select character size (see *tty*(4)).
0	hang up phone line immediately.
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb	Set terminal baud rate to the number given, if the hardware will support it.
hupcl (**–hupcl**)	hang up (do not hang up) modem connection on last close.
hup (**–hup**)	same as **hupcl** (**–hupcl**).
cstopb (**–cstopb**)	use two (one) stop bits per character.
cread (**–cread**)	enable (disable) the receiver.
crts (**–crts**)	enable (disable) request-to-send.
clocal (**–clocal**)	assume a line without (with) modem control.

Input Modes

ignbrk (**–ignbrk**)	ignore (do not ignore) break on input.
ienqak (**–ienqak**)	enable (disable) ENQ-ACK handshaking.
brkint (**–brkint**)	signal (do not signal) INTR on break.
ignpar (**–ignpar**)	ignore (do not ignore) parity errors.
parmrk (**–parmrk**)	mark (do not mark) parity errors (see *tty*(4)).
inpck (**–inpck**)	enable (disable) input parity checking.
istrip (**–istrip**)	strip (do not strip) input characters to seven bits.
inlcr (**–inlcr**)	map (do not map) NL to CR on input.
igncr (**–igncr**)	ignore (do not ignore) CR on input.
icrnl (**–icrnl**)	map (do not map) CR to NL on input.
iuclc (**–iuclc**)	map (do not map) upper-case alphabetics to lower case on input.
ixon (**–ixon**)	enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
ixany (**–ixany**)	allow any character (only DC1) to restart output.

ixoff (**–ixoff**)	request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.
Output Modes	
opost (**–opost**)	post-process output (do not post-process output; ignore all other output modes).
olcuc (**–olcuc**)	map (do not map) lower-case alphabetics to upper case on output.
onlcr (**–onlcr**)	map (do not map) NL to CR-NL on output.
ocrnl (**–ocrnl**)	map (do not map) CR to NL on output.
onocr (**–onocr**)	do not (do) output CRs at column zero.
onlret (**–onlret**)	on the terminal NL performs (does not perform) the CR function.
ofill (**–ofill**)	use fill characters (use timing) for delays.
ofdel (**–ofdel**)	fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	select style of delay for carriage returns (see *tty*(4)).
nl0 nl1	select style of delay for line-feeds (see *tty*(4)).
tab0 tab1 tab2 tab3	select style of delay for horizontal tabs (see *tty*(4)).
bs0 bs1	select style of delay for backspaces (see *tty*(4)).
ff0 ff1	select style of delay for form-feeds (see *tty*(4)).
vt0 vt1	select style of delay for vertical tabs (see *tty*(4)).
Local Modes	
isig (**–isig**)	enable (disable) the checking of characters against the special control characters INTR and QUIT.
icanon (**–icanon**)	enable (disable) canonical input (ERASE and KILL processing).
xcase (**–xcase**)	canonical (unprocessed) upper/lower-case presentation.
echo (**–echo**)	echo back (do not echo back) every character typed.
echoe (**–echoe**)	echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (**–echok**)	echo (do not echo) NL after KILL character.
lfkc (**–lfkc**)	the same as **echok** (**–echok**); obsolete.
echonl (**–echonl**)	echo (do not echo) NL.
noflsh (**–noflsh**)	disable (enable) flush after INTR or QUIT.
Control Assignments	
control-character c	set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **min**, or **time** (**min** and **time** are used with **–icanon**; see *tty*(4)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., ^d is a **CTRL-d**); ^? is interpreted as DEL and ^– is interpreted as undefined.
line *i*	set line discipline to *i* ($0 < i < 127$). (See tty(4).)
Combination Modes	
evenp or **parity**	enable **parenb** and **cs7**.
oddp	enable **parenb**, **cs7**, and **parodd**.

–parity, **–evenp**, or **–oddp**	disable **parenb**, and set **cs8**.
raw (**–raw** or **cooked**)	enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).
nl (**–nl**)	unset (set) **icrnl**, **onlcr**. In addition **–nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (**–lcase**)	set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (–LCASE)	same as **lcase** (**–lcase**).
tabs (**–tabs** or **tab3**)	preserve (expand to spaces) tabs when printing.
ek	reset ERASE and KILL characters back to normal # and @.
sane	resets all modes to some reasonable values.
term	set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, **hp**, or **tek**.

HARDWARE DEPENDENCIES

Series 500/600/700:

Refer to *tty*(4) for a description of the capabilities that are not supported.

SEE ALSO

tabs(1), ioctl(2), tty(4).

NAME

su – become another user

SYNOPSIS

su [–] [name [arg . . .]]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already super-user). If the password is correct, *su* will execute a new shell with the user ID set to that of the specified user. To restore normal user ID privileges, type an **EOF** to the new shell.

Any additional arguments are passed to the shell, permitting execution of the shell procedures with restricted privileges (an *arg* of the form **–c** *string* executes *string* via the shell). After the procedure has executed, the new user is logged out, and the original user becomes the current user. When additional arguments are passed, **/bin/sh** is always used. When no additional arguments are passed, *su* uses the shell specified in the password file.

An initial – flag causes the environment to be changed to the one that would be expected if the user actually logged in again. This is done by invoking the shell with an *arg0* of **–su** causing the **.profile** in the home directory of the new user ID to be executed. Otherwise, the environment is passed along unchanged. The super-user's **$PATH** is unconditionally set to **/bin:/etc:/usr/bin**. Note that the **.profile** can check *arg0* for **–sh** or **–su** to determine how it was invoked.

The – option always resets **$PATH** to **/bin:/etc:/usr/bin** for the super-user, and **/bin:/usr/bin** for all others. However, the files */etc/profile* and *.profile* are normally executed anyway, thus restoring the intended value of **$PATH**.

Su will ensure that the super-user gets a " # " prompt to remind him of his additional responsibilities.

Su logs all attempts to *su* in /usr/adm/sulog, including failures. Successful attempts are flagged with " + ", failures with " - ".

FILES

/etc/passwd	system's password file
$HOME/.profile	user's profile

SEE ALSO

env(1), login(1), sh(1), environ(7).

NAME

tabs – set tabs on a terminal

SYNOPSIS

tabs [tabspec] [+**m**n] [–**T**type]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs.

If you are using a non-HP terminal, you should keep in mind that behavior will vary for some tab settings.

Four types of tab specification are accepted for *tabspec*: "canned", repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **–8**, i.e., HP-UX "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0.

–code Gives the name of one of a set of "canned" tabs. The legal *code*s and their meanings are as follows:

–a 1,10,16,36,72
Assembler, IBM S/370, first format

–a2 1,10,16,40,72
Assembler, IBM S/370, second format

–c 1,8,12,16,20,55
COBOL, normal format

–c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:
<:t–c2 m6 s66 d:>

–c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **–c2.** This is the recommended format for COBOL. The appropriate format specification is:
<:t–c3 m6 s66 d:>

–f 1,7,11,15,19,23
FORTRAN

–p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

–s 1,10,55
SNOBOL

–u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

–n A repetitive specification requests tabs at columns $1+n$, $1+2*n$, etc. Of particular importance is the value **–8**: this represents the HP-UX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff*(1) **–h** option for high-speed output. Another special case is the value **–0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10, + 10, + 10 are considered identical.

--file If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

```
tabs -- file; pr file
```

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-T*type* *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(7). If no -T flag is supplied, *tabs* searches for the **$TERM** value in the *environment* (see *environ*(7)). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+**m***n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If +**m** is given without a value of *n*, the value assumed is 10. The normal (leftmost) margin on most terminals is obtained by +**m0**. The margin for most terminals is reset only when the +**m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

SEE ALSO

nroff(1), tset(1), environ(7), term(7).

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly.
illegal increment	when a zero or missing increment is found in an arbitrary specification.
unknown tab code	when a "canned" code cannot be found.
can't open	if *--file* option used, and file can't be opened.
file indirection	if *--file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.
It is generally impossible to usefully change the left margin without also setting tabs.
Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [[±[number][**lbc**][**f**]] | [**–f**]] [file]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +*number* from the beginning, or –*number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **–f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process.

EXAMPLES

Tail accepts at most two arguments: the first consists of specified options, and the second specifies the file of interest. If the *number* and **f** options are both desired, they must be concatenated to create a single option argument, as follows:

```
tail –3lf john
```

This example prints the last three lines in the file *john* to the standard output, and leaves *tail* in "follow" mode.

If only the **f** option is desired, it must be preceded by a –, as follows:

```
tail –f fred
```

This example prints the last ten lines of the file *fred*, followed by any lines that are appended to *fred* between the time *tail* is initiated and killed. Note that this output may build up very quickly for rapidly changing input files, perhaps too fast to read on a CRT.

SEE ALSO

dd(1).

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Thus, be wary of the results when piping output from other commands into *tail.*

Various kinds of anomalous behavior may happen with character special files.

Tail can pick up a maximum of 4K bytes of data from the specified file.

NAME

tar – tape file archiver

SYNOPSIS

tar [key] [files]

HP-UX COMPATIBILITY

Level: HP-UX/DEVELOPMENT

Origin: System III and UCB

DESCRIPTION

Tar saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. The *key* string may be preceded by a dash (–) (similar to the way options are specified in other HP-UX commands), but it is not necessary. Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the *key* is specified by one of the following letters:

r The named *files* are written on the end of the archive. The **c** function implies this function.

x The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t The names of all files are listed each time that they occur on the tape.

u The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape.

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file.

The following function modifiers may be used in addition to the function letters listed above:

0,...,8 This modifier selects the drive on which the tape is mounted. The default is **8** (as /dev/rmt8).

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no".

f causes *tar* to use the next argument as the name of the archive instead of **/dev/rmt?**. If the name of the file is –, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf – . (cd todir; tar xf –)
```

b causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (*key* letters **x** and **t**).

l tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m tells *tar* to not restore the modification times. The modification time of the file will be the time of extraction.

o On output, tar normally places information specifying owners and modes of directories in the archive. Former versions of tar, when encountering this information, gave error messages of the form

"<name>/: cannot create".

This option will suppress the directory information.

p This option says to restore files to their original modes, ignoring the present *umask*(2). *Setuid* and sticky information will also be restored to the super-user.

If a tape drive is used as the output device, it must be configured in Berkeley compatability mode; see *mt*(4).

EXAMPLE

```
tar cvfb /dev/rmf0 20 file1 file2
```

This example creates a new tape on /dev/rmf0 and copies file1 and file2 onto it, using a blocking factor of 20. The *key* is made up of one function letter (**c**) and three function modifiers (**v**, **f**, and **b**). Note that the arguments required by the **f** and **b** modifiers must be specified in the same order in which the modifiers are specified in the *key*.

FILES

/dev/rmt?
/tmp/tar*

SEE ALSO

ar(1), cpio(1), mt(4).

DIAGNOSTICS

Complains about bad key characters and tape read/write errors.
Complains if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
If the archive is on a disc file, the **b** option should not ever be used, because updating an archive stored on disc can destroy it.
The current limit on file-name length is 100 characters.
Some previous versions of tar have claimed to support selective listing of file names using the -t option with a list. To our knowledge this was an error in the documentation and does not appear in the original source code.
There is no way to restore an absolute path name to a relative position.

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [–**TX**] [files]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Tbl is a preprocessor that formats tables for *nroff*(1) or *troff*(1). The input files are copied to the standard output, except for lines between **.TS** and **.TE** command lines, which are assumed to describe tables and are re-formatted by *tbl*. (The **.TS** and **.TE** command lines are not altered by *tbl*).

.TS is followed by global options. The available global options are:

center	center the table (default is left-adjust);
expand	make the table as wide as the current line length;
box	enclose the table in a box;
doublebox	enclose the table in a double box;
allbox	enclose each item of the table in a box;
tab (*x*)	use the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semicolon (**;**).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, indicate where vertical bars are to appear between columns, determine column width, inter-column spacing, etc. The available key-letters are:

c	center item within the column;
r	right-adjust item within the column;
l	left-adjust item within the column;
n	numerically adjust item in the column: units positions of numbers are aligned vertically;
s	span previous item on the left into this column;
a	center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
^	span down previous entry in this column;
_	replace this entry with a horizontal line;
=	replace this entry with a double horizontal line.

The characters **B** and **I** stand for the bold and italic fonts, respectively; the character | indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by **.TE**. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ or =, a single or double line, respectively, is drawn across the table at that point. If a *single item* in a data line consists of only _ or =, then that item is replaced by a single or double line.

Full details of all these and other features of *tbl* are given in the reference manual cited below.

The –**TX** option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments, *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn*(1) or *neqn*(1), *tbl* should come first to minimize the volume of data passed through pipes.

EXAMPLE

If we let → represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population
_
Town→Households
→Number→Size
=
Bedminster→789→3.26
Bernards Twp.→3087→3.74
Bernardsville→2018→3.30
Bound Brook→3425→3.04
Bridgewater→7897→3.81
Far Hills→240→3.19
.TE
```

yields:

Household		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

TBL – A Program to Format Tables in *HP-UX: Selected Articles*.
eqn(1), mm(1), mmt(1), troff(1), mm(7), mv(7).

BUGS

See *BUGS* under *troff*(1).

NAME
tcio – 79XX tape cartridge utility

SYNOPSIS
/lbin/tcio –o [**vdVS**] [buffersize] filename
/lbin/tcio –i [**vdS**] [buffersize] filename
/lbin/tcio –u [**vVmrc**] [blocknumber] [**save** | **restore**] filename [disc_filename]

HP-UX COMPATIBILITY
Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: *Tcio* is implemented on the Series 500/600/700 only.

DESCRIPTION
Tcio –o (copy out) reads the standard input and writes the data to the raw cartridge tape specified by *filename.*

Tcio –i (copy in) reads the raw cartridge tape specified by *filename* and writes the data to the standard output.

Tcio –u (utility) performs utility functions on the cartridge tape, such as image backup and restore, release, mark, and/or verify cartridge.

Tcio puts data to be written to a cartridge tape in large buffers, saving wear and tear on the media and the mechanism. It also puts a tape mark in the first record on tape to prevent the tape from being image restored over a disc. One of the options **o**, **i**, or **u** must be specified. The meanings of the available modifiers are:

v Verbose mode; prints information and error messages to *stderr.*

d Prints a checksum to *stderr.* The checksum is a 32-bit unsigned addition of the bytes. This checksum provides an extra check of the validity of the tape in addition to tape verification. The value should be recorded on the media as it is only reported to the user; it is not written to the media. The checksum is valid only for the **i** and **o** options, and if the same number of bytes are read from the tape as were written to it. This option overrides the setting of the verbose modifier.

S Enables specification of buffer size. This option forces the allocation of a block of memory to be used to write to the tape. The size of the buffer is 1024 times the value specified for *buffersize.* A *buffersize* less than 32 will cause the program to terminate. A *buffersize* greater than 512 may cause unpredictable results. If *buffersize* is not specified, *tcio* will attempt buffer sizes from 512 down to 64. The primary uses of this option are to allow buffer sizes smaller than 64 Kbytes, and to allow the user to pick a buffer size that is most suitable for his application.

V This option turns off tape verification. It is suggested that this option not be used, for the sake of the integrity of the data output to tape.

m This option writes a tape mark on a tape at the specified block. If a tape is created by some other means than *tcio,* a tape mark in block 0 of the tape will prevent it from being image restored to a disc. Note that *blocknumber* must be specified.

r Releases the tape from the mechanism, unlocking the door.

c Image copy provides the same capability as the push-button save and restore available in the HP 79XX single controller drive. The **save** and **restore** keywords are the same as the labels on those switches. **Save** implies disc to tape; **restore** implies tape to disc. Currently only single controller disc/tape units can be backed up in this way.

SEE ALSO
cpio(1).

WARNING
To be able to use the save/restore facility, the following two conditions must be met:

your system must be in single-user mode;

you must never have used networking on your system. If networking has been used on your system, you must reboot the system before using the save/restore facility.

A "media misload" error (*errinfo* value 142) is given if a faulty tape or dirty mechanism is encountered.

NAME

tee – pipe fitting

SYNOPSIS

tee [**–i**] [**–a**] [file] ...

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The **–i** option ignores interrupts; the **–a** option causes the output to be appended to the *files* rather than overwriting them.

NAME

test, [– condition evaluation command

SYNOPSIS

test expr
[expr]

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

–r *file*	true if *file* exists and is readable.
–w *file*	true if *file* exists and is writable.
–x *file*	true if *file* exists and is executable.
–f *file*	true if *file* exists and is a regular file.
–d *file*	true if *file* exists and is a directory.
–c *file*	true if *file* exists and is a character special file.
–b *file*	true if *file* exists and is a block special file.
–u *file*	true if *file* exists and its set-user-ID bit is set.
–g *file*	true if *file* exists and its set-group-ID bit is set.
–k *file*	true if *file* exists and its sticky bit is set.
–s *file*	true if *file* exists and has a size greater than zero.
–t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–z *s1*	true if the length of string *s1* is zero.
–n *s1*	true if the length of the string *s1* is non-zero.
s1 = *s2*	true if strings *s1* and *s2* are identical.
s1 != *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 **–eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **–ne**, **–gt**, **–ge**, **–lt**, and **–le** may be used in place of **–eq**.

These primaries may be combined with the following operators:

!	unary negation operator.
–a	binary *and* operator.
–o	binary *or* operator (**–a** has higher precedence than **–o**).
(expr)	parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

Test is directly interpreted by the shell.

SEE ALSO

eval(1), find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses [ ], rather than the word *test*), the square brackets must *delimited* by by blanks.

NAME

time – time a command

SYNOPSIS

time command

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

The given command is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The execution time can depend on the performance of the memory in which the program is running.

The times are printed on standard error.

HARDWARE DEPENDENCIES

Series 600/700:

The child CPU times listed may be greater than the actual real elapsed time, since CPU time is counted on a per-CPU basis. Thus, if three CPUs are executing, the times listed are obtained by adding the execution times of each CPU.

SEE ALSO

times command in sh(1), timex(1), times(2).

NAME

touch – update access/modification/change times of file

SYNOPSIS

touch [–**amc**] [mmddhhmm[yy]] files

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Touch causes the access, modification, and last change times of each argument to be updated. If no time is specified (see *date*(1)) the current time is used. The –**a** and –**m** options cause touch to update only the access or modification times respectively (default is –**am**). The –**c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1), utime(2).

NAME

tr – translate characters

SYNOPSIS

tr [**–cds**] [string1 [string2]]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **–cds** may be used:

–c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

–d Deletes all input characters in *string1*.

–s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a–z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a**n*] Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

The following creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr –cs "[A–Z][a–z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1), ascii(7).

BUGS

Won't handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

NAME
true, false – provide truth values

SYNOPSIS
true

false

HP-UX COMPATIBILITY
Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION
True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true do
        command
done
```

SEE ALSO
sh(1).

DIAGNOSTICS
True has exit status zero, *false* nonzero.

NAME
tsort – topological sort

SYNOPSIS
tsort [file]

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION
Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO
lorder(1).

DIAGNOSTICS
Odd data: there is an odd number of fields in the input file.

BUGS
Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME
tty – get the terminal's name

SYNOPSIS
tty [**–s**]

HP-UX COMPATIBILITY
Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION
Tty prints the path name of the user's terminal. The **–s** option inhibits printing, allowing one to test just the exit code.

RETURN VALUE
0	if standard input is a terminal,
1	otherwise.

DIAGNOSTICS
"not a tty" if the standard input is not a terminal and **–s** is not specified.

NAME

ul – do underlining

SYNOPSIS

ul [**–i**] [**–t** terminal] [name ...]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	UCB

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable **TERM**. The **–t** option overrides the terminal kind specified in the environment. The file */etc/termcap* is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

The **–i** option causes *ul* to indicate underlining on a crt by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1).

BUGS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [ooo]

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The octal three digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed with four octal digits. The first digit, a zero, specifies that the output is expressed in octal.

Umask is recognized and executed by the shell.

Note that the file creation mask does not affect the set-user-ID, set-group-ID, or "sticky" bits.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

NAME

uname – print name of current HP-UX version

SYNOPSIS

uname [**–snrvmia**]

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Uname prints the current system name of HP-UX on the standard output file. It is mainly useful to determine what system you are using. The options cause selected information returned by *uname*(2) to be printed:

–s	print the system name (default).
–n	print the nodename (the nodename is a name that the system is known by on a communications network). (e.g. *uucp*).
–r	print the operating system release.
–v	print the operating system version.
–m	print the machine hardware name.
–i	print the machine identification number.
–a	print all the above information.

SEE ALSO

uname(2).

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

unget [–**r**SID] [–**s**] [–**n**] files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Unget undoes the effect of a **get –e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (see *sact*(1)) and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed (see *sact*(1)). Keyletter arguments apply independently to each named file.

–**r***SID*	Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.
–**s**	Suppresses the printout, on the standard output, of the intended delta's *SID*.
–**n**	Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [–**udc** [+n] [–n]] [input [output]]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the –**u** flag is used, just the lines that are not repeated in the original file are output. The –**d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the –**u** and –**d** mode outputs.

The –**c** option supercedes –**u** and –**d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

upm – unpack cpio archives from HP media

SYNOPSIS

upm –E [**BcdDmPrstuvx6**] pathname chardevice [patterns]

upm –iM [**BcdDmPrstuvx6**] [patterns] </dev/rmf?

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Upm* is implemented on the Series 500/600/700 only.

DESCRIPTION

Upm is similar to *cpio*(1), and is included to enable you to restore files from 88140L/S tape cartridges or 5.25-inch flexible discs more efficiently.

Upm –E (copy in from tape cartridge) extracts all files specified by *patterns* from the file named by *pathname* (assumed to be the product of a previous *cpio –o*). *Patterns* is a series of zero or more blank-separated character strings given in the name-generating notation of *sh*(1). Note that the metacharacters ?, *, and [...] match the slash (/) when used in *patterns*. The default *pattern* is '*', which selects all files. *Chardevice* identifies the character special device file describing the volume containing *pathname*. (Note that, if this volume is not the root, it must be mounted at the time *upm* is used, and *pathname* must include the directory name on which the volume is mounted.)

Upm –iM (copy in) extracts all files selected by zero or more of the specified *patterns* (see above for a description of *patterns*). The files are extracted from the standard input, which is redirected from a raw miniature flexible disc device /dev/rmf?. The resulting standard input is assumed to be the product of a previous *cpio –o*.

Any other options specified must be concatenated with the initial *E* or *iM* options. The options have the following meanings:

B	I/O is to be blocked 5120 bytes to the record (recommended only with data read from /dev/rmt?);
c	read header information which was previously written in ASCII character form for portability;
d	directories are to be created as needed;
D	print debugging information to the standard output as the command executes;
m	retain previous file modification time. This option is ineffective on directories that are being copied;
P	read a file written on a PDP–11 or VAX system (with byte swapping) that did not use the **–c** option. Only bytes contained in the header are swapped. Non-ASCII files will probably need further processing to be readable; this processing requires knowledge of the content of the file and thus cannot be done by this program. (PDP–11 and VAX are registered trademarks of Digital Equipment Corporation.)
r	interactively rename files; if the user types a null line, the file is skipped;
s	identical to the **P** option, except that all bytes in the file are swapped (including the header);
t	print a table of contents of the input; no files are created;
u	copy unconditionally (normally, an older file will not replace a newer file with the same name);
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));
x	restore device special files; *mknod*(2) is used to recreate these files, and thus **–Ex** or **–iMx** can only be used by the super-user. Restoring device files onto a different system can be very dangerous. This is intended for backup use;
6	process an old (version 6 format) file.

When the end of a volume is reached, *upm* will prompt the user for the next flexible disc and continue.

The number of blocks reported by ***upm*** is always in units of 512-byte blocks, regardless of the block size of the initialized media.

SEE ALSO

cpio(1), tcio(1), mknod(2).

WARNING

The **–B** option must not be used when performing raw I/O using the HP 9130K miniature flexible disc drive.

BUGS

Only the super-user can copy special files.

If /dev/tty is not accessible, ***upm*** issues a complaint, or refuses to work.

The **–Edr** and **–iMdr** options will not make empty directories.

NAME
val – validate SCCS file

SYNOPSIS
val –
val [–**s**] [–**r**SID] [–**m**name] [–**y**type] files

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a –, and named files.

Val has a special argument, –, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

–**s**	The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.
–**r***SID*	The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e. g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., **r**1.0 and **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.
–**m***name*	The argument value *name* is compared with the SCCS %M% keyword in *file*.
–**y***type*	The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = can't open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, –**y** mismatch;
bit 7 = %M%, –**m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned – a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO
admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS
Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a fatal error.

NAME

vi, view – visual text editor

SYNOPSIS

vi [**–t** tag] [**–r**] [+ command] [**–l**] [**–w**n] name ...

view [vi options]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	UCB
Remarks:	William Joy wrote the original *vi* editor. Mark Horton added macros to *visual* mode.

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*(1). *Ex* and *vi* run the same code; it is possible to get to the command mode of *ex* from within *vi*, and vice-versa. *View* is a read-only version of *vi*. The following options are recognized:

–t tag equivalent to an initial **tag** command. The file containing the *tag* is edited, with the editor positioned at the definition of the *tag*.

–r used to recover from the last editor or system crash. The last saved version of *name* is retrieved, or, if no *name* is specified, a list of saved files is printed.

+ command
The editor begins the editing session by executing the specified editor command. If *command* is omitted, it defaults to **$**, positioning the editor at the last line of the current file.

–l automatically sets the **showmatch** and **lisp** options.

–wn sets the default window size to *n* lines.

The Vi Editor provides full details on using *vi*.

FILES

See *ex*(1).

SEE ALSO

ex(1), *The Vi Editor* in *HP-UX: Selected Articles.*

BUGS

Software tabs using ^**T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line will not be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open*.

The **CTRL-F** command does not work if ENQ-ACK handshake is enabled.

If the system crashes or *vi* is killed, the terminal may be left in raw mode. To get out of raw mode, type the following sequence:

CTRL–j
stty hp
CTRL–j

This sequence may alter the baud rate; see *stty*(1).

NAME

wait – await completion of process

SYNOPSIS

wait

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

NAME
wall – write to all users

SYNOPSIS
/etc/wall

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Wall reads its standard input until an end-of-file. It then sends this message to all currently logged in users preceded by "Broadcast Message from ...". It is used to warn all users, typically prior to shutting down the system.

The sender should be super-user to override any protections the users may have invoked.

Wall has timing delays, and will take at least 30 seconds to complete.

FILES
/dev/tty*

SEE ALSO
mesg(1), write(1).

DIAGNOSTICS
"Cannot send to ..." when the open on a user's tty file fails.

NAME

wc – word, line, and character count

SYNOPSIS

wc [–**lwc**] [names]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is –**lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

BUGS

Wc counts the number of new-lines to determine the line count. If an ASCII text file has a final line that is not terminated with a new-line character, the count will be off by one.

If there are very many characters, words, and/or lines in an input file, the output may be hard to read. This is because *wc* reserves a fixed column width for each count.

NAME

what – identify files for SCCS information

SYNOPSIS

what files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = " @(#)identification information ";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

whereis – locate source, binary, and/or manual for program

SYNOPSIS

whereis [**–sbm**] [**–u**] [**–SBM** dir ... **–f**] name ...

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	UCB

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". Prefixes of "s." resulting from use of SCCS are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places. If any of the **–b, –s** or **–m** flags are given then *whereis* searches only for binaries, sources or manual sections respectively (or any two thereof). The **–u** flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u * " asks for those files in the current directory which have no documentation.

Finally, the **–B –M** and **–S** flags may be used to change or otherwise limit the places where *whereis* searches. The **–f** file flag is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
whereis –u –M /usr/man/man1 –S /usr/src/cmd –f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

BUGS

Since the program uses *chdir*(2) to run faster, pathnames given with the **–M –S** and **–B** must be full; i.e. they must begin with a "/".

NAME

who – which users are on the system

SYNOPSIS

who [who-file] [**am I**]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Who, without an argument, lists the login name, terminal name, and login time for each current HP-UX user.

Without an argument, *who* examines the /**etc**/**utmp** file to obtain its information. If a file is given, that file is examined. Typically the given file will be /**usr**/**adm**/**wtmp**, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with /**dev**/ suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with **x** in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in **who am I** (and also **who are you**), *who* tells who you are logged in as.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5).

NAME
whodo – which users are doing what

SYNOPSIS
/etc/whodo

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Whodo produces merged, reformatted, and dated output from the *who*(1) and *ps*(1) commands.

SEE ALSO
ps(1), who(1).

NAME
write – interactively write (talk) to another user

SYNOPSIS
write user [tty]

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *your-logname your-tty* . . .

The recipient of the message should *write* back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point, *write* writes **EOF** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *tty* argument may be used to indicate the appropriate terminal.

Permission to write may be denied or granted by use of the *mesg*(1) command. At the outset, writing is allowed. Certain commands, in particular *nroff*(1) and *pr*(1), disallow messages in order to prevent messy output.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for him or her to write back before starting to send. Each party should end each message with a distinctive signal (**(o)** for "over" is conventional), indicating that the other may reply; **(oo)** for "over and out" is suggested when conversation is to be terminated.

FILES
/etc/utmp	to find user
/bin/sh	to execute **!**

SEE ALSO
mail(1), mesg(1), who(1).

NAME
yacc – yet another compiler-compiler

SYNOPSIS
yacc [**–vd**] grammar

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **–v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **–d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

FILES
y.output	
y.tab.c	
y.tab.h	definitions for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO
lex(1).
YACC – Yet Another Compiler Compiler in *HP-UX: Selected Articles.*

DIAGNOSTICS
The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS
Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

NAME

intro – introduction to system calls

HP-UX COMPATIBILITY

Level:	This entry describes where in the HP-UX compatibility model this capability appears.
Origin:	System III, HP, or UCB

DESCRIPTION

This section describes all of the system calls. All of these calls return a function result. This result indicates the status of the call. Typically, a zero or positive result indicates that the call completed successfully, and −1 indicates an error. The individual descriptions specify the details. An error number is also made available in the external variable *errno* (see *errno*(2)).

HARDWARE DEPENDENCIES

Series 500/600/700:

A second error indicator, *errinfo*, is implemented in addition to *errno*. See *errinfo*(2).

SEE ALSO

intro(3).

NAME

access – determine accessibility of a file

SYNOPSIS

int access (path, amode)
char *path;
int amode;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to a path name naming a file. *Access* checks the named file for accessibility according to the bit pattern contained in *amode*, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID. The bit pattern contained in *amode* is constructed as follows:

04	read
02	write
01	execute (search)
00	check existence of file

Access to the file is denied if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

Read, write, or execute (search) permission is requested for a null path name. [ENOENT]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

Write access is requested for a file on a read-only file system. [EROFS]

Write access is requested for a pure procedure (shared text) file that is being executed. [ETXTBSY]

Permission bits of the file mode do not permit the requested access. [EACCES]

Path points outside the process's allocated address space. [EFAULT]

The owner of a file has permission checked with respect to the "owner" read, write, and execute mode bits, members of the file's group other than the owner have permissions checked with respect to the "group" mode bits, and all others have permissions checked with respect to the "other" mode bits. *Access* will always report accessibility when executed by the super-user.

RETURN VALUE

If the requested access is permitted, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

chmod(2), stat(2).

NAME

alarm – set process's alarm clock

SYNOPSIS

unsigned alarm (sec)
unsigned sec;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Alarm instructs the calling process's alarm clock to send the signal **SIGALRM** to the calling process after the number of real time seconds specified by *sec* have elapsed; see *signal*(2).

Response to the alarm may be delayed due to normal HP-UX scheduling.

Alarm requests are not stacked; successive calls reset the calling process's alarm clock.

If *sec* is 0, any previously made alarm request is canceled.

Alarms are not inherited by a child process across a *fork*.

RETURN VALUE

Alarm returns the amount of time previously remaining in the calling process's alarm clock.

SEE ALSO

sleep(1), pause(2), signal(2), sleep(3).

NAME

brk, sbrk – change data segment space allocation

SYNOPSIS

int brk (endds)
char *endds;

char *sbrk (incr)
int incr;

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

Brk and *sbrk* are used to change dynamically the amount of space allocated for the calling process's data segment; see *exec*(2). The change is made by resetting the process's break value. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.

Brk sets the break value to *endds* and changes the allocated space accordingly.

Sbrk adds *incr* bytes to the break value and changes the allocated space accordingly. *Incr* can be negative, in which case the amount of allocated space is decreased.

Brk and *sbrk* will fail without making any change in the allocated space if such a change would result in more space being allocated than is allowed by a system-imposed maximum (see *ulimit*(2)). [ENOMEM]

HARDWARE DEPENDENCIES

Series 500/600/700:

Brk sets the break value to *endds.*

Brk and *sbrk* will also fail without making any change in the allocated space if such a change would move the program break below the beginning of your process' data area. Note that it is not possible to release the direct data area with this system call.

If your process' data area is paged, the the size of that data area changes in increments of the page size, which is configurable. Consequently, increasing a paged process data area by one byte may cause it to increase by one page, and decreasing it by one byte may do nothing. If your process' data area is not paged, then the size of the process data area changes similarly in increments of 32 bytes.

The pointer returned by *sbrk* is not necessarily word-aligned. Loading or storing words through this pointer could cause word alignment problems.

RETURN VALUE

Upon successful completion, *brk* returns a value of 0 and *sbrk* returns the old break value. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

exec(2), end(3), malloc(3).

NAME

chdir – change working directory

SYNOPSIS

int chdir (path)
char *path;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to the path name of a directory. *Chdir* causes the named directory to become the current working directory, the starting point for path searches for path names not beginning with /.

Chdir will fail and the current working directory will be unchanged if one or more of the following are true:

A component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

Search permission is denied for any component of the path name. [EACCES]

Path points outside the process's allocated address space. [EFAULT]

Path is null. [ENOENT]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

cd(1), chroot(2).

NAME

chmod – change access mode of file

SYNOPSIS

int chmod (path, mode)
char *path;
int mode;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to a path name naming a file. *Chmod* sets the access permission portion of the named file's mode according to the bit pattern contained in *mode*.

Access permission bits are interpreted as follows:

04000	Set user ID on execution.
02000	Set group ID on execution.
01000	Save text image after execution
00400	Read by owner
00200	Write by owner
00100	Execute (or search if a directory) by owner
00070	Read, write, execute (search) by group
00007	Read, write, execute (search) by others

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user, mode bit 01000 (save text image on execution) is cleared.

If the effective user ID of the process is not super-user or the effective group ID of the process does not match the group ID of the file, mode bit 02000 (set group ID on execution) is cleared.

If an executable file is prepared for sharing then mode bit 01000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus, when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time.

Chmod will fail and the file mode will be unchanged if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

Path is null. [ENOENT]

HARDWARE DEPENDENCIES

Series 500/600/700:

Chmod changes the mode of files created only in the HP-UX environment (that is, not those created by the HP 9000 BASIC Language System).

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

chmod(1), chown(2), mknod(2).

NAME

chown – change owner and group of a file

SYNOPSIS

int chown (path, owner, group)
char *path;
int owner, group;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to a path name naming a file. The owner ID and group ID of the named file are set to the numeric values contained in *owner* and *group* respectively. Note that *owner* and *group* should be less than or equal to 65535, since only the least significant 16 bits are used.

Only processes with effective user ID equal to the file owner or super-user may change the ownership of a file.

If *chown* 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.

Chown will fail and the owner and group of the named file will remain unchanged if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]

The named file resides on a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

HARDWARE DEPENDENCIES

Series 500/600/700:

Chown changes the owner and group of files created only in the HP-UX environment (that is, not those created by the HP 9000 BASIC Langauge System).

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

chown(1), chmod(2).

NAME

chroot – change root directory

SYNOPSIS

int chroot (path)
char *path;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to a path name naming a directory. *Chroot* causes the named directory to become the root directory, the starting point for path searches for path names beginning with /. (*Chroot* does not affect the current working directory, ., thus it is still possible to access files outside the tree which is the new root unless or until a *chdir* is done to move the current working directory under the new root.)

The effective user ID of the process must be super-user to change the root directory.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, .. cannot be used to access files outside the subtree rooted at the root directory.

Chroot will fail and the root directory will remain unchanged if one or more of the following are true:

Any component of the path name is not a directory. [ENOTDIR]

The named directory does not exist. [ENOENT]

The effective user ID is not super-user. [EPERM]

Path points outside the process's allocated address space. [EFAULT]

Path is null. [ENOENT]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

chroot(1), chdir(2).

NAME

close – close a file descriptor

SYNOPSIS

int close (fildes)
int fildes;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fildes is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *Close* closes the file descriptor indicated by *fildes*.

Close will fail if *fildes* is not a valid open file descriptor. [EBADF]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

creat(2), dup(2), exec(2), fcntl(2), open(2), pipe(2).

NAME

creat – create new file, rewrite existing file

SYNOPSIS

int creat (path, mode)
char *path;
int mode;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Creat creates a new ordinary file or prepares to rewrite an existing file named by the path name pointed to by *path*.

If the file exists, the length is truncated to 0 and the mode and owner are unchanged. Otherwise, the file's owner ID is set to the process's effective user ID, the file's group ID is set to the process's effective group ID, and the low-order 12 bits of the file mode are set to the value of ***mode*** modified as follows:

All bits set in the process's file mode creation mask are cleared. See ***umask***(2).

The "save text image after execution bit" of the mode is cleared. See *chmod*(2).

Upon successful completion, a non-negative integer, namely the file descriptor, is returned and the file is open for writing, even if the mode does not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across *exec* system calls. See *fcntl*(2). No process may have more than 20 files open simultaneously. A new file may be created with a mode that forbids writing.

Creat will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

A component of the path prefix does not exist. [ENOENT]

Search permission is denied on a component of the path prefix. [EACCES]

The path name is null. [ENOENT]

The file does not exist and the directory in which the file is to be created does not permit writing. [EACCES]

The named file resides or would reside on a read-only file system. [EROFS]

The file is a pure procedure (shared text) file that is being executed. [ETXTBSY]

The file exists and write permission is denied. [EACCES]

The named file is an existing directory. [EISDIR]

Twenty (20) file descriptors are currently open. [EMFILE]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

close(2), dup(2), lseek(2), open(2), read(2), umask(2), write(2).

NAME

dup – duplicate an open file descriptor

SYNOPSIS

int dup (fildes)
int fildes;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fildes is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *Dup* returns a new file descriptor having the following in common with the original:

Same open file (or pipe).

Same file pointer (i.e., both file descriptors share one file pointer.)

Same access mode (read, write or read/write).

Same file status flags (see *fcntl*(2), F_DUPFD).

The new file descriptor is set to remain open across *exec* system calls. See *fcntl*(2).

The file descriptor returned is the lowest one available.

Dup will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Twenty (20) file descriptors are currently open. [EMFILE]

RETURN VALUE

Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

close(2), creat(2), exec(2), fcntl(2), open(2), pipe(2).

NAME

ems – Extended Memory System

SYNOPSIS

#include <sys/ems.h>

HP-UX COMPATIBILITY

Level:	Extended Memory - HP-UX/EXTENDED
Origin:	HP

DESCRIPTION

Extended Memory System consists of intrinsics which allocate and deallocate address space, map files into address spaces, support shared memory, and change the protection of address spaces. There are separate manual pages for the intrinsics. This page describes features in common to all the intrinsics in EMS.

Definitions

memory space
This is the actual physical memory of a machine.

address space
This refers to the logical memory of a process. Memory space is shared by having processes' address space refer to the same memory space.

segment
A contiguous piece of address space.

Properties of a Segment

During the allocation of a segment, the following types of segments can be requested:

MEM_SHARED
The address space is to be sharable with other processes. The data is shared across *fork*(2) (i.e. not copied on a fork).

MEM_PRIVATE
The address space is process local, and is copied on a *fork*(2). All memory segments will be either MEM_SHARED or MEM_PRIVATE; the default is MEM_PRIVATE.

MEM_CODE
The address space may, at some time in its lifetime, be made executable.

MEM_DATA
The address space may, at some time in its lifetime, be read and/or written. A segment may be MEM_CODE, MEM_DATA, or both. The default type is derived from the initial access permissions:

```
MEM_R | MEM_W                → MEM_DATA
MEM_X                        → MEM_CODE
(MEM_R | MEM_W) && MEM_X     → MEM_CODE | MEM_DATA.
```

MEM_PAGED
Requests that a segment be created as a paged object. (This is ignored if not significant for a particular implementation).

File Mapping

EMS provides the facility for mapping a file into process address space. This is done via *memallc*(2). Files can be either private or shared.

For private file mapped segments, the address space will contain an image of the file as it existed at the time of the *memallc*(2) call. Subsequent alterations of the file will have no effect on the contents of the address space, and vice-versa.

For shared file mapped segments, the address space is identically the file (at least the mapped portion thereof). Changes to the address space represent changes to the file, and vice-versa. For example, a write or read to or from the address space is, in all ways, equivalent to a file system write or read. Similarly, re-creating (using *creat*(2)) the file will result in the address space containing all zeros.

The access permissions (e.g. read, write) applied to a shared mapped file are established by the first *memallc*(2) referencing that file. Subsequent mappings of the same file by other processes must request identical access permissions.

File mapping, as described above, is only guaranteed to apply to regular local files and block structured device files. File mapping is not applicable to remote files at this time. Attempting to map an unsupported file type will result in error EINVAL.

Note that file mapping, either MEM_PRIVATE or MEM_SHARED, *always* requires read permission on *fileid*. Access modes cannot exceed those on *fileid* for shared, mapped files.

Shared Memory
It is possible to share a memory space between processes. Access to shared memory can occur in two ways. The first way is to associate a file name as the name of the shared memory space. Each related or unrelated process performs a *memallc*(2) to gain access to the shared memory through mapping the file.

Another method of sharing, without the file, is for related processes: a process can allocate a non-file-mapped shared segment; upon a *fork*(2), the child process will have access to the same memory space as the parent.

SEE ALSO

memadvise(2), memallc(2), memchmd(2), memlck(2), memvary(2), vsadv(2), vson(2).

NAME

errinfo – error indicator

SYNOPSIS

extern int errinfo;

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Errinfo* is implemented on the Series 500/600/700 only.

DESCRIPTION

When an error occurs in a system call, the external variable *errno* is set to the standard HP-UX error number, and more detailed information is stored in the external variable *errinfo*. *Errinfo* obtains its value from the escape code returned by the underlying HP-UX kernel.

Errinfo is not cleared on successful system calls, so it should only be checked after an error has been indicated.

Software that is intended to be portable across HP-UX implementations should not reference *errinfo*.

The *errinfo* values and their meanings are as follows:

VALUE	MEANING
*4	NVM address out of range;
5	buffer request is not within valid range;
6	buffer address space overflow;
*7	address specified does not reference a valid buffer;
*10	specified process priority level out of range;
*11	a non-existent code segment is specified;
*12	attempt to delete non-existent partition;
*13	system parameter not addressable;
*14	system parameter cannot be referenced with an EDS pointer;
*20	invalid message link;
*21	invalid message link;
*22	message limit exceeded;
*23	link limit exceeded;
*24	link being deleted contains processes waiting for messages;
*30	timer cancelled;
*31	timer stopped;
*32	cancel already done for specified timer ID;
*33	stop already done for specified timer ID;
*34	timer ID not stopped before cleared;
*35	timer ID not cancelled before cleared;
*36	attempt to set time and date to a value outside accessible range (midnight January 1, 1900 to midnight December 31, 25599);
*37	stack extension error;
40	memory overflow (private partition);
41	memory overflow;
42	no free partition available for allocation;
43	segment table overflow;
44	memory controller block overflow;
45	partition overflow;
46	pointer passed as an argument does not point to a valid segment;
47	segment size is out of range;
*48	free space chains are inconsistent, segment map corruption;
*49	free space chains are inconsistent, block map corruption;

50	pointer passed as an argument does not point to a valid segment;
*51	block address within a segment is invalid;
56	device or interface card timed out;
57	system call aborted by *signal*(2);
*59	improper resource management in operating system;
*60	improper resource management in operating system;
*63	routine called for wrong I/O device or at wrong time;
*64	routine called for wrong I/O device or at wrong time;
*65	used in BASIC only;
66	hardware or firmware error in interface card;
*67	I/O transaction aborted by device or interface card;
68	an HP-IO interface card failed its self test;
*69	used during power-up, produces "System halted - incompatible IOP's" message;
*70	no such object;
*73	out of timer ID's;
*74	timer ID out of range;
*75	start_partition parameters not consistent;
*76	parameter to start_partition not addressable;
*77	attempt to change to non-existent partition;
*78	must be a system process to change to partition;
80	device not ready for request, may be busy with some other operation, or power may be off;
81	media is write-protected and cannot be altered;
82	media has been mis-inserted;
83	format switch disables driver from doing a media format operation;
84	media error was detected, usually a CRC, parity, or checksum error; data may not be valid;
85	cannot find record on media; usually indicates trouble in reading the header/servo information on the media;
86	the read check of data written to a record has failed;
88	media may have been changed since last access; buffered data may have to be thrown away;
*89	used to implement internally generated re-tries;
*90	software failure was detected; perhaps data structures were corrupted, or an unexpected event occurred;
91	unknown error; indicates some type of device or interconnect malfunction;
*94	media_active (true) request must be made before first access;
95	a parameter for a particular request is not supported by this driver; usually indicates that the type of card does not support a special function;
97	termination mode is not supported by this device driver;
98	EOI must have a data byte associated with it before it can be written;
*99	driver must be opened for request;
100	record number out of allowed range; usually indicates corrupt directory structure;
*101	the transfer length was negative, zero, or odd for a halfword read or write request; can also indicate a transfer past the end of the media volume;
102	halfword or byte mode transfers are not supported by this driver;
*103	cannot close a locked driver; this is a fault of the calling code;
107	an attempt was made to attach two different drivers to the same device; these drivers are incompatible and cannot co-exist; the new driver is not attached, but the old driver remains unchanged;
108	the size of the string is not correct for this string register access;
109	interleave factor not supported by disc; it is either zero, negative, or too big;
110	invalid address was detected by the driver, or the interface card occupies the same subaddress as the device;
*111	capacity of disc exceeds 32-bit record address range assumed by driver;

112	reference to an unsupported pseudo-register was made; if the request accessed multiple registers, the previous (if valid) register accesses were made;
113	HP-IB TCT byte must be at the end of the ATN sequence because you have passed control;
114	a request is not supported by this driver;
115	no driver with that name was found;
116	no driver is available for that card, or the device address value is out of range;
117	write verify is not supported for this mass storage device;
118	length of –1 specifying that a transfer should be used is invalid;
119	an invalid value was assigned to a pseudo-register;
120	data transfer was terminated due to the reception of a secondary address;
121	for buffered devices, a data transfer cannot be satisfied due to un-transferred data from the other direction; for example, a write may not be possible if there is still unread data present on the device;
122	device cannot satisfy this request because of a previous request or the current state of the device;
123	the beginning of the tape was encountered before the operation could be completed;
124	the interface cannot be the HP-IB active controller when doing this operation;
125	synchronous data rate could not be met to complete the operation; system may be too heavily loaded, or the specified bandwidth parameters for this or another device may be wrong;
126	a hardware fault was detected; controller/status card should be examined for further information;
127	the device/interface was not found at the specified address; power may be off, or the address could be wrong;
128	the end of tape was encountered before the operation was complete;
129	the device failed its self test or a diagnostic; no further access to this device should be attempted;
130	the HP-IB interface is too slow for this synchronous device;
131	tape end of file was encountered before request could be completed;
132	the device was busy and could not handle the request;
133	the media is absent from the device;
134	the media is not formatted, and must be formatted before use;
135	too many media errors prevent formatting to complete; formatting operation may be only partially done;
136	the media has no more spares left but had to spare some data; the sparing was not done;
137	the HP-IB interface must be the active controller to execute this operation;
138	the HP-IB interface must be the system controller to execute this operation;
139	no data seen on media after a device specific length of media; this is a sequential tape error;
140	more data was found in the record than was requested for the read operation; the remaining data was lost, and cannot be read by the next read request;
141	the media physical format is incorrect for this disc;
142	media failure has occurred, or the media has deteriorated such that replacement is suggested; writing is no longer allowed; media may only last long enough for a back-up;
143	the HP-IB interface is not addressed to read or write as requested, and because it is active controller, it cannot become addressed;
144	the read or write request data transfer was aborted by an HP-IB IFC or an HP-IB device clear operation;
145	not all the data (or commands) were accepted by the device;
146	not all the data was sourced by the device;
147	controller or unit fault was reported by the device;
148	some failure occurred in receiving the device status result; usually means that not all

	the status was returned, or the controller reported a failure when the driver attempted to receive the status;
149	the operation cannot be completed because a user programmed holdoff has occurred;
*150	system problem or failure;
*151	successful completion of task; should not be visible;
157	the volume label specified in the volume specifier does not match the volume label on the volume;
158	links may not be removed if the file has been opened with the "no purge link" option;
160	cannot open a directory with write access;
161	two or more volumes have the same volume label and the file system is unable to distinguish between them for this request;
162	an attempt was made to access an open file in a way forbidden by the file system;
163	the disc format does not support the requested operation;
164	the file cannot be opened for writing because it is currently being *exec*ed, or the file may not be opened with execute access because it is currently opened for writing;
165	the file/device could not be opened because the system open file table is full; this is caused by a memory error;
166	a file may not be opened in both "shared" and "exclusive" modes; your access mode conflicts with the current mode;
167	a signal was received while waiting to read or write to a pipe;
168	the request cannot be performed because the designated file is open or in use at the current time;
169	an attempt was made to purge a link to the file without obtaining the necessary access rights;
170	not enough disc space could be allocated to satisfy the request;
171	a file with the same name already exists in the directory;
172	the file ID passed to the system was bad;
173	an attempt was made to read beyond the physical end of the file;
174	tried to write to a pipe for which there are no readers;
*175	the request made is not supported by the file system;
176	same as error 162, except that the file may not be open;
177	a "position" (*lseek*) request was made on a pipe;
178	the device driver specified in the volume specifier does not match the current device driver being used for the volume;
179	the disc format specified in the volume specifier does not match the disc format on the volume;
181	some file in the file path could not be found;
182	the device specified is not a random access blocked device;
183	the disc format on the disc does not support volume labels;
184	the disc format on the disc does not support file passwords;
185	the disc does not contain a recognizable disc format; the disc format name given for an initialize request is not known to the system;
189	a volume may not be initialized while there are open files on it;
193	a non-directory was specified where a directory was required;
198	the request cannot be satisfied because another file cannot be added to the directory; no i-nodes were available;
201	the request cannot be satisfied because the directory is not empty;
204	the file system was unable to extend a "contiguous" file without creating another extent;
*210	invalid file code;
216	the select code in the device address in the volume specifier is not within the acceptable range for this hardware configuration;
*217	an attempt was made to remove or change a password which does not exist;

*218	an attempt was made to put two identical passwords on a file with different capability sets;
*219	a simple deadlock was encountered when locking a file;
221	the file name is too long (LIF discs support 10 characters, HP 9845 format discs support 6 characters, and SDF discs support 16 characters);
222	invalid character in LIF or HP 9845 format disc file name;
*223	invalid character in LIF or HP 9845 format disc password;
*224	volume label is too long on a LIF or HP 9845 format disc;
*225	password too long on a LIF or HP 9845 format disc;
*226	invalid character in volume label on a LIF or HP 9845 format disc;
*227	invalid date on LIF or HP 9845 format disc;
*228	invalid record size on LIF or HP 9845 format disc;
229	invalid record mode on LIF or HP 9845 format disc;
230	a file name was expected and none was specified, or an attempt was made to purge the "." or ".." links from a directory;
231	a subdirectory was specified when the disc format does not support subdirectories;
232	links not supported on LIF or HP 9845 format discs;
233	non-UNIX systems are not allowed to establish duplicate links to a directory;
234	the device (file) specified for the ***mount/umount*** request is not a block special device;
235	the device (file) specified for the ***umount*** request is not currently mounted;
236	a volume could not be unmounted because it is currently being used (there are open files or working directories established on the mounted volume); a volume could not be mounted because it is already mounted; the directory being mounted on is open or is the root directory;
237	an attempt was made to establish a link from one volume to another;
241	the byte address on a file access was outside the acceptable range for the file; the byte address must be non-negative;
242	the file system saw a directory, i-node, or bit map record which contains inconsistent data;
244	an attempt was made to read beyond the logical end of the file;
249	an attempt was made to unlock an unlocked file;
*252	time value out of range;
*253	hours, minutes, or seconds value out of range;
*254	day, month, or year value out of range;
*255	invalid date;
256	specified segment does not exist;
257	page table has not been initialized;
258	page has not been initialized;
259	lock count has overflowed;
260	lock count has underflowed;
261	entire working set cannot be locked;
262	lock length is invalid;
263	segment is not locked;
264	locked segment cannot be extended;
265	page is not locked;
266	segment is not paged;
267	segment is not shared;
268	requested segment lengths are inconsistent;
269	minimum working set request cannot be satisfied;
270	frame pool cannot be expanded;
271	virtual memory device table overflow;
272	virtual memory device index is invalid;
273	default virtual memory device cannot be removed;
274	virtual memory device index is inactive;
275	virtual memory device index is in use;

276	a locked page was encountered;
*301	escape through user code for *exec*;
302	target process not found in *kill* call;
303	target process has the wrong real user ID in *kill* call;
304	no processes found in a broadcast signal attempt;
305	signal number out of range;
306	not super-user; requires super-user permission;
307	a bad argument was supplied to a system call;
308	an attempt was made to wait with no children;
309	an intrinsic was aborted by a signal;
310	process stack overflow;
311	unrecognized *ulimit* command;
312	your DB relative argument had an offset greater than 512 Kbytes;
313	fix-up offset exceeds segment size (see *a.out*(5));
314	stack pointer passed to *brk*;
315	invalid segment number in user pointer;
316	an attempt was made to *kill*(0,sig) with no current process group;
317	file number out of range;
318	specified file ID not open;
319	*ioctl* call not implemented;
320	inappropriate *ioctl* command for device;
321	ID not in the range 0 to 65535;
323	floating point divide-by-zero;
324	floating point overflow;
325	floating point underflow;
327	wrong number of system call parameters;
328	inconsistent executable file;
329	front panel timeout (series 500, models 30 and 40 only);
330	graphics to internal CRT timed out;
331	graphics hardware does not respond;
*332	unexpected error when performing an open;
*333	unexpected error when performing a close;
334	illegal mode of driver was requested;
335	a buffer was passed to an intrinsic that is too large;
336	DMA terminated abnormally;
337	received one more x coordinate than y coordinate;
345	attempt to execute a file which is too small;
346	attempt to execute a file with a bad magic number;
347	unimplemented configure function;
348	maximum stack exceeded;
349	fatal stack overflow;
350	the requested heap size is too big;
*440	internal error;
441	protection modes do not match with existing segment;
442	device is not a 'CS80' device;
443	attempt to add a device not specified with a device file;
444	attempt to pass an EMS intrinsic a parameter which is out of range;
445	attempt to *memchmd* segment codes which are shared by more than one process;
446	attempt to filemap a file which has already been filemapped by process;
447	insufficient memory available to complete *memallc* request;
448	the specified memory address is invalid;
449	attempt to use EMS intrinsic on memory not allocated by *memallc*.

All *errinfo* values marked with an asterisk (*) indicate a serious system problem which should be checked by qualified HP personnel.

SEE ALSO

err(1), errno(2), perror(3C).

WARNING

Errinfo is intended for diagnostic purposes only. Values and meanings may change in future releases of HP-UX.

NAME

errno – error indicator for system calls

SYNOPSIS

#include <errno.h>
extern int errno;

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Errno is an external variable whose value is set whenever an error occurs in a system call. This value can be used to obtain a more detailed description of the error. *Errno* is not cleared on successful system calls, so its value should be checked only when an error has been indicated.

The following is a complete list of the error numbers and their names as defined in **errno.h**.

1 EPERM Not owner
: Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory
: This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process
: No process can be found corresponding to that specified by *pid* in *kill* or *ptrace*, or is not accessible.

4 EINTR Interrupted system call
: An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error
: Some physical I/O error. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address
: I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
: An argument list longer than 5,120 bytes is presented to a member of the *exec* family.

8 ENOEXEC Exec format error
: A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see *a.out*(5)), or the file is too small to have a valid executable file header.

9 EBADF Bad file number
: Either a file descriptor refers to no open file, a read (respectively write) request is made to a file which is open only for writing (respectively reading), or the file descriptor is not in the range 0 to 19.

10 ECHILD No child processes
: A *wait* was executed by a process that had no existing or unwaited-for child processes.

11 EAGAIN No more processes
: A *fork* failed because the system's process table is full or the user is not allowed to create any more

processes.

12 ENOMEM Not enough space
: During an *exec*, *brk*, *fork*, or *sbrk*, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a *fork*.

13 EACCES Permission denied
: An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address
: The system encountered a hardware fault in attempting to use an argument of a system call; can also result from passing the wrong number of parameters to a system call.

15 ENOTBLK Block device required
: A non-block file was mentioned where a block device was required, e.g., in *mount*.

16 EBUSY Mount device busy
: An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled, or if a non-shareable device file is in use.

17 EEXIST File exists
: An existing file was mentioned in an inappropriate context, e.g., *link*.

18 EXDEV Cross-device link
: A link to a file on another device was attempted.

19 ENODEV No such device
: An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.

20 ENOTDIR Not a directory
: A non-directory was specified where a directory is required, for example in a path prefix or as an argument to *chdir*(2).

21 EISDIR Is a directory
: An attempt to open a directory for writing.

22 EINVAL Invalid argument
: Some invalid argument (e.g., dismounting a non-mounted device; mentioning an undefined signal in *signal*, or *kill*; reading or writing a file for which *lseek* has generated a negative pointer). Also set by the math functions described in the (3M) entries of this manual.

23 ENFILE File table overflow
: The system's table of open files is full, and temporarily no more *opens* can be accepted.

24 EMFILE Too many open files
: No process may have more than 20 file descriptors open at a time.

25 ENOTTY Not a typewriter
: The command is inappropriate to selected device type.

26 ETXTBSY Text file busy
: An attempt to execute an executable file which is currently open for writing. Also, an attempt to open for writing a writable file which is currently open for execution.

27 EFBIG File too large
: The size of a file exceeded the maximum file size (for the file system) or ULIMIT; see *ulimit*(2).

28 ENOSPC No space left on device
: During a *write* to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek
: An *lseek* was issued to a pipe.

30 EROFS Read-only file system
: An attempt to modify a file or directory was made on a device mounted read-only.

31 EMLINK Too many links
: An attempt to make more than the maximum number of links (1000) to a file.

32 EPIPE Broken pipe
: A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument
: The argument of a function in the math package (3M) is out of the domain of the function.

34 ERANGE Result too large
: The value of a function in the math package (3M) is not representable within machine precision.

35 ENET Local area network error
: An error occurred in the software or hardware associated with your local area network.

99 Unexpected error
: An unexpected error was returned from the system, indicating some type of system problem. This error should never occur; if it does, it indicates a system bug.

HARDWARE DEPENDENCIES

Series 500/600/700:

In the definition of error 12 (ENOMEM), the maximum space size is not a system parameter. Also, the terms "text, data, and stack segments", "segmentation registers", and "swap space" are invalid.

In the definition of error 31 (EMLINK), the maximum number of links is 32767.

A second error indicator, *errinfo*, is implemented in addition to *errno*. See *errinfo*(2).

SEE ALSO

err(1), errinfo(2).

NAME

execl, execv, execle, execve, execlp, execvp – execute a file

SYNOPSIS

int execl (path, arg0, arg1, ..., argn, 0)
char *path, *arg0, *arg1, ..., *argn;

int execv (path, argv)
char *path, *argv[ ];

int execle (path, arg0, arg1, ..., argn, 0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp[ ];

int execve (path, argv, envp);
char *path, *argv[ ], *envp[ ];

int execlp (file, arg0, arg1, ..., argn, 0)
char *file, *arg0, *arg1, ..., *argn;

int execvp (file, argv)
char *file, *argv[ ];

HP-UX COMPATABILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Exec, in all its forms, loads a program from an ordinary, executable file onto the current process. This file consists of a header (see *a.out*(5)), a text segment, and a data segment. The data segment contains an initialized portion and an uninitialized portion (bss). There can be no return from a successful *exec* because the calling program is overlaid by the new program.

Path points to the absolute path name that identifies the executable file containing the new program.

File points to a file name identifying the executable file containing the new program. The path prefix for this file is obtained by a search of the directories passed as the *environment* line "PATH =" (see *environ*(7)). The environment is supplied by the shell (see *sh*(1)). If *file* does not have an executable magic number (*magic*(5)), then it is passed to */bin/sh* under the assumption that *file* is a shell script.

Arg0, *arg1*, ..., *argn* are pointers to null-terminated character strings. These strings constitute the argument list available to the new command. By convention, at least *arg0* must be present and point to a string that is the same as *path* (or its last component).

Argv is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new command. By convention, *argv* must have at least one member, and it must point to a string that is the same as *path* (or its last component). *Argv* is terminated by a null pointer.

Envp is an array of character pointers to null-terminated strings. These strings constitute the environment in which the new command will run. *Envp* is terminated by a null pointer. Note that, in *exec* calls not requiring *envp*, the environment for the new command defaults to the current environment.

File descriptors open in the calling process remain open, except for those whose close-on-exec flag is set; see *fcntl*(2). For those file descriptors that remain open, the file pointer is unchanged.

Signals set to terminate the calling program will be set to terminate the new program. Signals set to be ignored by the calling program will be set to be ignored by the new program. Signals set to be caught by the calling program will be set to terminate the new program; see *signal*(2).

If the set-user-ID mode bit of the executable file pointed to by *path* or *file* is set (see *chmod*(2)), *exec* sets the effective user ID of the new program to the owner ID of the executable file. Similarly, if the set-group-ID mode bit of the executable file is set, the effective group ID of the new program is set to the group ID of the executable file. The real user ID and real group ID of the new program remain the same as those of the calling program.

Profiling is disabled for the new program; see *profil*(2).

The new program also inherits the following attributes from the calling program:

nice value (see *nice*(2))
process ID
parent process ID
process group ID
tty group ID (see *exit*(2) and *signal*(2))
trace flag (see *ptrace*(2) request 0)
time left until an alarm clock signal (see *alarm*(2))
current working directory
root directory
file mode creation mask (see *umask*(2))
file size limit (see *ulimit*(2))
utime, *stime*, *cutime*, and *cstime* (see *times*(2))

Exec will fail and return to the calling program if one or more of the following are true:

One or more components of the executable file's path name do not exist. [ENOENT]

A component of the executable file's path prefix is not a directory. [ENOTDIR]

Search permission is denied for a directory listed in the executable file's path prefix. [EACCES]

The executable file is not an ordinary file. [EACCES]

The file pointed to by *path* or *file* is not executable. [EACCES]

The executable file has the appropriate access permission, but has an invalid magic number in its header. [ENOEXEC]

The executable file is currently open for writing. Note: normal executable files are only open for a short time when they start execution. Other executable file types may be kept open for a long time, or indefinitely under some circumstances. [ETXTBSY]

The new program requires more memory than is available, or than is allowed by the system-imposed maximum MAXMEM. [ENOMEM]

The number of bytes in the new program's argument list is greater than the system-imposed limit of 5120 bytes. [E2BIG]

The executable file is not as long as indicated by the size values in its header, or is otherwise inconsistent. [EFAULT]

Path, *argv*, or *envp* point to an illegal address. [EFAULT]

Path is null. [ENOENT]

HARDWARE DEPENDENCIES

Series 500/600/700:

References to memory, such as "text segment", "data segment", "initialized portion", "uninitialized portion", and "bss", are invalid. See *a.out*(5) for the Series 500/600/700.

Ptrace(2) (and trace flags) are not currently supported.

RETURN VALUE

If *exec* returns to the calling program, an error has occurred; the return value will be –1 and *errno* will be set to indicate the error.

SEE ALSO

exit(2), fork(2), environ(7).

BUGS

The super-user cannot exec a file unless at least one of the three execute bits is set in the file's mode.

NAME

exit, _exit – terminate process

SYNOPSIS

exit (status)
int status;

_exit (status)
int status;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Exit terminates the calling process with the following consequences:

All of the file descriptors open in the calling process are closed, and standard I/O buffers are flushed (see *stdio*(3S)).

If the parent process of the calling process is executing a *wait*, it is notified of the calling process's termination and the low order eight bits (i.e., bits 0377) of *status* are made available to it; see *wait*(2).

If the parent process of the calling process is not executing a *wait*, the calling process is transformed into a zombie process. A *zombie process* is a process that only occupies a slot in the process table, it has no other space allocated either in user or kernel space. Time accounting information is recorded for use by *times* (see <**sys/proc.h**>).

The parent process ID of all of the calling process's existing child processes and zombie processes is set to 1. This means the initialization process (proc1, see *intro*(2)) inherits each of these processes.

An accounting record is written on the accounting file if the system's accounting routine is enabled; see *acct*(2).

If the process ID, tty group ID, and process group ID of the calling process are equal, the **SIGHUP** signal is sent to each process that has a process group ID equal to that of the calling process.

_exit is equivalent to *exit* except that standard I/O buffers are not flushed.

HARDWARE DEPENDENCIES

Series 500/600/700:
Accounting is not currently supported.

The include file **sys/proc.h** is not provided.

SEE ALSO

signal(2), wait(2).

WARNING

See *WARNING* in *signal*(2).

NAME

fcntl – file control

SYNOPSIS

#include <fcntl.h>

int fcntl (fildes, cmd, arg)
int fildes, cmd, arg;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fcntl provides for control over open files. *Fildes* is an open file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

The *cmd*s available are:

F_DUPFD Return a new file descriptor as follows:

Lowest numbered available file descriptor greater than or equal to *arg*.

Same open file (or pipe) as the original file.

Same file pointer as the original file (i.e., both file descriptors share one file pointer).

Same access mode (read, write or read/write).

Same file status flags (i.e., both file descriptors share the same file status flags).

The close-on-exec flag associated with the new file descriptor is set to remain open across *exec*(2) system calls.

F_GETFD Get the close-on-exec flag associated with the file descriptor *fildes*. If the low-order bit is **0** the file will remain open across *exec*, otherwise the file will be closed upon execution of *exec*.

F_SETFD Set the close-on-exec flag associated with *fildes* to the low-order bit of *arg* (**0** or **1** as above).

F_GETFL Get *file* status flags.

F_SETFL Set *file* status flags to *arg*. Only certain flags can be set; see *fcntl*(7).

Fcntl will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Cmd is F_DUPFD and 20 file descriptors are currently open. [EMFILE]

Cmd is F_DUPFD and *arg* is negative or greater than 20. [EINVAL]

RETURN VALUE

Upon successful completion, the value returned depends on *cmd* as follows:

F_DUPFD	A new file descriptor.
F_GETFD	Value of flag (only the low-order bit is defined).
F_SETFD	Value other than –1.
F_GETFL	Value of file flags.
F_SETFL	Value other than –1.

Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

close(2), exec(2), open(2), fcntl(7).

NAME

fork, vfork – create a new process

SYNOPSIS

int fork ( )
int vfork ( )

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fork causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process) except for the following:

The child process has a unique process ID.

The child process has a different parent process ID (i.e., the process ID of the parent process).

The child process has its own copy of the parent's file descriptors. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.

The child process's *utime*, *stime*, *cutime*, and *cstime* are set to **0**; see *times*(2).

The child's alarm time is cleared.

Fork returns a value of **0** to the child process.

Fork returns the process ID of the child process to the parent process.

Fork will fail and no child process will be created if one or more of the following are true:

The system-imposed limit on the total number of processes under execution would be exceeded. [EAGAIN]

The system-imposed limit on the total number of processes under execution by a single user would be exceeded. [EAGAIN]

The parent and child processes resume execution immediately after the *fork* call; they are identified by the value returned by *fork*.

Note that standard I/O buffers are duplicated in the child. Thus, if you fork after a buffered I/O operation that was not flushed, you may get duplicate output.

Vfork is a synonym for *fork*. *Vfork* is included for compatibility with other UNIX systems.

HARDWARE DEPENDENCIES

Series 210:
Fork will also fail if there is not enough swapping memory to create the new process. [ENOSPC]

Series 500/600/700:
Fork will also fail if there is not enough physical memory to create the new process. [ENOMEM]

RETURN VALUE

Upon successful completion, *fork* returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of –1 is returned to the parent process, no child process is created, and *errno* is set to indicate the error.

SEE ALSO

exec(2), wait(2).

NAME
gethostname – get name of current host

SYNOPSIS
char hostname[];
gethostname(hostname, sizeof (hostname));

HP-UX COMPATABILITY
Level:	HP-UX/RUN ONLY
Origin:	UCB

DESCRIPTION
Gethostname returns the standard host name for the current processor, as set by *sethostname*(2). The name is truncated to sizeof(hostname)-1 and is null-terminated.

SEE ALSO
sethostname(2).

NAME

getpid, getpgrp, getppid – get process, process group, and parent process IDs

SYNOPSIS

int getpid ( )

int getpgrp ( )

int getppid ( )

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Getpid returns the process ID of the calling process.

Getpgrp returns the process group ID of the calling process.

Getppid returns the parent process ID of the calling process.

SEE ALSO

exec(2), fork(2), setpgrp(2), signal(2).

NAME

getuid, geteuid, getgid, getegid – get real/effective user, real/effective group IDs

SYNOPSIS

int getuid ( )

int geteuid ( )

int getgid ( )

int getegid ( )

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Getuid returns the real user ID of the calling process.

Geteuid returns the effective user ID of the calling process.

Getgid returns the real group ID of the calling process.

Getegid returns the effective group ID of the calling process.

SEE ALSO

setuid(2).

NAME

ioctl – control device

SYNOPSIS

#include <sys/ioctl.h>

ioctl(fildes, request, arg)

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

Ioctl performs a variety of functions on special files (devices). The writeups of various devices in Section 4 discuss how *ioctl* applies to them.

Ioctl will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

The request is not appropriate to the selected device. [ENOTTY]

Request or *arg* is not valid. [EINVAL]

HARDWARE DEPENDENCIES

Series 500/600/700:

The include file **sys/ioctl.h** is not currently supported. Use instead **termio.h** (see *tty*(4)) or **sys/mtio.h** (see *mt*(4)), depending on the device. Note that these are only two of the available include files. The actual include file you use depends on the device.

RETURN VALUE

If an error has occurred, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

Section 4.

NAME

kill – send signal to process(s)

SYNOPSIS

int kill (pid, sig)
int pid, sig;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Kill sends a signal to a process or a group of processes. The process or group of processes to which the signal is to be sent is specified by *pid*. The signal that is to be sent is specified by *sig* and is either one from the list given in *signal*(2), or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. This can be used to check the validity of *pid*.

The effective user ID of the sending process must match the real user ID of the receiving process, unless the effective user ID of the sending process is super-user, or the process is sending to itself.

The processes with a process ID of 0 and a process ID of 1 are special processes (see *intro*(2)) and will be referred to below as *proc0* and *proc1* respectively.

If *pid* is greater than zero, *sig* will be sent to the process whose process ID is equal to *pid*. *Pid* may equal 1.

If *pid* is 0, *sig* will be sent to all processes excluding *proc0* and *proc1* whose process group ID is equal to the process group ID of the sender.

If *pid* is –1 and the effective user ID of the sender is not super-user, *sig* will be sent to all processes excluding *proc0* and *proc1* whose real user ID is equal to the effective user ID of the sender.

If *pid* is –1 and the effective user ID of the sender is super-user, *sig* will be sent to all processes excluding *proc0* and *proc1*.

If *pid* is negative but not –1, *sig* will be sent to all processes (excluding *proc1*) whose process group ID is equal to the absolute value of *pid*.

The algorithm for determining signal access rights to a prospective process (that meets *pid* requirements) is as follows:

```
if sender effective user id = super-user
or sender effective user id = target real user id
or sender process = target process
then it is legal to send the signal.
```

Also, if *pid* is less than or equal to zero, the signal is not sent to *proc1*.

Kill will fail and no signal will be sent if one or more of the following are true:

Sig is not a valid signal number. [EINVAL]

No process can be found corresponding to that specified by *pid*. [ESRCH]

The sending process is not sending to itself, its effective user ID is not super-user, and its effective user ID does not match the real user ID of the receiving process. [EPERM]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

kill(1), getpid(2), setpgrp(2), signal(2).

NAME

link – link to a file

SYNOPSIS

int link (path1, path2)
char *path1, *path2;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path1 points to a path name naming an existing file. *Path2* points to a path name naming the new directory entry to be created. *Link* creates a new link (directory entry) for the existing file.

Link will fail and no link will be created if one or more of the following are true:

A component of either path prefix is not a directory. [ENOTDIR]

A component of either path prefix does not exist. [ENOENT]

A component of either path prefix denies search permission. [EACCES]

The file named by *path1* does not exist. [ENOENT]

The link named by *path2* exists. [EEXIST]

The file named by *path1* is a directory and the effective user ID is not super-user. [EPERM]

The link named by *path2* and the file named by *path1* are on different logical devices (file systems). [EXDEV]

Path2 points to a null path name. [ENOENT]

The requested link requires writing in a directory with a mode that denies write permission. [EACCES]

The requested link requires writing in a directory on a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

Path1 or *path2* is null. [ENOENT]

Too many links to *path1*. [EMLINK]

HARDWARE DEPENDENCIES

Series 500/600/700:

For Structured Directory Format (SDF) discs, if *path2* is "..", then that directory's i-node will be altered such that its ".." entry points to the directory specified by *path1*. In this way, the super-user can establish the parent directory of an existing directory.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

cp(1), link(1), unlink(2).

NAME

lseek – move read/write file pointer; seek

SYNOPSIS

long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fildes is a file descriptor returned from a *creat*, *open*, *dup*, or *fcntl* system call. *Lseek* sets the file pointer associated with *fildes* as follows:

If *whence* is 0, the pointer is set to *offset* bytes.

If *whence* is 1, the pointer is set to its current location plus *offset*.

If *whence* is 2, the pointer is set to the size of the file plus *offset*.

Upon successful completion, the resulting pointer location as measured in bytes from the beginning of the file is returned.

Lseek will fail and the file pointer will remain unchanged if one or more of the following are true:

Fildes is not an open file descriptor. [EBADF]

Fildes is associated with a pipe or fifo. [ESPIPE]

Whence is not 0, 1 or 2. [EINVAL and SIGSYS signal]

The resulting file pointer would be negative. [EINVAL]

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

RETURN VALUE

Upon successful completion, a non-negative integer indicating the file pointer value is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

creat(2), dup(2), fcntl(2), open(2).

NAME

memadvise – advise OS about segment reference patterns

SYNOPSIS

#include <sys/ems.h>

```
memadvise(addr,len,behav,adrtype);
caddr_t         addr;
int             len,behav;
enum            memtype {mem_code,mem_data} adrtype;
```

HP-UX COMPATIBILITY

Level:	Backing Store Control - HP-UX/EXTENDED
Origin:	HP
Remarks:	*Memadvise* is currently implemented on the Series 500/600/700 only.

DESCRIPTION

The purpose of this call is to allow an application program to notify the system of its known patterns of reference in specific areas of process memory. The intent is to allow the system to then adapt its memory management algorithms and/or policies based on this knowledge to maximize the performance of the program. For example, a program that uses a very large hash table might inform the system of its random patterns of reference to this area. The system might, then, elect not to do any pre-fetching or clustered reads in this area.

Addr is the starting address of the area in question and *len* is the length in bytes. *Addr* may be any legal address in the process's address space. Since some implementations use different (and indistinguishable) addressing formats for code and data space, *adrtype* is used to indicate whether *addr* is a code or data address. On systems with a uniform addressing format for code and data, *adrtype* will have no effect.

The boundaries of the address space for which the advice is applied may be rounded up and/or down to appropriate system dependent values (e.g. pages, segments, blocks, etc).

Variable *behav* describes the reference pattern in the specified area:

MEM_NORMAL
: No known extraordinary patterns of reference.

MEM_SEQ
: References are highly sequential in nature.

MEM_RANDOM
: References are totally random and unpredictible.

MEM_NEEDED
: Area is expected to be highly referenced in near future.

MEM_NOTNEEDED
: Area is not expected to be referenced in the near future.

Memadvise may be reduced to a no-op, or some of the behaviour types may be ignored (treated as no-ops).

SEE ALSO

ems(2), memallc(2)

NAME

memallc, memfree – allocate and free address space

SYNOPSIS

#include <sys/ems.h>

```
caddr_t     memallc(fileid, offset, len, maxlen, type, mode);
int         fileid, offset, len;
int         maxlen, type, mode;

int         memfree(addr);
caddr_t     addr;
```

HP-UX COMPATIBILITY

Level: Extended Memory - HP-UX/EXTENDED

Origin: HP

Remarks: *Memallc* and *memfree* are currently implemented on the Series 500/600/700 only.

DESCRIPTION

Memallc allocates a memory segment (i.e. a contiguous piece of process address space) and returns a pointer to it. The memory segment may be shared (i.e. accessible by other processes) or private. Private segments are copied on *fork*(2), giving separate, per-process images of the segment. Shared segments are not copied across *fork*(2) but, instead, both processes have access to the same memory space. The segment may optionally be initialized to the contents of a specific open file (private mapped file) or can be made equivalent to a specific file (shared mapped file).

Fileid is the HP-UX file id of an open file which will be mapped into the process's address space. If *fileid* is -1, the allocated address space will be initialized to zeros. A mapping of a file (either shared or private) generates an implicit reference to the file (similar to the result of *dup*(2)). Subsequent to the mapping, *fileid* may safely be closed.

Offset specifies the starting point in *fileid* (i.e. byte offset) where mapping is to begin. The value returned by *memallc* is a pointer to the byte in the new address space that corresponds to byte *offset*. If *fileid* is not specified (i.e. set to -1), *offset* is ignored.

Len specifies the size (in bytes) of the address space. The guaranteed range of accessability is from *ptr* thru *ptr + len-1* (where *ptr* is the value returned by the *memallc* call). Depending on the value of *offset*, *len*, and the specific implementation, additional data space MAY be accessible at addresses less than *ptr* and/or greater than *ptr + len-1* but the effects of reading and/or writing these areas are undefined.

If *len + offset* is greater than the size of the file, the additional address space is initialized to zeros. If the segment is shared, the file is extended to the required size (if fileid is not writable, the call fails). A *creat*(2) call on a file that has a shared mapping applied to it will zero the file but will not alter the file size.

Maxlen specifies the maximum length to which a segment may grow using *memvary*(2).

Type specifies the attributes assigned to the segment, which is constructed by taking the union of the desired attributes: MEM_SHARED, MEM_PRIVATE, MEM_PAGED, MEM_DATA, or MEM_CODE (see *ems*(2)).

Mode specifies the access permissions assigned to the segment for the requesting process.

MEM_R, MEM_W, MEM_X:
: Initial access modes to be assigned to segment (see *memchmd*(2)).

Note that all MEM_SHARED mappings of a specific file must use identical access modes. An attempt to map a file with access modes different than those already in effect will return an error [EACCESS].

Memfree deallocates a memory segment created by *memallc*. It takes, as an argument, a pointer returned by *memallc*. When the segment is shared, the memory will not be deallocated until the last reference to the memory is removed.

The number of segments allocated to a given process at any one time may be limited to a system dependent maximum of **MAXSEGS** found in **ems.h**.

HARDWARE DEPENDENCIES

Series 500/600/700:

The file ID *fileid* must refer to a file on a CS-80 disc.

RETURN VALUE

Upon successful completion, *memallc* returns the byte pointer to the address space. Otherwise, a value of -1 is returned and *errno* is set to indicate error.

SEE ALSO

ems(2), memvary(2), memchmd(2).

NAME

memchmd – change memory segment access modes

SYNOPSIS

#include <sys/ems.h>

```
int             memchmd(addr,mode);
caddr_t         addr;
int             mode;
```

HP-UX COMPATIBILITY

Level:	Extended Memory - HP-UX/EXTENDED
Origin:	HP
Remarks:	*Memchmd* is currently implemented on the Series 500/600/700 only.

DESCRIPTION

This procedure may be used to change the access mode of a memory segment created by *memallc*(2). The procedure returns the previous access mode (or -1 if there is an error).

Addr is the segment pointer returned by *memallc*(2).

The access modes for a shared segment is an attribute of the segment and is the same for all processes sharing the segment or any portion thereof. The access mode of a segment may not be changed if it is being shared with any other process (e.g. more than one *memallc* of a paricular file, or a *memallc* followed by a *fork*(2)). An attempt to *memchmd* such a shared segment will return an error [EACCESS].

The access mode of a MEM_PRIVATE segment may be changed without restrictions.

The definition of the access modes are:

MEM_X	Execute capability
MEM_W	Write capability
MEM_R	Read capability

An error is returned if *addr* is not a valid segment pointer.

Access modes granted to a MEM_SHARED file mapped segment may not exceed the access modes granted to the user of the file when it was opened.

RETURN VALUE

Upon successful completion, *memchmd*(2) returns the old set of access modes. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

SEE ALSO

ems(2), memallc(2), memvary(2).

NAME

memlck, memulck – lock/unlock process address space or segment

SYNOPSIS

```
#include <sys/ems.h>
int          memlck (addr, len, adrtype);
caddr_t      addr;
int          len;
enum    memtype {mem_code, mem_data} adrtype;

int          memulck (addr, len, adrtype);
caddr_t      addr;
int          len;
enum    memtype {mem_code, mem_data} adrtype;
```

HP-UX COMPATIBILITY

Level: Backing Store Control - HP-UX/RUN ONLY

Origin: HP

Remarks: *Memlck* and *memulck* are currently implemented on the Series 500/600/700 only.

DESCRIPTION

Memlck is used to lock a section of process address space into physical memory. This call may take a substantial amount of time to complete, but the address space in question is guaranteed to be in memory and locked upon successful completion of the call. The locked address space will not migrate to backing store regardless of process state and will, furthermore, remain at the same physical address space for the duration of the lock. Locks are not inherited across *fork*(2). Multiple locks on any address range can occur (unlocking requires that as many unlocks as locks occur). The locks will be segment local, and unlocking may be done by a process unrelated to the one which did the locking. A locked segment will be released when there are no processes with references to the locked segment. (This may occur either via *memfree*(2) or process death.)

Addr is the starting address of the area in question and *len* is the length in bytes. *Addr* may be any legal address in the process's address space. Since some implementations use different (and indistinguishable) addressing formats for code and data space, *adrtype* is used to indicate whether addr is a code or data address. On systems with a uniform addressing format for code and data, *adrtype* will have no effect.

The boundaries of the locked address space may be rounded up (on the upper end of the address range) and down (on the lower end of the address range) to appropriate system dependent values (e.g. pages, segments, blocks, etc). Locking will not cross segment boundaries. For example, one *memlck* call cannot lock part of a text segment and part of a data segment.

Memulck undoes the effects of a *memlck*.

The use of this call is restricted to the super-user.

This call may be reduced to a no-op.

RETURN VALUE

Upon successful completion, *memlck* and *memulck* return a value of 0. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

SEE ALSO

ems(2), memallc(2).

NAME

memvary – modify segment length

SYNOPSIS

#include <sys/ems.h>
int memvary(addr, len);
caddr_t addr;
int len;

HP-UX COMPATIBILITY

Level:	Extended Memory - HP-UX/EXTENDED
Origin:	HP
Remarks:	*Memvary* is currently implemented on the Series 500/600/700 only.

DESCRIPTION

Memvary allows the modification of the size of the memory space allocated by *memallc*(2).

Addr is the pointer to the address space which can be either shared or private. If the address space has been file mapped and is extended beyond the the end of the file, then the file will also reflect the change in length. When the file mapped address space is reduced, the actual file length will be unchanged and the file space after the end of the mapped file space will also remain unchanged. A change in length for a private file mapped address space will have no effect on the source file.

Len specifies the new length of the address space. In the case of an error, the address space and file space will be the same as before the intrinsic call.

When private file mapped address space is extended, the additional address space is initialized to zeroes. When shared file mapped address space is extended, the additional space is initialized to the contents of the file, or zeros if the file is extended.

The address space cannot be extended beyond the 'maxlen' specified during the *memallc*(2) intrinsic call.

RETURN VALUE

Upon successful completion, *memvary* returns 0. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

SEE ALSO

ems(2), memallc(2), memchmd(2).

NAME

mknod – make directory, special or ordinary file

SYNOPSIS

#include <mknod.h>
int mknod (path, mode, dev)
char *path;
int mode;
dev_t dev;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Mknod creates a new file named by the path name pointed to by *path*. The mode of the new file is initialized from *mode*, where the value of *mode* is interpreted as follows:

```
0170000 file type; one of the following:
        0010000 fifo special
        0020000 character special
        0040000 directory
        0060000 block special
        0100000 or 0000000 ordinary file
0004000 set user ID on execution
0002000 set group ID on execution
0001000 save text image after execution
0000777 access permissions; constructed from the following:
        0000400 read by owner
        0000200 write by owner
        0000100 execute (search on directory) by owner
        0000070 read, write, execute (search) by group
        0000007 read, write, execute (search) by others
```

Values of *mode* other than those above are undefined and should not be used.

The file's owner ID is set to the process's effective user ID. The file's group ID is set to the process's effective group ID.

The low-order 9 bits of *mode* are modified by the process's file mode creation mask: all bits set in the process's file mode creation mask are cleared. See *umask*(2).

Dev is meaningful only if *mode* indicates a block or character special file, and is ignored otherwise. It is an implementation dependent specification of a character or block I/O device. A device name is created by using the **makedev** macro defined in *mknod.h*. The arguments to **makedev** are the major and minor device numbers, the value and interpretation of which are implementation dependent. The result is an object of type **dev_t**.

Mknod may be invoked only by the super-user for file types other than FIFO special.

Mknod will fail and the new file will not be created if one or more of the following are true:

The process's effective user ID is not super-user. [EPERM]

A component of the path prefix is not a directory. [ENOTDIR]

A component of the path prefix does not exist. [ENOENT]

The directory in which the file is to be created is located on a read-only file system. [EROFS]

The named file exists. [EEXIST]

Path points outside the process's allocated address space. [EFAULT]

Path is null. [ENOENT]

Path is in a directory that denies write permission, *mode* is for fifo special file, and the caller is not super-user. [EACCES]

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

mkdir(1), chmod(2), exec(2), umask(2), fs(5), mknod(5), mknod(8).

NAME

mount – mount a file system

SYNOPSIS

int mount (spec, dir, rwflag)
char *spec, *dir;
int rwflag;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Mount requests that a removable file system contained on the block special device identified by *spec* be mounted on the directory identified by *dir*. *Spec* and *dir* are pointers to path names.

Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

The low-order bit of *rwflag* is used to control write permission on the mounted file system; if **1**, writing is forbidden, otherwise writing is permitted according to individual file accessibility.

Mount may be invoked only by the super-user.

Mount will fail if one or more of the following are true:

The effective user ID is not super-user. [EPERM]

Any of the named files does not exist. [ENOENT]

A component of a path prefix is not a directory. [ENOTDIR]

Spec is not a block special device. [ENOTBLK]

The device associated with *spec* does not exist. [ENXIO]

Dir is not a directory. [ENOTDIR]

Spec or *dir* points outside the process's allocated address space. [EFAULT]

Dir is currently mounted on, is someone's current working directory or is otherwise busy. [EBUSY]

The device associated with *spec* is currently mounted. [EBUSY]

Spec or *dir* is null. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

mount(1), umount(2).

BUGS

If *mount* is called from the program level (i.e. not called from *mount*(1)), the table of mounted devices contained in /etc/mnttab is not updated.

NAME

nice – change priority of a process

SYNOPSIS

int nice (incr)
int incr;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Nice adds the value of *incr* to the nice value of the calling process. A process's *nice value* is a positive number for which a more positive value results in lower CPU priority.

A maximum nice value of 39 and a minimum nice value of 0 are imposed by the system. Requests for values above or below these limits result in the nice value being set to the corresponding limit.

Nice will fail and not change the nice value if *incr* is negative and the effective user ID of the calling process is not super-user. [EPERM]

Note that *nice* assumes a user process priority value of 20. If the super-user of your system has changed the user process priority value to something less than 20, certain increments can cause *nice* to return –1, which is indistinguishable from an error return.

RETURN VALUE

Upon successful completion, *nice* returns the new nice value minus 20. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

nice(1), exec(2).

NAME

open – open file for reading or writing

SYNOPSIS

#include <fcntl.h>
int open (path, oflag[, mode])
char *path;
int oflag, mode;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to a path name naming a file. *Open* opens a file descriptor for the named file and sets the file status flags according to the value of *oflag*. *Oflag* values are constructed by or-ing flags from the following list. Note that one and only one of the first three flags below **must** be used.

O_RDONLY Open for reading only.

O_WRONLY Open for writing only.

O_RDWR Open for reading and writing.

O_NDELAY This flag may affect subsequent reads and writes. See *read*(2) and *write*(2).

When opening a FIFO with O_RDONLY or O_WRONLY set:

If O_NDELAY is set:

An *open* for reading-only will return without delay. An *open* for writing-only will return an error if no process currently has the file open for reading.

If O_NDELAY is clear:

An *open* for reading-only will block until a process opens the file for writing. An *open* for writing-only will block until a process opens the file for reading.

When opening a file associated with a communication line:

If O_NDELAY is set:

The open will return without waiting for carrier.

If O_NDELAY is clear:

The open will block until carrier is present.

O_APPEND If set, the file pointer will be set to the end of the file prior to each write.

O_CREAT If the file exists, this flag has no effect. Otherwise, the file's owner ID is set to the process's effective user ID, the file's group ID is set to the process's effective group ID, and the low-order 12 bits of the file mode are set to the value of *mode* modified as follows (see *creat*(2)):

All bits set in the process's file mode creation mask are cleared. See *umask*(2).

The "save text image after execution bit" of the mode is cleared. See *chmod*(2).

O_TRUNC If the file exists, its length is truncated to 0 and the mode and owner are unchanged.

O_EXCL If O_EXCL and O_CREAT are set, *open* will fail if the file exists.

Upon successful completion a non-negative integer, the file descriptor, is returned.

The file pointer used to mark the current position within the file is set to the beginning of the file.

The new file descriptor is set to remain open across *exec* system calls. See *fcntl*(2).

No process may have more than 20 file descriptors open simultaneously.

The named file is opened unless one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

O_CREAT is not set and the named file does not exist. [ENOENT]

A component of the path prefix denies search permission. [EACCES]

Oflag permission is denied for the named file. [EACCES]

The named file is a directory and *oflag* is write or read/write. [EISDIR]

The named file resides on a read-only file system and *oflag* is write or read/write. [EROFS]

Twenty (20) file descriptors are currently open. [EMFILE]

The named file is a character special or block special file, and the device associated with this special file does not exist. [ENXIO]

The file is a pure procedure (shared text) file that is being executed and *oflag* is write or read/write. [ETXTBSY]

Path points outside the process's allocated address space. [EFAULT]

O_CREAT and O_EXCL are set, and the named file exists. [EEXIST]

O_NDELAY is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading. [ENXIO]

Path is null. [ENOENT]

Oflag specifies both O_WRONLY and O_RDWR. [EINVAL]

RETURN VALUE

Upon successful completion, a non-negative integer, namely a file descriptor, is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

close(2), creat(2), dup(2), fcntl(2), lseek(2), read(2), write(2).

NAME
pause – suspend process until signal

SYNOPSIS
pause ( )

HP-UX COMPATIBILITY
Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION
Pause suspends the calling process until it receives a signal. The signal must be one that is not currently set to be ignored by the calling process.

If the signal causes termination of the calling process, *pause* will not return.

If the signal is *caught* by the calling process and control is returned from the signal catching-function (see *signal*(2)), the calling process resumes execution from the point of suspension; with a return value of −1 from *pause* and *errno* set to EINTR.

SEE ALSO
alarm(2), kill(2), signal(2), wait(2).

NAME

pipe – create an interprocess channel

SYNOPSIS

int pipe (fildes)
int fildes[2];

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Pipe creates an I/O mechanism called a pipe and returns two file descriptors, *fildes*[0] and *fildes*[1]. *Fildes*[0] is opened for reading and *fildes*[1] is opened for writing.

Writes up to 5120 bytes of data are buffered by the pipe before the writing process is blocked. A read on file descriptor *fildes*[0] accesses the data written to *fildes*[1] on a first-in-first-out basis.

No process may have more than 20 file descriptors open simultaneously.

Pipe will fail if 19 or more file descriptors are currently open. [EMFILE]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

sh(1), read(2), write(2), popen(3S).

NAME
read – read from file

SYNOPSIS
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

HP-UX COMPATIBILITY
Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION
Fildes is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

Read attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*.

On devices capable of seeking, the *read* starts at a position in the file given by the file pointer associated with *fildes*. Upon return from *read*, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, *read* returns the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if the file is associated with a communication line (see *ioctl*(2) and *tty*(4)), or if the number of bytes left in the file is less than *nbyte* bytes. A value of 0 is returned when an end-of-file has been reached.

When attempting to read from an empty pipe (or FIFO):

If O_NDELAY is set, the read will return a 0.

If O_NDELAY is clear, the read will block until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently available:

If O_NDELAY is set, the read will return a 0.

If O_NDELAY is clear, the read will block until data becomes available.

Read will fail if one or more of the following are true:

Fildes is not a valid file descriptor open for reading. [EBADF]

Buf points outside the allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion a non-negative integer is returned indicating the number of bytes actually read. Otherwise, a –1 is returned and *errno* is set to indicate the error.

SEE ALSO
creat(2), dup(2), fcntl(2), ioctl(2), open(2), pipe(2), ustat(2), tty(4).

BUGS
Reading from raw disc device files must be done in multiples of the device's physical record size (see *ustat*(2)), and must begin on a physical sector boundary.

NAME

sethostname – set name of host cpu

SYNOPSIS

sethostname(name, namelen)
char *name;
int namelen;

HP-UX COMPATABILITY

Level:	HP-UX/RUN ONLY
Origin:	UCB

DESCRIPTION

This call sets the name of the host processor to be ***name,*** which has a length of ***namelen*** characters. The maximum value of ***namelen*** is determined by the ***uname*** structure. This is normally executed when the system is bootstrapped, executed out of the file **/etc/rc.** The name set should not be a nickname for the machine, but the full name of the machine, i.e. "hpdcda". This intrinsic sets the ***nodename*** field in the ***utsname*** structure returned by ***uname***(2).

Sethostname can only be executed by the super-user.

SEE ALSO

hostname(1), uname(1), gethostname(2), uname(2).

NAME

setpgrp – set process group ID

SYNOPSIS

int setpgrp ( )

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Setpgrp sets the process group ID of the calling process to the process ID of the calling process and returns the new process group ID. The new process group ID is always identical to the process ID of the calling process.

Setpgrp breaks the calling process's terminal affiliation. See *tty*(4).

RETURN VALUE

Setpgrp returns the value of the new process group ID.

SEE ALSO

exec(2), fork(2), getpid(2), kill(2), signal(2).

NAME

setuid, setgid – set user and group IDs

SYNOPSIS

int setuid (uid)
int uid;

int setgid (gid)
int gid;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Setuid is used to set the real user ID and effective user ID of the calling process.

Setgid is used to set the real group ID and effective group ID of the calling process.

If the effective user ID of the calling process is super-user, the real user (group) ID and effective user (group) ID are set to *uid* (*gid*).

If the effective user ID of the calling process is not super-user, but its real user (group) ID is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

Setuid will fail if the real user (group) ID of the calling process is not equal to *uid* (*gid*) and its effective user ID is not super-user. [EPERM] It will also fail if *uid* (*gid*) is not a valid user (group) ID [EINVAL].

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

getuid(2).

NAME

signal – set up signal handling for program

SYNOPSIS

#include <signal.h>

int (*signal (sig, func))( )
int sig;
int (*func)( );

func(signo [, code, scp])
int signo, code;
struct sigcontext *scp;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Signal allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. *Sig* specifies the signal and *func* specifies the choice.

Sig can be assigned any one of the following except **SIGKILL**:

SIGHUP	01	hangup
SIGINT	02	interrupt
SIGQUIT	03*	quit
SIGILL	04*	illegal instruction (not reset when caught)
SIGTRAP	05*	trace trap (not reset when caught)
SIGIOT	06*	software generated (sent by *abort*(3C))
SIGEMT	07*	software generated
SIGFPE	08*	floating point exception
SIGKILL	09	kill (cannot be caught or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCLD	18	death of a child (see *WARNING* below)
SIGPWR	19	power fail (see *WARNING* below)

See below for the significance of the asterisk in the above list.

Func is assigned one of three values: **SIG_DFL**, **SIG_IGN**, or a *function address*. The actions prescribed by these values are as follows:

SIG_DFL – terminate process upon receipt of a signal.
Upon receipt of the signal *sig*, the receiving process is to be terminated with the following consequences:

All of the receiving process's open file descriptors will be closed.

If the parent process of the receiving process is executing a *wait*, it will be notified of the termination of the receiving process and the terminating signal's number will be made available to the parent process; see *wait*(2).

If the parent process of the receiving process is not executing a *wait*, the receiving process will be transformed into a zombie process (except in the case where **SIGCLD** is set to be ignored; see *WARNING*). See *exit*(2) for a definition of zombie process.

The parent process ID of each of the receiving process's existing child processes and zombie processes will be set to 1. This means the initialization process (*proc1*) (see glossary) inherits each of these processes.

An accounting record will be written on the accounting file if the system's accounting routine is enabled; see *acct*(2).

If the receiving process's process ID, tty group ID, and process group ID are equal, the signal **SIGHUP** will be sent to all of the processes that have a process group ID equal to the process group ID of the receiving process.

A "core image" will be made in the current working directory of the receiving process if *sig* is one for which an asterisk appears in the above list *and* the following conditions are met:

> The effective user ID and the real user ID of the receiving process are equal.
>
> An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:
>
> > a mode of 0666 modified by the file creation mask (see *umask*(2))
> >
> > a file owner ID that is the same as the effective user ID of the receiving process
> >
> > a file group ID that is the same as the effective group ID of the receiving process

SIG_IGN – ignore signal
The signal *sig* is to be ignored.

Note: the signal **SIGKILL** cannot be ignored.

function address – catch signal
Upon receipt of the signal *sig*, the receiving process is to execute the signal-catching function pointed to by *func*. The signal number *sig* will be passed as the first parameter to the signal-catching function. The HP-UX kernel will also pass three additional (optional) parameters to signal handler routines. The parameters are:

signo	signal number
code	a word of information usually provided by the hardware.
scp	a pointer to a machine dependent structure defined in the include file signal.h.

Depending on the value of *signo*, *code* may be zero and/or *scp* may be NULL. The meanings of *code* and *scp* and the conditions upon which they are other than zero or NULL are implementation dependent. It is permissible for *code* and *signo* to always be zero, and *scp* to always be NULL.

The pointer scp will only be valid during the context of the signal handler.

The optional parameters can be omitted from the handler parameter list, in which case the handler is exactly compatible with System III UNIX.

Truly portable software should not use the optional parameters in signal-catching routines.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted and the value of *func* for the caught signal will be set to **SIG_DFL** unless the signal is **SIGILL**, **SIGTRAP**, **SIGCLD**, or **SIGPWR**.

When a signal that is to be caught occurs during a *read*, a *write*, an *open*, or an *ioctl* system call on a slow device (like a terminal, but not a file), during a *pause* system call, or during a *wait* system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal catching function will be executed and then the interrupted system call will return a −1 to the calling process with *errno* set to EINTR.

Note: the signal **SIGKILL** cannot be caught.

A call to *signal* cancels a pending signal *sig* except for a pending **SIGKILL** signal.

Signal will fail if one or more of the following are true:

Sig is an illegal signal number, or is equal to **SIGKILL**. [EINVAL]

Func points to an illegal address. [EFAULT]

HARDWARE DEPENDENCIES

Series 210:

The signal **SIGPWR** is not currently generated.

Series 500/600/700:

Accounting is not currently supported.

Core image files are currently files with zero length.

The **SIGEMT** signal means "out of memory", and is generated by the HP-UX Operating System.

SIGIOT can be sent if an invalid string operation is attempted, or if a bounds range check trap is encountered.

The signals **SIGBUS** and **SIGPWR** are not currently generated.

The signal handler parameter *code* contains the trap number provided by the hardware; see *trapno*(2).

A zero value is returned on floating point underflow. Floating point overflow, divide-by-zero, integer divide-by-zero, and illegal floating point operation exceptions result in the signal **SIGFPE** being sent to the process. An undefined value is returned as the result of the operation if the signal **SIGFPE** is ignored or caught.

SIGTERM is not sent on integer overflow.

RETURN VALUE

Upon successful completion, *signal* returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SIGKILL can be sent in the event of an unsuccessful *exec*, if the original program has already been deleted.

SEE ALSO

kill(1), kill(2), lseek(2), pause(2), ptrace(2), trapno(2), wait(2), abort(3C), setjmp(3C).

WARNING

Two signals that behave differently from the signals described above exist in this release of the system; they are:

SIGCLD	18	death of a child (not reset when caught)
SIGPWR	19	power fail (not reset when caught)

For these signals, *func* is assigned one of three values: **SIG_DFL**, **SIG_IGN**, or a *function address*. The actions prescribed by these values are as follows:

SIG_DFL - ignore signal
The signal is to be ignored.

SIG_IGN - ignore signal
The signal is to be ignored. Also, if *sig* is **SIGCLD**, the calling process's child processes will not create zombie processes when they terminate; see *exit*(2).

function address - catch signal
If the signal is **SIGPWR**, the action to be taken is the same as that described above for *func* equal to *function address*. The same is true if the signal is **SIGCLD**, except that while the process is executing the signal-catching function, any received **SIGCLD** signals will be queued, and the signal-catching function will be continually reentered until the queue is empty.

The **SIGCLD** affects two other system calls (*wait*(2), and *exit*(2)) in the following ways:

wait If the *func* value of **SIGCLD** is set to **SIG_IGN** and a *wait* is executed, the *wait* will block until all of the calling process's child processes terminate; it will then return a value of −1 with *errno* set to ECHILD.

exit If in the exiting process's parent process the *func* value of **SIGCLD** is set to **SIG_IGN**, the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the preceding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set **SIGCLD** to be caught.

Some implementations do not generate **SIGPWR**. For systems without non-volatile memory, it is not useful. If **SIGPWR** is generated, it occurs when power is restored and the system has done all necessary re-initialization. Processes will re-start by responding to **SIGPWR**.

NAME

stat, fstat – get file status

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

int stat (path, buf)
char *path;
struct stat *buf;

int fstat (fildes, buf)
int fildes;
struct stat *buf;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to a path name naming a file. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable. *Stat* obtains information about the named file.

Similarly, *fstat* obtains information about an open file known by the file descriptor *fildes*, obtained from a successful *open*, *creat*, *dup*, *fcntl*, or *pipe* system call.

Buf is a pointer to a *stat* structure into which information is placed concerning the file.

The contents of the structure pointed to by *buf* include the following members:

```
dev_t   st_dev;    /* ID of device containing a dir entry for this file */
ino_t   st_ino;    /* Inode number */
ushort  st_mode;   /* File mode; see mknod(2) */
short   st_nlink;  /* Number of links */
ushort  st_uid;    /* User ID of file owner */
ushort  st_gid;    /* Group ID of file group */
dev_t   st_rdev;   /* Device ID; defined only for char, blk spec files */
off_t   st_size;   /* File size (bytes) */
time_t  st_atime;  /* Time of last access */
time_t  st_mtime;  /* Last modification time */
time_t  st_ctime;  /* Last file status change time */
                   /* Measured in secs since 00:00:00 GMT, Jan 1, 1970 */
```

st_size For special files which refer to discs, either returns the total physical size (in bytes) of the mass storage volume, when appropriate, or -1 otherwise. This is a property of the physical device, not any directory structure imposed upon it.

st_atime Time when file data was last accessed. Changed by the following commands and system calls: *touch*(1), *creat*(2), *mknod*(2), *pipe*(2), *utime*(2), and *read*(2).

st_mtime Time when data was last modified. Changed by the following commands and system calls: *touch*(1), *creat*(2), *mknod*(2), *pipe*(2), *utime*(2), and *write*(2).

st_ctime Time when file status was last changed. Changed by the following commands and system calls: *touch*(1), *chmod*(2), *chown*(2), *creat*(2), *link*(2), *mknod*(2), *pipe*(2), *unlink*(2), *utime*(2), and *write*(2).

Stat will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix. [EACCES]

Buf or *path* points to an invalid address. [EFAULT]

Path is null. [ENOENT]

Fstat will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Buf points to an invalid address. [EFAULT]

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

touch(1), chmod(2), chown(2), creat(2), link(2), mknod(2), time(2), unlink(2), stat(7).

NAME

stime – set time and date

SYNOPSIS

int stime (tp)
long *tp;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Stime sets the system's idea of the time and date. *Tp* points to the value of time as measured in seconds from 00:00:00 GMT January 1, 1970.

Stime will fail if the effective user ID of the calling process is not super-user. [EPERM]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

time(2).

NAME

stty, gtty – control device

SYNOPSIS

#include <sgtty.h>

stty(fildes,argp)
struct sgttyb *argp;

gtty(fildes,argp)
struct sgttyb *argp;

HP-UX COMPATIBILITY

Level:	Version 6 Compatibility - HP-UX/STANDARD
Origin:	Version 6
Remarks:	This system call is preserved for backward compatibility with Bell Version 6. It provides as close an approximation as possible to the old Version 6 function. All new code should use the TCSETA/TCGETA *ioctl* calls described in *tty*(4). Note that these calls are incompatible with the Version 7 calls of the same names.

DESCRIPTION

For certain status setting and status inquiries about terminal devices, the functions *stty* and *gtty* are equivalent to

```
ioctl(fildes, TIOCSETP, argp)
ioctl(fildes, TIOCGETP, argp)
```

respectively; see *tty*(4).

SEE ALSO

stty(1), exec(2), tty(4).

DIAGNOSTICS

Zero is returned if the call was successful; -1 if the file descriptor does not refer to the kind of file for which it was intended.

NAME

sync – update super-block

SYNOPSIS

sync ( )

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Sync causes all information in memory that should be on disc to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs which examine a file system, for example *fsck*, *df*, etc. It is mandatory before a boot.

The writing, although scheduled, is not necessarily complete upon return from *sync*.

Sync may be reduced to a no-op. This is permissible on a system which does not cache buffers, or in a system that in some way ensures that the discs are always in a consistent state.

SEE ALSO

sync(8).

NAME

time – get time

SYNOPSIS

long time ((long *) 0)

long time (tloc)
long *tloc;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Time returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.

If *tloc* (taken as an integer) is non-zero, the return value is also stored in the location to which *tloc* points.

Time will fail if *tloc* points to an illegal address. [EFAULT]

HARDWARE DEPENDENCIES

Series 210:

Tloc is not checked for an illegal address. If *tloc*'s value is an illegal address, a run-time error occurs.

RETURN VALUE

Upon successful completion, *time* returns the value of time. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

stime(2).

NAME

times – get process and child process times

SYNOPSIS

#include <sys/times.h>

long times (buffer)
struct tms *buffer;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Times fills the structure pointed to by *buffer* with time-accounting information. This information comes from the calling process and each of its terminated child processes for which it has executed a *wait*. The structure defined in **sys/times.h** looks as follows:

```
struct tms {
   time_t tms_utime;   /* user time */
   time_t tms_stime;   /* system time */
   time_t tms_cutime;  /* user time, children */
   time_t tms_cstime;  /* system time, children */
};
```

All times are in 60ths of a second.

Utime is the CPU time used while executing instructions in the user space of the calling process.

Stime is the CPU time used by the system on behalf of the calling process.

Cutime is the sum of the *utime*s and *cutime*s of the child processes.

Cstime is the sum of the *stime*s and *cstime*s of the child processes.

Times will fail if *buffer* points to an illegal address. [EFAULT]

HARDWARE DEPENDENCIES

Series 600/700:

The child CPU times listed can be greater than the actual elapsed real time, since the CPU time is counted on a per-CPU basis. Thus, if all three CPUs are executing, the CPU time is the sum of the three execution times of the CPUs.

RETURN VALUE

Upon successful completion, *times* returns the elapsed real time, in 60ths of a second, since an arbitrary point in the past (e.g., system start-up time). This point does not change from one invocation of *times* to another. If *times* fails, a –1 is returned and *errno* is set to indicate the error.

SEE ALSO

exec(2), fork(2), time(2), wait(2).

BUGS

Not all CPU time expended by system processes on behalf of a user process is counted in the system CPU time for that process.

NAME

trapno – hardware trap numbers

HP-UX COMPATIBILITY

Level: HP-UX/NON-STANDARD

Origin: HP

Remarks: The following description of hardware trap numbers is valid for the Series 500/600/700 only.

DESCRIPTION

The following trap numbers refer to hardware traps occurring on the HP 9000 Series 500/600/700 computers. *Trapno* values are reported by the *err*(1) command, and are passed to signal handlers (see *signal*(2)) when hardware traps cause signals to be sent to the current process.

The *trapno* value, trap name, and description are listed below for each possible trap condition. By convention, trap numbers are shown in octal.

VALUE	NAME: DESCRIPTION
01	Bounds Violation: An address is outside the limits for the program, stack, or global data segments. [2]
02	Check Trap: A user value is outside a prescribed range. [1]
03	Breakpoint Trap: Debugging trap. [1]
04	Machine Instruction Trap: Used by the operating system.
05	String Trap: Illegal string operation or data. [2]
06	Unused.
07	Unused.
010	Reset: Used by the operating system.
011	Page Table Violation: The page table entry referenced is beyond the current length of the page table. [2]
012	Inconsistent Registers: An attempt was made to set up an inconsistent set of registers describing the global data segment, stack segment, or program segment. [2]
013	External Data Segment Bounds Violation: An address is outside the limits of an external data segment. [2]
014	System Error: Used by the operating system.
015	External Data Segment Pointer Violation: Illegal data segment pointer; probably a pointer between 0 and 524287 decimal. [2]
016	Pointer Conversion Violation: An attempt was made to form a data segment pointer with an offset which is too large for the type of pointer being used. [2]
017	External Program Pointer Violation: Illegal procedure pointer. [2]
020	Unimplemented Instruction: Attempt to execute an undefined instruction. [1]
021	STT Violation: Illegal procedure pointer. [2]
022	CST Violation: Illegal procedure pointer. [2]
023	DST Violation: Illegal segment number in an external data segment pointer. [2]
024	Stack Overflow: The operating system normally handles this trap by extending the stack segment.
025	Stack Underflow: An attempt to pop a word from the local stack when the local stack is empty. [2]
026	Privileged Mode Violation: An attempt to execute a privileged instruction or return to a privileged procedure while in unprivileged mode. [2]
027	Privileged Mode Data Violation: An attempt to reference a privileged data

	segment while in unprivileged mode. [2]
030	Unexpected Pointer Type: An instruction has encountered a pointer type which it cannot handle. [2]
031	User Traps: Integer divide by zero. [1]
032	Illegal Decimal Number: A decimal math instruction has been supplied an illegal operand. [2]
033	Exponent Size Trap: Exponent too large during a number conversion instruction. [2]
034	Floating Point Operand Trap: Attempt to operate on illegal numbers, divide by zero, or convert a 64-bit number to a 32-bit number which cannot accommodate the exponent. [1]
035	Floating Point Result Trap: Floating point overflow; also caused by an explicit request to trap on an inexact result. [1]
036	Unexpected External Data Segment Type: A paged external data segment was encountered when an unpaged segment was expected, or vice versa. [2]
037	Absent Code Segment: Handled by the operating system.
040	Absent Page: Handled by the operating system.
041	Uncallable Procedure: Attempt to call an uncallable privileged procedure while in unprivileged mode. [2]
042	Absent Data Segment: Handled by the operating system.
043	Absent Page Table: Handled by the operating system.
044	Start-of-Line: Debugging trap. [1]
045	Variable Trace: Debugging trap. [1]
046	Start-of-Procedure: Debugging trap. [1]
047	End-of-Procedure: Debugging trap. [1]
050	Start-of-Subroutine: Debugging trap. [1]
051	End-of-Subroutine: Debugging trap. [1]
052	Code Segment Violation: Attempt to modify a code segment. [2]
053	Branch Violation: Illegal branch instruction. [2]
054	Message Trap: Used internally by the operating system.
055	Instruction Sequencing Bounds Violation: Program destination is out of bounds; probably a stack marker has been incorrectly modified.
056	Start-of-Line-Check Trap: Debugging trap. [1]
057	Data Segment Write Violation: Attempt to modify a write-protected data segment. [2]

The footnotes are as follows:

[1]: If the program returns from the trap (signal) handler, execution will resume with the next instruction.

[2]: If the program returns from the trap (signal) handler, execution will resume at the current instruction.

SEE ALSO

err(1), signal(2).

WARNING

Trapno is intended for diagnostic purposes only. Values and meanings may change in future releases of HP-UX.

NAME

ulimit – get and set user limits

SYNOPSIS

long ulimit (cmd, newlimit)
int cmd;
long newlimit;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

This function provides for control over process limits. The *cmd* values available are:

1 Get the process's file size limit. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.

2 Set the process's file size limit to the value of *newlimit*. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. *Ulimit* will fail and the limit will be unchanged if a process with an effective user ID other than super-user attempts to increase its file size limit. [EPERM]

3 Get the maximum possible break value. See *brk*(2).

HARDWARE DEPENDENCIES

Series 500/600/700:

The limit (for *cmd* = **1**) is in units of 1024-byte blocks.

RETURN VALUE

Upon successful completion, a non-negative value is returned. If *cmd* is not in the correct range, a –1 is returned, and *errno* is set to EINVAL. Other errors also return a –1, with *errno* set appropriately.

SEE ALSO

brk(2), write(2).

NAME

umask – get and set file creation mask

SYNOPSIS

int umask (cmask)
int cmask;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Umask sets the process's file mode creation mask to the octal number *cmask* and returns the previous value of the mask. Only the low-order 9 bits of *cmask* and the file mode creation mask are used.

The bits that are set in *cmask* specify which permission bits to turn off in the mode of the created file. For example, suppose a value of 007 is specified for *cmask*. Then, if a file is normally created with permissions of 777, its mode after creation would be 770.

RETURN VALUE

The previous value of the file mode creation mask is returned.

SEE ALSO

mkdir(1), sh(1), chmod(2), creat(2), mknod(2), open(2), mknod(8).

NAME

umount – unmount a file system

SYNOPSIS

int umount (spec)
char *spec;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Umount requests that a previously mounted file system contained on the block special device identified by *spec* be unmounted. *Spec* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

Umount may be invoked only by the super-user.

Umount will fail if one or more of the following are true:

The process's effective user ID is not super-user. [EPERM]

Spec does not exist. [ENOENT]

Spec is not a block special device. [ENOTBLK]

Spec is not mounted. [EINVAL]

A file on *spec* is busy. [EBUSY]

Spec points outside the process's allocated address space. [EFAULT]

The device associated with *spec* does not exist. [ENXIO]

A component of *spec* is not a directory. [ENOTDIR]

Spec is null. [ENOENT]

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

mount(1), mount(2).

BUGS

If *umount* is called from the program level (i.e. not from the *mount*(1) level), the table of mounted devices contained in /etc/mnttab is not updated.

NAME
uname – get name of current HP-UX system

SYNOPSIS
#include <sys/utsname.h>

int uname (name)
struct utsname *name;

HP-UX COMPATIBILITY
Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION
Uname stores information identifying the current HP-UX system in the structure pointed to by *name*.

Uname uses the structure defined in <**sys/utsname.h**>:

```
#define UTSLEN              9
#define SNLEN               15

struct utsname {
        char        sysname[UTSLEN];
        char        nodename[UTSLEN];
        char        release[UTSLEN];
        char        version[UTSLEN];
        char        machine[UTSLEN];
        char        idnumber[SNLEN];
};
extern struct utsname utsname;
```

Uname returns a null-terminated string in each field. *Sysname* contains "HP-UX". Similarly, *nodename* contains the name that the system is known by on a communications network and is accessible via *hostname*(1), *sethostname*(2), and *gethostname*(2). *Release* contains the release number of the operating system, e.g. "1.0" or "3.0.1". *Version* contains additional information about the operating system. *Machine* contains a standard name that identifies the hardware on which the HP-UX system is running. *Idnumber* contains an identification number which is unique within that class of hardware, possibly a hardware or software serial number. This field may return the null string to indicate the lack of an identification number.

Uname will fail if *name* points to an invalid address. [EFAULT]

HARDWARE DEPENDENCIES
Series 500/600/700:

The first character of the *version* field is set to "A" for single user, and "B" for multi-user.

RETURN VALUE
Upon successful completion, a non-negative value is returned. Otherwise, –1 is returned and *errno* is set to indicate the error.

SEE ALSO
hostname(1), uname(1), gethostname(2), sethostname(2).

NAME

unlink – remove directory entry; delete file

SYNOPSIS

int unlink (path)
char *path;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Unlink removes the directory entry named by the path name pointed to by *path*.

The named file is unlinked unless one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]

The named file does not exist. [ENOENT]

Search permission is denied for a component of the path prefix. [EACCES]

Write permission is denied on the directory containing the link to be removed. [EACCES]

The named file is a directory and the effective user ID of the process is not super-user. [EPERM]

The entry to be unlinked is the mount point for a mounted file system. [EBUSY]

The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed. [ETXTBSY]

The directory entry to be unlinked is part of a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

Path is null. [ENOENT]

A component of *path* does not exist. [ENOENT]

When all links to a file have been removed and no process has the file open, the space occupied by the file is freed and the file ceases to exist. If one or more processes have the file open when the last link is removed, the removal is postponed until all references to the file have been closed.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

rm(1), close(2), link(2), open(2).

BUGS

If the link count is 1, then a directory must be empty to be unlinked. If a directory is not empty, it may be unlinked if its link count is greater than 1.

NAME

ustat – get file system statistics

SYNOPSIS

#include <sys/types.h>
#include <ustat.h>

int ustat (dev, buf)
dev_t dev;
struct ustat *buf;

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

Ustat returns information about a mounted file system. *Dev* is a device number identifying a device containing a mounted file system. *Buf* is a pointer to a *ustat* structure (defined in **ustat.h**) that includes the following elements:

```
daddr_t f_tfree;      /* Total free blocks */
ino_t   f_tinode;     /* Number of free inodes */
char    f_fname[6];   /* Filsys name */
char    f_fpack[6];   /* Filsys pack name */
int     f_blksize;    /* Block size */
```

Ustat will fail if one or more of the following are true:

Dev is not the device number of a device containing a mounted file system. [EINVAL]

Buf points outside the process's allocated address space. [EFAULT]

HARDWARE DEPENDENCIES

Series 500/600/700:

In the above structure, f_fname[6] is the driver name, not the file system name.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

stat(2), fs(5).

NAME

utime – set file access and modification times

SYNOPSIS

#include <sys/types.h>
int utime (path, times)
char *path;
struct utimbuf *times;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Path points to a path name naming a file. *Utime* sets the access and modification times of the named file.

If *times* is **NULL**, the access and modification times of the file are set to the current time. A process must be the owner of the file or have write permission to use *utime* in this manner.

If *times* is not **NULL**, *times* is interpreted as a pointer to a *utimbuf* structure and the access and modification times are set to the values contained in the designated structure. Only the owner of the file or the super-user may use *utime* this way.

The times in the following structure are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

```
struct utimbuf {
   time_t actime;    /* access time */
   time_t modtime;  /* modification time */
};
```

Utime will fail if one or more of the following are true:

The named file does not exist. [ENOENT]

A component of the path prefix is not a directory. [ENOTDIR]

Search permission is denied by a component of the path prefix. [EACCES]

The effective user ID is not super-user and not the owner of the file and *times* is not **NULL**. [EPERM]

The effective user ID is not super-user and not the owner of the file and *times* is **NULL** and write access is denied. [EACCES]

The file system containing the file is mounted read-only. [EROFS]

Times is not **NULL** and points outside the process's allocated address space. [EFAULT]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

stat(2).

NAME

vsadv – advise system about backing store usage

SYNOPSIS

```
#include <sys/ems.h>

vsadv (index);
int       index;
```

HP-UX COMPATIBILITY

Level:	Backing Store Control - HP-UX/STANDARD
Origin:	HP
Remarks:	*Vsadv* is currently implemented on the Series 500/600/700 only.

DESCRIPTION

This call requests that all future backing store space allocated for this process be placed on the backing store device specified by *index* (see *vson*(2)). It may be used to tune an application to the local system environment. This request remains in effect until the next call to *vsadv* by this process. An *index* of -1 will set backing store allocation back to the default system policy.

This call is advisory in nature and will never cause subsequent program failures (e.g. if the device has no room, the system will simply override the request and use another device).

This characteristic is inherited across *fork*(2) and *exec*(2).

This call may be reduced to a no-op.

SEE ALSO

ems(2), vson(2).

NAME

vson, vsoff – advise OS about backing store devices

SYNOPSIS

#include <sys/ems.h>

```
int     vson(pathname, size, q);
int     size, q;
char    *pathname;

int     vsoff(index, force);
int     index, force;
```

HP-UX COMPATIBILITY

Level: Backing Store Control - HP-UX/STANDARD

Origin: HP

Remarks: *Vson* and *vsoff* are currently implemented on the Series 500/600/700 only.

DESCRIPTION

Vson is used to make the block special file *pathname* available for use by the system as a backing store device for whatever form of backing store is needed by the system. The call returns an id by which the backing store device may be referenced in subsequent *vsoff* or *vsadv*(2) calls. Multiple *vson* calls for the same device will return the same id (here "same device" means identical devno - major and minor - and not necessarily the same file name).

Pathname specifies a block special device file, which may or may not contain a mounted file system. If device does not contain a file system (i.e. an "empty" disc), *size* specifies the available backing storage space (in blocks) to be made available (the storage space is assumed to start at block 0 in this case). If *size* is set to –1 and the device does not contain a file system, the whole block special device will be used for backing store.

Q is a quality (i.e. performance) factor for the device. It is used by the system in load balancing decisions. Higher values suggest secondary choices for backing store devices. There is no inherent significance to the value of *q* other than its value relative to the *q* factor of the other devices in the list. This parameter may be ignored on some implementations.

Vsoff is used to remove a device from the list of backing store devices availible to the system. *Index* is the value returned by *vson* when the device was added to the list.

If *force* is not set (i.e. is 0) the system attempts to "gracefully" eliminate backing store usage of device by migrating backing store space onto other devices. If *force* is set (if, for instance, the device has failed) no attempt is made to salvage images stored on the disc. Processes with images on the device will, in all probability, be rather ungracefully terminated in the near future (i.e. when the images are required).

Only the super-user may add or remove backing store devices. A normal user may call *vson* to get the id for a device already known to the system as a backing store device (for subsequent use in a *vsadv*(2) call).

HARDWARE DEPENDENCIES

Series 500/600/700:

The device specified by *pathname* must be a CS-80 device.

RETURN VALUES

Upon successful completion, *vson* returns the index for the device and *vsoff* returns 0. If there is an error, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

ems(2), memallc(2), vsadv(2)

NAME

wait – wait for child process to terminate

SYNOPSIS

int wait (stat_loc)
int *stat_loc;

int wait ((int *)0)

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Wait suspends the calling process until it receives a signal that is to be caught (see *signal*(2)), or until any one of the calling process's child processes stops in a trace mode (see *ptrace*(2)) or terminates. If a child process stopped or terminated prior to the call on *wait*, return is immediate.

If *stat_loc* (taken as an integer) is non-zero, status information is stored in the location pointed to by *stat_loc* (only the 16 low bits are significant). *Status* can be used to differentiate between stopped and terminated child processes. If the child process is terminated, status identifies the cause of termination and passes useful information to the parent. This is accomplished in the following manner:

> If the child process stopped, the high order 8 bits of status will be zero and the low order 8 bits will be set equal to 177.
>
> If the child process terminated due to an *exit* or *_exit* call, the low order 8 bits of status will be zero and the high order 8 bits will contain the low order 8 bits of the argument that the child process passed to *exit*; see *exit*(2).
>
> If the child process terminated due to a signal, the high order 8 bits of status will be zero and the low order 8 bits will contain the number of the signal that caused the termination. In addition, if the low order seventh bit (i.e., bit 0200) is set, a "core image" will have been produced; see *signal*(2).

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process (*proc1*) inherits the child processes.

Wait will fail if one or more of the following are true:

> The calling process has no existing unwaited-for child processes. [ECHILD] In this case, *wait* returns immediately.
>
> *Stat_loc* points to an illegal address. [EFAULT]

HARDWARE DEPENDENCIES

Series 500/600/700:

> *Ptrace*(2) is not currently supported. Thus, trace modes and the "stopped" state are not currently implemented.

RETURN VALUE

If *wait* returns due to the receipt of a signal, a value of –1 is returned to the calling process and *errno* is set to EINTR. If *wait* returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of –1 is returned and *errno* is set to indicate the error.

SEE ALSO

exec(2), exit(2), fork(2), pause(2), signal(2).

WARNING

See *WARNING* in *signal*(2).

NAME

write – write on a file

SYNOPSIS

int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fildes is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call.

Write attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from *write*, the file pointer is incremented by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

Write will fail and the file pointer will remain unchanged if one or more of the following are true:

> *Fildes* is not a valid file descriptor open for writing. [EBADF]
>
> An attempt is made to write to a pipe that is not open for reading by any process. [EPIPE and SIGPIPE signal]
>
> An attempt was made to write a file that exceeds the process's file size limit or the maximum file size. See *ulimit*(2). [EFBIG]
>
> *Buf* points outside the process's allocated address space. [EFAULT]

If a *write* requests that more bytes be written than there is room for (e.g., the *ulimit* (see *ulimit*(2)) or the physical end of a medium), only as many bytes as there is room for will be written, and a value of -1 will be returned with *errno* set to EFBIG. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return –1, but 20 bytes will actually have been written. The next write of a non-zero number of bytes will give a failure return (–1), except as noted below.

If the file being written is a pipe (or FIFO), no partial writes will be permitted. Thus, the write will fail if a write of *nbyte* bytes would exceed a limit.

If the file being written is a pipe (or FIFO) and the O_NDELAY flag of the file flag word is set, then write to a full pipe (or FIFO) will return a count of 0. Otherwise (O_NDELAY clear), writes to a full pipe (or FIFO) will block until space becomes available.

HARDWARE DEPENDENCIES

Series 500/600/700:

> If you perform a write operation following an *lseek* into memory that has not yet been written on, all "empty" bytes up to your new position in memory are zeroed-out before writing your data.

RETURN VALUE

Upon successful completion the number of bytes actually written is returned. Otherwise, –1 is returned and *errno* is set to indicate the error.

SEE ALSO

creat(2), dup(2), lseek(2), open(2), pipe(2), ulimit(2), ustat(2).

BUGS

Writing to raw disc device files must be done in multiples of the device's physical record size (see *ustat*(2)), and must begin on a physical sector boundary.

NAME

intro – introduction to subroutines and libraries

SYNOPSIS

#include <stdio.h>

#include <math.h>

HP-UX COMPATIBILITY

Level: The level given is the level for which the library is available, not the level at which the linkable object code appears. The supporting host will contain appropriate libraries for HP-UX/RUN ONLY and HP-UX/NUCLEUS systems.

Origin: System III

DESCRIPTION

This section describes functions found in various libraries, other than those functions that directly invoke HP-UX system primitives, which are described in Section 2 of this volume. Certain major collections are identified by a letter after the section number:

(3C) These functions, together with those of Section 2 and those marked (3S), constitute library *libc*, which is automatically loaded by the C compiler, *cc*(1). The link editor *ld*(1) searches this library under the **–lc** option. Declarations for some of these functions may be obtained from **#include** files indicated on the appropriate pages.

(3M) These functions constitute the math library, *libm*. The link editor searches this library under the **–lm** option. Declarations for these functions may be obtained from the **#include** file **<math.h>**.

(3S) These functions constitute the "standard I/O package" (see *stdio*(3S)). These functions are in the library *libc*, already mentioned. Declarations for these functions may be obtained from the **#include** file **<stdio.h>**.

(3X) Various specialized libraries. The files in which these libraries are found are given on the appropriate pages.

The descriptions of some functions refer to **NULL**. This is the value that is obtained by casting **0** into a character pointer. The C language guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return it, for example, to indicate an error. **NULL** is defined in **<stdio.h>** as **0**; the user can include his own definition if he is not using **<stdio.h>**.

FILES

/lib/libc.a
/lib/libm.a

SEE ALSO

ar(1), cc(1), fc(1), ld(1), nm(1), ranlib(1), stdio(3S).

DIAGNOSTICS

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable *errno* (see *errno*(2)) is set to the value EDOM or ERANGE.

NAME

a64l, l64a – convert between long and base-64 ASCII

SYNOPSIS

long a64l (s)
char *s;

char *l64a (l)
long l;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

These routines are used to maintain numbers stored in *base-64* ASCII. This is a notation by which long integers can be represented by up to six characters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are **.** for 0, / for 1, **0** through **9** for 2–11, **A** through **Z** for 12–37, and **a** through **z** for 38–63.

The leftmost character is the least most significant digit. For example,

a0 = (38 x 64^0) + (2 x 64^1) = 166

A64l takes a pointer to a null-terminated base-64 representation and returns a corresponding **long** value. *L64a* takes a **long** argument and returns a pointer to the corresponding base-64 representation.

BUGS

The value returned by *l64a* is a pointer into a static buffer, the contents of which are overwritten by each call.

NAME

abort – generate an IOT fault

SYNOPSIS

abort()

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

Abort causes the SIGIOT signal to be sent to the process. This usually results in termination with a core dump.

It is possible for *abort* to return control if **SIGIOT** is caught or ignored. The result is identical to that of *kill*(2).

SEE ALSO

adb(1), exit(2), kill(2), signal(2).

DIAGNOSTICS

Usually "abort – core dumped" from the shell.

NAME
abs – integer absolute value

SYNOPSIS
int abs (i)
int i;

HP-UX COMPATIBILITY
Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION
Abs returns the absolute value of its integer operand.

HARDWARE DEPENDENCIES
Series 210:
Series 500/600/700:
The largest negative integer returns itself.

SEE ALSO
fabs(3M).

NAME

assert – program verification

SYNOPSIS

#include <assert.h>

assert (expression);

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

This macro is useful for putting diagnostics into programs. When it is executed, if *expression* is false, it prints "Assertion failed: file *xyz*, line *nnn*" on the standard error file and exits. *Xyz* is the source file and *nnn* the source line number of the *assert* statement. Compiling with the preprocessor option **–DNDEBUG** (see *cc*(1)) will cause *assert* to be ignored.

NAME

atof, atoi, atol – convert ASCII to numbers

SYNOPSIS

double atof (nptr)
char *nptr;

int atoi (nptr)
char *nptr;

long atol (nptr)
char *nptr;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

These functions convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional **e** or **E** followed by an optionally signed integer.

Atoi and *atol* recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

HARDWARE DEPENDENCIES

Series 500/600/700:
Atoi and *atol* are identical.

SEE ALSO

scanf(3S).

BUGS

There are no provisions for overflow.

NAME

j0, j1, jn, y0, y1, yn – bessel functions

SYNOPSIS

#include <math.h>

double j0 (x)
double x;

double j1 (x)
double x;

double jn (n, x);
int n;
double x;

double y0 (x)
double x;

double y1 (x)
double x;

double yn (n, x)
int n;
double x;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause *y0*, *y1*, and *yn* to return the value of HUGE (defined in *math.h*), and sets *errno* to EDOM.

NAME

_toupper, _tolower, toupper, tolower, toascii – character translation

SYNOPSIS

int toupper (c)
int c;

int tolower (c)
int c;

#include <ctype.h>

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Toupper and *tolower* have as domain the range of *getc*: the integers from –1 through 255. If the argument of *toupper* represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of *tolower* represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.

_toupper and *_tolower* are macros that accomplish the same thing as *toupper* and *tolower* but have restricted domains and are faster. *_toupper* requires a lower-case letter as its argument; its result is the corresponding upper-case letter. *_tolower* requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause garbage results. Use of this form will never work with foreign character sets.

Toascii yields its argument with all bits turned off that are not part of a standard 7 bit ASCII character; it is intended for compatibility with other systems.

SEE ALSO

ctype(3C).

NAME

crypt, setkey, encrypt – DES encryption

SYNOPSIS

char *crypt (key, salt)
char *key, *salt;

setkey (key)
char *key;

encrypt (block, edflag)
char *block;
int edflag;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard (DES), with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to *crypt* is a user's typed password. The second is a 2-character string chosen from the set [**a-zA-Z0-9.**/]; this *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The *setkey* and *encrypt* entries provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the *encrypt* entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

login(1), passwd(1), getpass(3C), passwd(5).

BUGS

The return value points to static data that are overwritten by each call.

NAME

ctermid – generate file name for terminal

SYNOPSIS

#include <stdio.h>

char *ctermid(s)
char *s;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Ctermid generates a string that refers to the controlling terminal for the current process when used as a file name.

If (int)*s* is zero, the string is stored in an internal static area, the contents of which are overwritten at the next call to *ctermid*, and the address of which is returned. If (int)*s* is non-zero, then *s* is assumed to point to a character array of at least **L_ctermid** elements; the string is placed in this array and the value of *s* is returned. The manifest constant **L_ctermid** is defined in <**stdio.h**>.

The difference between *ctermid* and *ttyname*(3C) is that *ttyname* must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while *ctermid* returns a magic string (/**dev**/**tty**) that will refer to the terminal if used as a file name. Thus *ttyname* is useless unless the process already has at least one file open to a terminal.

SEE ALSO

ttyname(3C).

NAME

ctime, daylight, localtime, gmtime, asctime, timezone, tzset, tzname – convert date and time to ASCII

SYNOPSIS

char *ctime (clock)
long *clock;

#include <time.h>

struct tm *localtime (clock)
long *clock;

struct tm *gmtime (clock)
long *clock;

char *asctime (tm)
struct tm *tm;

tzset ( )

extern long timezone;

extern daylight;

extern char *tzname[2];

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

Ctime converts a time pointed to by *clock* such as returned by *time*(2) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

Localtime and *gmtime* return pointers to structures containing the broken-down time. *Localtime* corrects for the time zone and possible daylight saving time; *gmtime* converts directly to GMT, which is the time the HP-UX system uses. *Asctime* converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
struct      tm{     /* see ctime(3C) */
            int     tm_sec;
            int     tm_min;
            int     tm_hour;
            int     tm_mday;
            int     tm_mon;
            int     tm_year;
            int     tm_wday;
            int     tm_yday;
            int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday = 0), year – 1900, day of year (0-365), and a flag that is non-zero if daylight saving time is in effect.

The external **long** variable *timezone* contains the difference, in seconds, between GMT and local standard time (in EST, *timezone* is 5*60*60); the external variable *daylight* is non-zero if and only if the standard U.S.A. Daylight Saving Time conversion should be applied. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

If an environment variable named **TZ** is present, *asctime* uses the contents of the variable to override the default time zone. The value of **TZ** must be a three-letter time zone name, followed by a number representing the difference between local time and Greenwich time in hours, followed by an optional three-letter name for a daylight time zone. For example, the setting for New Jersey would be **EST5EDT**. The effects of setting **TZ** are thus to change the values of the external variables *timezone* and *daylight*; in addition, the time zone names contained in the external variable

```
char *tzname[2] = {"EST", "EDT"};
```

are set from the environment variable. The function *tzset* sets the external variables from **TZ**; it is called by *asctime* and may also be called explicitly by the user.

SEE ALSO

time(2), getenv(3C), environ(7).

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii – character classification

SYNOPSIS

#include <ctype.h>

int isalpha (c)
int c;

. . .

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value **EOF** (see *stdio*(3S)). Note that *c* is specified in octal.

isalpha	*c* is a letter
isupper	*c* is an upper case letter
islower	*c* is a lower case letter
isdigit	*c* is a digit [0-9]
isxdigit	*c* is a hexidecimal digit [0-9], [A-F] or [a-f]
isalnum	*c* is an alphanumeric
isspace	*c* is a space, tab, carriage return, new-line, vertical tab, or form-feed
ispunct	*c* is a punctuation character (neither control nor alphanumeric)
isprint	*c* is a printing character, code 040 (space) through 0176 (tilde)
isgraph	*c* is a printing character, like *isprint* except false for space
iscntrl	*c* is a delete character (0177) or ordinary control character (less than 040).
isascii	*c* is an ASCII character, code less than 0200

SEE ALSO

ascii(7).

NAME

cuserid – character login name of the user

SYNOPSIS

#include <stdio.h>

char *cuserid (s)
char *s;

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Cuserid generates a character representation of the login name of the owner of the current process. If (int)*s* is zero, this representation is generated in an internal static area, the address of which is returned. If (int)*s* is non-zero, *s* is assumed to point to an array of at least **L_cuserid** characters; the representation is left in this array. The manifest constant **L_cuserid** is defined in **<stdio.h>**.

SEE ALSO

getlogin(3C), getpwuid(3C).

DIAGNOSTICS

If the login name cannot be found, *cuserid* returns **NULL**; if *s* is non-zero in this case, **\0** will be placed at **s*.

BUGS

Cuserid uses *getpwnam*(3C); thus the results of a user's call to the latter will be obliterated by a subsequent call to the former.

NAME

ecvt, fcvt, gcvt – output conversion

SYNOPSIS

char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt (value, ndigit, buf)
double value;
char *buf;
int ndigit;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Ecvt converts the *value* to a null-terminated string of *ndigit* ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to *ecvt*, except has been rounded for Fortran F-format output (fw.d), where *d* is equal to *ndigits*.

Gcvt converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf*. It attempts to produce *ndigit* significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3S).

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

end, etext, edata – last locations in program

SYNOPSIS

extern char end;
extern char etext;
extern char edata;

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region. Note that the definition of each of these is implementation-dependent. See *HARDWARE DEPENDENCIES* below.

When execution begins, the program break coincides with *end*, but the program break may be reset by the routines of *brk*(2), *malloc*(3C), standard input/output (*stdio*(3S)), the profile (**–p**) option of *cc*(1), and so on. Thus, the current value of the program break should be determined by "sbrk(0)" (see *brk*(2)).

These symbols are accessible from assembly language provided you prefix them with an underscore (i.e. *_end*, *_etext*, *_edata*).

HARDWARE DEPENDENCIES

Series 500/600/700:

End is the lowest heap address available to the user. *Etext* is the lowest available address in the D-data segment. *Edata* is the first available address in the I-data area.

Use *memallc*(2) instead of *malloc*(3C) to set the program break.

In C, these names must look like addresses. Thus, you would write **&end** instead of **end** to access the current value of *end*.

SEE ALSO

brk(2), malloc(3C).

NAME

exp, log, log10, pow, sqrt – exponential, logarithm, power, square root functions

SYNOPSIS

#include <math.h>

double exp (x)
double x;

double log (x)
double x;

double log10 (x)
double x;

double pow (x, y)
double x, y;

double sqrt (x)
double x;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Exp returns the exponential function of *x*.

Log returns the natural logarithm of *x*.

Log10 returns the common logarithm of *x*.

Pow returns x^y.

Sqrt returns the square root of *x*.

HARDWARE DEPENDENCIES

Series 210:
Series 500/600/700:
The algorithms used are those from HP 9000 BASIC.

SEE ALSO

hypot(3M), sinh(3M).

DIAGNOSTICS

Exp returns a huge value when the correct value would overflow. An argument out of range may also result in *errno* being set to **ERANGE**. Underflow returns a zero.

Log returns a huge negative value and sets *errno* to **EDOM** when *x* is non-positive.

Pow returns a huge negative value and sets *errno* to **EDOM** when *x* is non-positive and *y* is not an integer, or when *x* and *y* are both zero. It also returns a huge value when the correct value would overflow. A truly outrageous argument may also result in *errno* being set to **ERANGE**. Underflow causes *pow* to return zero and set *errno* to **ERANGE**.

Sqrt returns 0 and sets *errno* to **EDOM** when *x* is negative.

NAME

fclose, fflush – close or flush a stream

SYNOPSIS

#include <stdio.h>

int fclose (stream)
FILE *stream;

int fflush (stream)
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fclose causes any buffers for the named *stream* to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling *exit*(2).

Fflush causes any buffered data for the named output *stream* to be written to that file. The stream remains open.

These functions return 0 for success, and **EOF** if any errors were detected.

SEE ALSO

close(2), fopen(3S), setbuf(3S).

NAME

ferror, feof, clearerr, fileno – stream file status inquiries

SYNOPSIS

#include <stdio.h>

int feof (stream)
FILE *stream;

int ferror (stream)
FILE *stream

clearerr (stream)
FILE *stream

fileno(stream)
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Feof returns non-zero when end of file is read on the named input *stream*, otherwise zero.

Ferror returns non-zero when error has occurred reading or writing the named *stream*, otherwise zero. Unless cleared by *clearerr*, the error indication lasts until the stream is closed.

Clearerr resets the error indication on the named *stream*.

Fileno returns the integer file descriptor associated with the *stream*, see *open*(2).

Feof, *ferror*, and *fileno* are implemented as macros; they cannot be re-declared.

SEE ALSO

open(2), fopen(3S).

NAME

floor, fabs, ceil, fmod – absolute value, floor, ceiling, remainder functions

SYNOPSIS

#include <math.h>

double floor (x)
double x;

double ceil (x)
double x;

double fmod (x, y)
double x, y;

double fabs (x)
double x;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fabs returns $|x|$.

Floor returns the largest integer (as a double precision number) not greater than x.

Ceil returns the smallest integer not less than x.

Fmod returns the number f such that $x = iy + f$, for some integer i, and $0 \leq f < y$.

SEE ALSO

abs(3C).

NAME

fopen, freopen, fdopen – open or re-open a stream file; convert file to stream

SYNOPSIS

#include <stdio.h>

FILE *fopen (file-name, type)
char *file-name, *type;

FILE *freopen (file-name, type, stream)
char *file-name, *type;
FILE *stream;

FILE *fdopen (fildes, type)
int fildes;
char *type;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fopen opens the file named by *file-name* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r"	open for reading
"w"	create for writing
"a"	append; open for writing at end of file, or create for writing
"r+"	open for update (reading and writing)
"w+"	create for update
"a+"	append; open or create for update at end of file

Freopen substitutes the named file in place of the open *stream*. It returns the original value of *stream*. The original stream is closed, regardless of whether the open ultimately succeeds.

Freopen is typically used to attach the preopened constant names **stdin**, **stdout**, and **stderr** to specified files.

Fdopen associates a stream with a file descriptor obtained from *open*, *dup*, *creat*, or *pipe*(2). The *type* of the stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *fseek*, *rewind*, or an input operation which encounters end of file.

SEE ALSO

open(2), fclose(3S), popen(3S).

DIAGNOSTICS

Fopen and *freopen* return the pointer **NULL** if *file-name* cannot be accessed, if there are too many open files, or if the arguments are incorrect.

Fdopen returns a **NULL** if there are too many open files, or if the arguments are ill-formed.

NAME

fread, fwrite – buffered binary input/output to a stream file

SYNOPSIS

#include <stdio.h>

int fread ((char *) ptr, sizeof (*ptr), nitems, stream)
FILE *stream;

int fwrite ((char *) ptr, sizeof (*ptr), nitems, stream)
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fread reads, into a block beginning at *ptr*, *nitems* of data of the type of **ptr* from the named input *stream*. It returns the number of items actually read.

Fwrite appends at most *nitems* of data of the type of **ptr* beginning at *ptr* to the named output *stream*. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3S), getc(3S), gets(3S), printf(3S), putc(3S), puts(3S), scanf(3S).

NAME

frexp, ldexp, modf – split into mantissa and exponent

SYNOPSIS

double frexp (value, eptr)
double value;
int *eptr;

double ldexp (value, exp)
double value;

double modf (value, iptr)
double value, *iptr;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Frexp returns the mantissa of a double *value* as a double quantity, *x*, of magnitude less than 1, and stores an integer *n* indirecly through *eptr*, such that *value* = *x* * 2^*n*.

Ldexp returns the quantity *value* * 2^*exp*. If this quantity overflows, then a positive or negative huge value is returned, depending on the sign of *value*, and *errno* is set to ERANGE. If this quantity underflows then 0 is returned and *errno* is set to ERANGE.

Modf returns the signed fractional part of *value* and stores the integer part indirectly through *iptr*.

NAME

fseek, ftell, rewind – reposition a stream

SYNOPSIS

#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

long ftell (stream)
FILE *stream;

rewind(stream)
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

Fseek undoes any effects of *ungetc*(3S).

After *fseek* or *rewind*, the next operation on an update file may be either input or output.

Ftell returns the current value of the offset relative to the beginning of the file associated with the named *stream*. The offset is measured in bytes on HP-UX; on some other systems, the unit of measure varies, and is the only foolproof way to obtain an *offset* for *fseek*.

Rewind(*stream*) is equivalent to *fseek*(*stream*, 0L, 0).

SEE ALSO

lseek(2), fopen(3S).

DIAGNOSTICS

Fseek returns -1 for improper seeks, otherwise zero.

Ftell returns -1 for error conditions.

NAME

gamma, signgam – log gamma function

SYNOPSIS

#include <math.h>
extern int signgam;

double gamma (x)
double x;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Gamma returns $\ln |\Gamma(|x|)|$. The sign of $\Gamma(|x|)$ is returned in the external integer *signgam*. The following C program fragment might be used to calculate Γ:

```
y = gamma (x);
if (y > logmax)
        error ( );
y = exp (y) * signgam;
/* logmax is floor(ln(maxreal)) */
```

DIAGNOSTICS

For negative integer arguments, a huge value is returned, and *errno* is set to **EDOM**.

NAME

getc, getchar, fgetc, getw – get character or word from stream file

SYNOPSIS

#include <stdio.h>

int getc (stream)
FILE *stream;

int getchar ( )

int fgetc (stream)
FILE *stream;

int getw (stream)
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Getc returns the next character from the named input *stream*.

Getchar( ) is identical to *getc*(*stdin*).

Fgetc behaves like *getc*, but is a genuine function, not a macro; it may therefore be used as an argument. *Fgetc* runs more slowly than *getc*, but takes less space per invocation.

Getw returns the next word (**int**) from the named input *stream*. It returns the constant **EOF** upon end of file or error, but since that is a valid integer value, *feof* and *ferror*(3S) should be used to check the success of *getw*. *Getw* assumes no special alignment in the file.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), gets(3S), putc(3S), scanf(3S).

DIAGNOSTICS

These functions return the integer constant **EOF** at end of file or upon read error.

BUGS

Getc and its variant *getchar* return **EOF** on end of file.
Because it is implemented as a macro, *getc* treats incorrectly a *stream* argument with side effects. In particular, **getc(*f++);** doesn't work sensibly.

NAME

getenv – value for environment name

SYNOPSIS

char *getenv (name)
char *name;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Getenv searches the environment list (see *environ*(7)) for a string of the form *name = value* and returns *value* if such a string is present, otherwise 0 (**NULL**). *Name* may be either the desired name, null-terminated, or of the form *name = value,* in which case *getenv* uses the portion to the left of the " = " as the search key.

SEE ALSO

environ(7).

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent – get group file entry

SYNOPSIS

#include <grp.h>

struct group *getgrent ( );

struct group *getgrgid (gid)
int gid;

struct group *getgrnam (name)
char *name;

int setgrent ( );

int endgrent ( );

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Getgrent, *getgrgid* and *getgrnam* each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
struct group { /* see getgrent(3C) */
   char  *gr_name;
   char  *gr_passwd;
   int    gr_gid;
   char  *gr_mem;
};
```

The members of this structure are:

gr_name	The name of the group.
gr_passwd	The encrypted password of the group.
gr_gid	The numerical group ID.
gr_mem	Null-terminated vector of pointers to the individual member names.

Getgrent reads the next line of the file, so successive calls may be used to search the entire file. *Getgrgid* and *getgrnam* search from the beginning of the file until a matching *gid* or *name* is found, or **EOF** is encountered.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete, although that is usually not necessary.

FILES

/etc/group

SEE ALSO

getlogin(3C), getpwent(3C), group(5).

DIAGNOSTICS

A **NULL** pointer (0) is returned on **EOF** or error. Nothing is returned by *endgrent* or *setgrent*.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getlogin – get login name

SYNOPSIS

char *getlogin ( );

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Getlogin returns a pointer to the login name as found in /**etc**/**utmp**. It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same user ID is shared by several login names.

If *getlogin* is called within a process that is not attached to a terminal, it returns **NULL**. The correct procedure for determining the login name is to call *cuserid*, or to call *getlogin* and if it fails, to call *getpwuid*.

FILES

/etc/utmp

SEE ALSO

cuserid(3S), getgrent(3C), getpwent(3C), utmp(5).

DIAGNOSTICS

Getlogin returns **NULL** if the name is not found.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

getopt, optarg, optind, opterr – get option letter from argv

SYNOPSIS

int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;
extern char *optarg;
extern int optind;
extern int opterr;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Getopt returns the next option letter in *argv* (starting from *argv*[1]) that matches a letter in *optstring*. *Optstring* is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. *Optarg* is set to point to the start of the option argument on return from *getopt*.

Getopt places in *optind* the *argv* index of the next argument to be processed. *Optind* is initialized to one automatically before the first call to *getopt*.

When all options have been processed (i.e., up to the first non-option argument), *getopt* returns **EOF**. The special option –– may be used to delimit the end of the options; **EOF** will be returned, and –– will be skipped.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the options **f** and **o**, both of which require arguments:

```
main (argc, argv)
int argc;
char **argv;
{
        int c;
        extern int optind;
        extern char *optarg;
        :
        :
        while ((c = getopt (argc, argv, "abf:o:")) != EOF)
                switch (c) {
                case 'a':
                        if (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;
                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bproc();
                        break;
                case 'f':
                        ifile = optarg;
                        break;
```

```
                    case 'o':
                            ofile = optarg;
                            bufsiza = 512;
                            break;
                    case '?':
                            errflg++;
                    }
            if (errflg) {
                    fprintf (stderr, "usage: . . . ");
                    exit (2);
            }
            .
            .
            .
    }
```

DIAGNOSTICS

Getopt prints an error message on *stderr* and returns a question mark (**?**) when it encounters an option letter not included in *optstring*. The error message may be suppressed by setting *opterr* to zero.

BUGS

Options can be any ASCII characters except colon (:), question mark (?), or null (\0). It is impossible to distinguish between a ? used as a legal option, and the character that *getopt* returns when it encounters an invalid option character in the input.

NAME

getpass – read a password

SYNOPSIS

char *getpass (prompt)
char *prompt;

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Getpass reads a password from the file **/dev/tty**, or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES

/dev/tty

SEE ALSO

crypt(3C).

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

getpw – get name from UID

SYNOPSIS

getpw (uid, buf)
int uid;
char *buf;

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

This routine is included only for compatibility with prior systems and should not be used; see *getpwent*(3C) for routines to use instead.

FILES

/etc/passwd

SEE ALSO

getpwent(3C), passwd(5).

DIAGNOSTICS

Getpw returns non-zero on error.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent – get password file entry

SYNOPSIS

#include <pwd.h>

struct passwd *getpwent ( );

struct passwd *getpwuid (uid)
int uid;

struct passwd *getpwnam (name)
char *name;

int setpwent ( );

int endpwent ( );

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Getpwent, *getpwuid* and *getpwnam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct passwd {
        char                *pw_name;
        char                *pw_passwd;
        unsigned int        pw_uid;
        unsigned int        pw_gid;
        char                *pw_age;
        char                *pw_comment;
        char                *pw_gecos;
        char                *pw_dir;
        char                *pw_shell;
};
```

The *pw_comment* field is unused; the others have meanings described in *passwd*(5).

Getpwent reads the next line in the file, so successive calls can be used to search the entire file. *Getpwuid* and *getpwnam* search from the beginning of the file until a matching *uid* or *name* is found, or **EOF** is encountered.

A call to *setpwent* has the effect of rewinding the password file to allow repeated searches. *Endpwent* may be called to close the password file when processing is complete.

FILES

/etc/passwd

SEE ALSO

getlogin(3C), getgrent(3C), passwd(5).

DIAGNOSTICS

Null pointer (0) returned on **EOF** or error. Nothing is returned by *endpwent* or *setpwent*.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

gets, fgets – get a string from a stream file

SYNOPSIS

#include <stdio.h>

char *gets (s)
char *s;

char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Gets reads a string into *s* from the standard input stream **stdin**. The string is terminated by a new-line character, which is replaced in *s* by a null character. *Gets* returns its argument.

Fgets reads *n*–1 characters, or up to a new-line character (which is retained), whichever comes first, from the *stream* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its first argument.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), getc(3S), puts(3S), scanf(3S).

DIAGNOSTICS

Gets and *fgets* return the constant pointer **NULL** upon end-of-file or error.

NAME

hypot – Euclidean distance

SYNOPSIS

#include <math.h>

double hypot (x, y)
double x, y;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Hypot returns

sqrt(x*x + y*y),

taking precautions against unwarranted overflows.

SEE ALSO

sqrt(3M).

NAME

malloc, free, realloc, calloc – main memory allocator

SYNOPSIS

char *malloc (size)
unsigned size;

free (ptr)
char *ptr;

char *realloc (ptr, size)
char *ptr;
unsigned size;

char *calloc (nelem, elsize)
unsigned elem, elsize;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Malloc and *free* provide a simple general-purpose memory allocation package. *Malloc* returns a pointer to a block of at least *size* bytes beginning on a word boundary.

The argument to *free* is a pointer to a block previously allocated by *malloc*; this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, serious disorder will result if the space assigned by *malloc* is overrun or if some random number is handed to *free*.

Malloc allocates the first sufficiently large contiguous reach of free space found in a circular search from the last block allocated or freed, adding up adjacent free blocks as it searches. It calls *sbrk* (see *brk*(2)) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

Realloc also works if *ptr* points to a block freed since the last call of *malloc*, *realloc*, or *calloc*; thus sequences of *free*, *malloc* and *realloc* can exploit the search strategy of *malloc* to do storage compaction.

Calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS

Malloc, *realloc* and *calloc* return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. When *realloc* returns 0, the block pointed to by *ptr* may be destroyed.

BUGS

Free does not check its pointer argument for validity. When passed a null pointer (value 0), it causes a memory fault.

NAME

mktemp – make a unique file name

SYNOPSIS

char *mktemp (template)
char *template;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Mktemp replaces *template* by a unique file name, and returns the address of the template. The template should look like a file or path name with six trailing **X**s, which will be replaced with a letter and the current process ID. The letter will be chosen so that the resulting name does not duplicate the name of an existing file. If there are less than 6 **X**s, the letter will be dropped first, and then high order digits of the process ID will be dropped.

RETURN VALUE

Mktemp returns its argument except when it runs out of letters, in which case the result is a pointer to the string "/".

SEE ALSO

getpid(2).

BUGS

It is possible to run out of letters.

Mktemp does not check to see if the file name part of *template* exceeds the system limit of 14 characters.

NAME

perror, sys_errlist, sys_nerr – system error messages

SYNOPSIS

perror (s)
char *s;

int sys_nerr;
char *sys_errlist[ ];

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Perror produces a short error message on the standard error, describing the last error encountered during a system call from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. To be of most use, the argument string should be the name of the program that incurred the error. The error number is taken from the external variable *errno*, which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the new-line. *Sys_nerr* is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

HARDWARE DEPENDENCIES

Series 500/600/700:
The error indicator *errinfo* is implemented in addition to *errno*, enabling you to obtain a more detailed description of the error. See *errinfo*(2).

SEE ALSO

errinfo(2), errno(2).

NAME

popen, pclose – initiate pipe I/O to/from a process

SYNOPSIS

#include <stdio.h>

FILE *popen (command, type)
char *command, *type;

int pclose (stream)
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

The arguments to *popen* are pointers to null-terminated strings containing, respectively, a shell command line and an I/O mode, either **r** for reading or **w** for writing. *Popen* creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by *popen* should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type **r** command may be used as an input filter, and a type **w** as an output filter.

SEE ALSO

pipe(2), wait(2), fclose(3S), fopen(3S), system(3S).

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or if the shell cannot be accessed.

Pclose returns –1 if *stream* is not associated with a "*popen*ed" command.

BUGS

Only one stream opened by *popen* can be in use at once.

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, e.g. with *fflush*; see *fclose*(3S).

NAME

printf, fprintf, sprintf – output formatters

SYNOPSIS

#include <stdio.h>

int printf (format [**,** arg] . . . **)**
char *format;

int fprintf (stream, format [**,** arg] . . . **)**
FILE *stream;
char *format;

int sprintf (s, format [**,** arg] . . . **)**
char *s, *format;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Printf places output on the standard output stream *stdout*. *Fprintf* places output on the named output *stream*. *Sprintf* places "output", followed by the null character (**0**) in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the **0** in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more *arg*s. The results are undefined if there are insufficient *args* for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag (see below) has been given) to the field width;

A *precision* that gives the minimum number of digits to appear for the **d**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e** and **f** conversions, the maximum number of significant digits for the **g** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (**.**) followed by a decimal digit string: a null digit string is treated as zero.

An optional **l** specifying that a following **d**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*, or an optional **h** specifying that a following **d**, **o**, **u**, **x**, or **X** conversion character applies to a short integer *arg*.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted.

The flag characters and their meanings are:

–	The result of the conversion will be left-justified within the field.
+	The result of a signed conversion will always begin with a sign (+ or –).
blank	If the first character of a signed conversion is not a sign, a blank will be prepended to the

result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

#	This flag specifies that the value is to be converted to an "alternate form." For **c**, **d**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** (**X**) conversion, a non-zero result will have **0x** (**0X**) prepended to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,o,u,x,X	The integer *arg* is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of zero is a null string (unless the conversion is **o**, **x**, or **X** *and* the # flag is present).
f	The float or double *arg* is converted to decimal notation in the style "[–]ddd.ddd", where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly 0, no decimal point appears.
e,E	The float or double *arg* is converted in the style "[–]d.ddd**e**±ddd", where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains exactly three digits.
g,G	The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** will be used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.
c	The character *arg* is printed.
s	The *arg* is taken to be a string (character pointer) and characters from the string are printed until a null character (**0**) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed.
%	Print a **%**; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by *printf* and *fprintf* are printed as if *putchar* had been called (see *putc*(3S)).

EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02", where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %d, %.2d:%.2d", weekday, month, day, hour, min);
```

To print π to 5 decimal places:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO

ecvt(3C), putc(3S), scanf(3S), stdio(3S).

NAME

putc, putchar, fputc, putw – put character or word on a stream

SYNOPSIS

#include <stdio.h>

int putc (c, stream)
char c;
FILE *stream;

putchar (c)

fputc (c, stream)
FILE *stream;

putw (w, stream)
int w;
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Putc appends the character *c* to the named output *stream*. It returns the character written.

Putchar(c) is defined as *putc(c, stdout)*.

Fputc behaves like *putc*, but is a genuine function rather than a macro; it may therefore be used as an argument. *Fputc* runs more slowly than *putc*, but takes less space per invocation.

Putw appends the word (i.e., **int**) *w* to the output *stream*. *Putw* neither assumes nor causes special alignment in the file.

The standard stream *stdout* is normally buffered. However, if the output refers to a terminal then the stream is line-buffered. When a stream is line buffered, its buffer will be flushed when a read(2) from the standard input is necessary, when a "\n" is written to stdout, when the buffer is full, or when fflush(stdout) is called. This default may be changed by *setbuf*(3S).

The standard stream *stderr* is by default unbuffered unconditionally, but use of *freopen*(3S) will cause it to become unbuffered; *setbuf*, again, will set the state to whatever is desired. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. See also *fflush*(3S).

SEE ALSO

ferror(3S), fopen(3S), fwrite(3S), getc(3S), printf(3S), puts(3S).

DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, *ferror*(3S) should be used to detect *putw* errors.

Programs which use standard I/O routines but use *read*(2) to read from the standard input may become confused or malfunction when line buffering is used. In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to *fflush*(3) the standard output before doing any computation so that the output will appear.

BUGS

Because it is implemented as a macro, *putc* treats incorrectly a *stream* argument with side effects. In particular, **putc(c, *f++);** doesn't work sensibly.

NAME

putpwent – write password file entry

SYNOPSIS

#include <pwd.h>
#include <stdio.h>

int putpwent (p, f)
struct passwd *p;
FILE *f;

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Putpwent is the inverse of *getpwent*(3C). Given a pointer to a *passwd* structure as created by *getpwent* (or *getpwuid*(3C) or *getpwnam*(3C)), *putpwent* writes a line on the stream *f* which matches the format of /**etc**/**passwd**.

DIAGNOSTICS

Putpwent returns non-zero if an error was detected during its operation, otherwise zero.

NAME

puts, fputs – put a string on a stream file

SYNOPSIS

#include <stdio.h>

int puts (s)
char *s;

int fputs (s, stream)
char *s;
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Puts copies the null-terminated string *s* to the standard output stream *stdout* and appends a new-line character.

Fputs copies the null-terminated string *s* to the named output *stream*.

Neither routine copies the terminating null character. Note that *puts* appends a new-line character, but *fputs* does not.

SEE ALSO

ferror(3S), fopen(3S), fwrite(3S), gets(3S), printf(3S), putc(3S).

DIAGNOSTICS

Both routines return **EOF** on error.

NAME

qsort – quicker sort

SYNOPSIS

qsort (base, nel, width, compar)
char *base;
int nel, width;
int (*compar)( );

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. It sorts vectors of arbitrarily-sized elements based on user-supplied size information and a comparison routine.

The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second (this is the same return convention that *strcmp* uses).

SEE ALSO

sort(1), bsearch(3C), lsearch(3C), strcmp(3C).

BUGS

If *width* is zero, a divide-by-zero error is generated.

NAME

rand, srand – random number generator

SYNOPSIS

srand (seed)
unsigned seed;

rand ( )

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Rand uses a multiplicative congruential random number generator with period 2^32 to return successive pseudo-random numbers.

The generator is reinitialized by calling *srand* with 1 as argument. It can be set to a random starting point by calling *srand* with whatever number you like as the argument.

HARDWARE DEPENDENCIES

Series 210/500/600/700:

Rand returns successive pseudo-random numbers in the range from 0 to 2^15 – 1.

NAME

scanf, fscanf, sscanf – formatted input conversion, read from stream file

SYNOPSIS

#include <stdio.h>

scanf (format [, pointer] . . .)
char *format;

fscanf (stream, format [, pointer] . . .)
FILE *stream;
char *format;

sscanf (s, format [, pointer] . . .)
char *s, *format;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Scanf reads from the standard input stream *stdin*. *Fscanf* reads from the named input *stream*. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs, or new-lines, which cause input to be read up to the next non-white-space character.
2. An ordinary character (not **%**), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character **%**, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

%	a single **%** is expected in the input at this point; no assignment is done.
d	a decimal integer is expected; the corresponding argument should be an integer pointer.
o	an octal integer is expected; the corresponding argument should be an integer pointer.
x	a hexadecimal integer is expected; the corresponding argument should be an integer pointer.
s	a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a space character or a new-line. Note that *scanf* will not read a null string.
c	a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, use **%1s**. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.
e,f	a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an **E** or an **e**, followed by an optionally signed integer.
[	indicates a string that is not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of

characters making up the string. If the first character is not a circumflex (^), the input field consists of all characters up to the first character that is not in the set between the brackets; if the first character after the left bracket is a ^, the input field consists of all characters up to the first character that is in the set of the remaining characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o**, and **x** may be capitalized and/or preceded by **l** to indicate that a pointer to **long** rather than to **int** is in the argument list. Similarly, the conversion characters **e** and **f** may be capitalized and/or preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list. The character **h**, similarly, indicates **short** data items.

Scanf conversion terminates at **EOF**, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

Scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, **EOF** is returned.

Trailing white space (including a new-line) is left unread unless matched in the *format.*

EXAMPLES

The call:

```
int i; float x; char name[50];
scanf ("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E−1 thompson
```

will assign to *i* the value **25**, to *x* the value **5.432**, and *name* will contain **thompson\0**. Or:

```
int i; float x; char name[50];
scanf ("%2d%f%*d %[1234567890]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

will assign **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name*. The next call to *getchar* (see *getc*(3S)) will return **a**.

SEE ALSO

atof(3C), getc(3S), printf(3S).

DIAGNOSTICS

These functions return **EOF** on end of input and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

setbuf – assign buffering to a stream file

SYNOPSIS

#include <stdio.h>

setbuf (stream, buf)
FILE *stream;
char *buf;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array *buf* to be used instead of an automatically allocated buffer. If *buf* is the constant pointer **NULL**, input/output will be completely unbuffered.

A manifest constant **BUFSIZ** tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from *malloc*(3C) upon the first *getc* or *putc*(3S) on the file, except that output streams directed to terminals are normally line buffered, and the standard error stream *stderr* is normally not buffered.

A common source of error is allocation of buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

HARDWARE DEPENDENCIES

Series 500/600/700:
The system call *memallc*(2) is used instead of *malloc*.

SEE ALSO

fopen(3S), getc(3S), malloc(3C), putc(3S).

NAME

setjmp, longjmp – non-local goto

SYNOPSIS

#include <setjmp.h>

int setjmp (env)
jmp_buf env;

longjmp (env, val)
jmp_buf env;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in *env* for later use by *longjmp*. It returns value 0.

Longjmp restores the environment saved by the last call of *setjmp*. It then returns in such a way that execution continues as if the call of *setjmp* had just returned the value *val* to the corresponding call to *setjmp*, which must not itself have returned in the interim. *Longjmp* cannot return the value 0. If *longjmp* is invoked with a second argument of 0, it will return 1. All accessible data have values as of the time *longjmp* was called.

Upon the return from a *setjmp* call caused by a *longjmp*, the values of any register variables are undefined. Depending on such values renders code using register variables non-portable.

If a *longjmp* is executed and the environment in which the *setjmp* was executed no longer exists, errors can occur. The conditions under which the environment of the *setjmp* no longer exists include: exiting the procedure which contains the *setjmp* call, and exiting an inner block with temporary storage (e.g. a block with declarations in C, a *with* statement in Pascal). This condition may or may not be detectable. An attempt is made by determining if the stack frame pointer in *env* points to a location not in the currently active stack. If this is the case, *longjmp* will return a -1. Otherwise, the *longjmp* will occur, and if the environment no longer exists, the contents of the temporary storage of an inner block are unpredictable. This condition may also cause unexpected process termination. If the procedure has been exited the results are unpredictable.

Passing *longjmp* a pointer to a buffer not created by *setjmp*, or a buffer that has been modified by the user, can cause all the problems listed above, and more.

The Pascal language may support a *try/recover* mechanism, which also creates stack marker information. If a *longjmp* operation occurs in a scope which is nested inside a try/recover, and the corresponding *setjmp* is not inside the scope of the try/recover, the recover block will not be executed and the currently active recover block will become the one enclosing the *setjmp* (if there is one).

HARDWARE DEPENDENCIES

Series 500/600/700:
Register variables remain defined upon returning from a *setjmp* call caused by a *longjmp*.

SEE ALSO

signal(2).

NAME
sinh, cosh, tanh – hyperbolic functions

SYNOPSIS
#include <math.h>

double sinh (x)
double x;

double cosh (x)
double x;

double tanh (x)
double x;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION
These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS
Sinh and *cosh* return a huge value of appropriate sign when the correct value would overflow.

NAME

sleep – suspend execution for interval

SYNOPSIS

unsigned long sleep (seconds)
unsigned long seconds;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

The current process is suspended from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals, and (2) because any caught signal will terminate the *sleep* following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by *sleep* will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested *sleep* time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling *sleep*. If the *sleep* time exceeds the time till such an alarm signal, the process sleeps only until the alarm signal would have occurred, and the caller's alarm catching routine is executed just before the *sleep* routine returns. If the *sleep* time is less than the time till such an alarm, the prior alarm time is reset to go off at the same time it would have without the intervening *sleep*.

Seconds must be less than 2^32.

SEE ALSO

alarm(2), pause(2), signal(2).

NAME

stdio – standard buffered input/output stream file package

SYNOPSIS

#include <stdio.h>

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

The functions described in the entries of sub-class 3S of this manual constitute an efficient, user-level I/O buffering scheme. The in-line macros *getc*(3S) and *putc*(3S) handle characters quickly. The macros *getchar*, *putchar*, and the higher-level routines *fgetc*, *fgets*, *fprintf*, *fputc*, *fputs*, *fread*, *fscanf*, *fwrite*, *gets*, *getw*, *printf*, *puts*, *putw*, and *scanf* all use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type **FILE**. *Fopen*(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are 3 open streams with constant pointers declared in the "include" file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file.

A constant "pointer" **NULL** (0) designates the null stream.

An integer constant **EOF** (–1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

Any program that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in the entries of sub-class 3S of this manual are declared in that "include" file and need no further declaration. The constants and the following "functions" are implemented as macros (redeclaration of these names is perilous): *getc*, *getchar*, *putc*, *putchar*, *feof*, *ferror*, and *fileno*.

SEE ALSO

close(2), open(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), ferror(3S), fopen(3S), fread(3S), fseek(3S), getc(3S), gets(3S), popen(3S), printf(3S), putc(3S), puts(3S), scanf(3S), setbuf(3S), system(3S), tmpnam(3S).

DIAGNOSTICS

Invalid *stream* pointers will cause serious disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok – character string operations

SYNOPSIS

char *strcat (s1, s2)
char *s1, *s2;

char *strncat (s1, s2, n)
char *s1, *s2;
int n;

int strcmp (s1, s2)
char *s1, *s2;

int strncmp (s1, s2, n)
char *s1, *s2;
int n;

char *strcpy (s1, s2)
char *s1, *s2;

char *strncpy (s1, s2, n)
char *s1, *s2;
int n;

int strlen (s)
char *s;

char *strchr (s, c)
char *s, c;

char *strrchr (s, c)
char *s, c;

char *strpbrk (s1, s2)
char *s1, *s2;

int strspn (s1, s2)
char *s1, *s2;

int strcspn (s1, s2)
char *s1, *s2;

char *strtok (s1, s2)
char *s1, *s2;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strncat* copies at most *n* characters. It copies less if *s2* is shorter than *n* characters. Both return a pointer to the null-terminated result (the original value of *s1*).

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as *s1* is lexicographically greater than, equal to, or less than *s2*. (Nil pointers for *s1* and *s2* are treated the same as pointers to null strings.) *Strncmp* makes the same comparison but looks at at most *n* characters (*n* less than or equal to zero implies equality).

Strcpy copies string *s2* to *s1*, stopping after the null character has been moved. *Strncpy* copies exactly *n* characters, truncating or null-padding *s2*; the target may not be null-terminated if the length of *s2* is *n* or more. Both return *s1*. Note that *strncpy* should not be used to copy *n* bytes of an arbitrary structure. If that structure contains a null byte anywhere, *strncpy* will terminate the copy when it encounters the null byte, thus returning less than *n* bytes.

Strlen returns the number of non-null characters in *s*. (The "length" of a string does not count the null terminator.)

Strchr (*strrchr*) returns a pointer to the first (last) occurrence of character *c* (an 8-bit ASCII value) in string *s*, or **NULL** if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

Strpbrk returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or **NULL** if no character from *s2* exists in *s1*.

Strspn (*strcspn*) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

Strtok considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a **NULL** character into *s1* immediately following the returned token. Subsequent calls with zero for the first argument, will work through the string *s1* in this way until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a **NULL** is returned.

HARDWARE DEPENDENCIES

Series 500/600/700:

n is limited to about 500 Mbytes.

Strcmp and *strncmp* use signed characters for comparison.

BUGS

All string movement is performed character by character starting at the left. Thus overlapping moves toward the left will work as expected, but overlapping moves to the right (i.e. higher addresses) may yield surprises.

The copy operations cannot check for overflow of any receiving string. **NULL** destinations cause errors; **NULL** sources are treated as zero-length strings.

NAME

swab – swap bytes

SYNOPSIS

swab (from, to, nbytes)
char *from, *to;
int nbytes;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Swab copies *nbytes* bytes pointed to by *from* to the position pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between byte-swapped and non-byte-swapped machines. *Nbytes* should be even.

NAME

system – issue a shell command

SYNOPSIS

#include <stdio.h>

int system (string)
char *string;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

System causes the *string* to be given to *sh*(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO

sh(1), exec(2).

DIAGNOSTICS

System stops if it can't execute *sh*(1).

NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – access terminal capabilities in termcap(5)

SYNOPSIS

char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	UCB

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base *termcap*(5). These are low level routines.

Tgetent extracts the entry for terminal *name* into the buffer at *bp*. *Bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to *tgetnum*, *tgetflag*, and *tgetstr*. *Tgetent* returns –1 if it cannot open the *termcap* file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type *name* is the same as the environment string TERM, the TERMCAP string is used instead of reading the *termcap* file. If it does begin with a slash, the string is used as a path name rather than */etc/termcap*. This can speed up entry into programs that call *tgetent*, as well as to help debug new terminal descriptions or to make one for your terminal if you can't write the file */etc/termcap*.

Tgetnum gets the numeric value of capability *id*, returning –1 if it is not given for the terminal. *Tgetnum* is useful only with capabilities having numeric values.

Tgetflag returns 1 if the specified capability is present in the terminal's entry, and 0 if it is not. *Tgetflag* is useful only with capabilities that are boolean in nature (i.e. either present or missing in *termcap*(5)).

Tgetstr gets the string value of capability *id*, placing it in the buffer at *area*, advancing the *area* pointer. It decodes the abbreviations for this field described in *termcap*(5), except for cursor addressing and padding information. *Tgetstr* is useful only with capabilities having string values.

Tgoto returns a cursor addressing string decoded from *cm* (see *termcap*(5)) to go to column *destcol* in line *destline*. It uses the external variables **UP** (from the **up** capability) and **BC** (if **bc** is given rather than **bs**) if necessary to avoid placing **\n**, **^D** or **^@** in the returned string. (Programs which call *tgoto* should be sure to turn off the XTABS bit(s), since *tgoto* may now output a tab. See *tty*(4). Note that programs using *termcap* should in general turn off XTABS anyway since some terminals use control–I for other functions, such as nondestructive space.) If a **%** sequence is given which is not understood, then *tgoto* returns "OOPS".

Tputs decodes the leading padding information of the string *cp*. *Affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable. *Outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the output speed of the terminal as encoded by *stty*(2). The external variable **PC** should contain a pad character to be used (from the **pc** capability) if a null (^@) is inappropriate.

FILES

/usr/lib/libtermcap.a	–ltermcap library
/etc/termcap	data base

SEE ALSO

ex(1), tty(4), termcap(5).

NAME

tmpfile – create a temporary file

SYNOPSIS

#include <stdio.h>

FILE *tmpfile ( )

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Tmpfile creates a temporary file and returns a corresponding **FILE** pointer. Arrangements are made so that the file will automatically be deleted when the process using it terminates. The file is opened for update.

Tmpfile uses a name generated by *tmpnam*.

SEE ALSO

creat(2), unlink(2), fopen(3S), mktemp(3C), tmpnam(3S).

NAME

tmpnam – create a name for a temporary file

SYNOPSIS

#include <stdio.h>

char *tmpnam (s)
char *s;

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Tmpnam generates a file name that can safely be used for a temporary file. If (int)*s* is zero, *tmpnam* leaves its result in an internal static area and returns a pointer to that area. The next call to *tmpnam* will destroy the contents of the area. If (int)*s* is nonzero, *s* is assumed to be the address of an array of at least **L_tmpnam** bytes; *tmpnam* places its result in that array and returns *s* as its value. *Tmpnam* generates a different file name each time it is called. Files created using *tmpnam* and either *fopen* or *creat* are only temporary in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use *unlink* (2) to remove the file when its use is ended.

File names are initially of the form /usr/tmp/[a-z][a-z][a-z]XXXXXX and are then passed to *mktemp* before returning the result.

SEE ALSO

creat(2), unlink(2), fopen(3S), mktemp(3C).

BUGS

If called more than 17,576 times in a single process, *tmpnam* will start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using *tmpnam* or *mktemp*, and the file names are chosen so as to render duplication by other means unlikely.

NAME

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS

#include <math.h>

double sin (x)
double x;

double cos (x)
double x;

double tan (x)
double x;

double asin (x)
double x;

double acos (x)
double x;

double atan (x)
double x;

double atan2 (y, x)
double x, y;

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

Sin, *cos* and *tan* return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.

Acos returns the arc cosine in the range 0 to π.

Atan returns the arc tangent of x in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arc tangent of y/x in the range $-\pi$ to π.

HARDWARE DEPENDENCIES

Series 210:
The approximate limit is 1.49 x 10^8.

Series 500/600/700:
The approximate limit is 1.49 x 10^8.

The algorithms used for all functions except *atan2* are from HP 9000 BASIC.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause *asin* and *acos* to return value 0.

For *sin*, *cos*, *tan*, if the argument is out of range of accurate calculations, *errno* is set to EDOM and zero is returned.

NAME

ttyname, isatty – find name of a terminal

SYNOPSIS

char *ttyname (fildes)

int isatty (fildes)

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *fildes*.

Isatty returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

FILES

/dev/*

DIAGNOSTICS

Ttyname returns a null pointer (0) if *fildes* does not describe a terminal device in directory /**dev**.

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

ungetc – push character back into input stream

SYNOPSIS

#include <stdio.h>

int ungetc (c, stream)
char c;
FILE *stream;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Ungetc pushes the character *c* back on an input stream. That character will be returned by the next *getc* call on that stream. *Ungetc* returns **EOF** on error, zero on success.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push **EOF** are rejected.

Fseek(3S) erases all memory of pushed back characters.

SEE ALSO

fseek(3S), getc(3S), setbuf(3S).

DIAGNOSTICS

Ungetc returns **EOF** if it cannot push a character back.

NAME

intro – introduction to special files

HP-UX COMPATIBILITY

Remarks: In general, device drivers are not portable across systems; however, every effort has been made to make their behavior portable. Due to variation in hardware, this is not always possible. Programs which use these driver's directly are at higher than average risk of not being portable.

DESCRIPTION

This section describes various special files that refer to specific HP peripherals and device drivers. The names of the entries are generally derived from the type of device being described (disc, plotter, etc.), not the names of the special files themselves. Characteristics of both the hardware device and the corresponding HP-UX device driver are discussed where applicable.

HARDWARE DEPENDENCIES

Series 500/600/700:

You cannot open a block special file for reading or writing.

The following naming convention for special files is recommended:

[**r**] dev_id [prod_no] [model_initial] [**s**|**d**] [.digit]

where:

r	indicates a raw device;
dev_id	consists of one of the following mnemonics:

Mnemonic	Meaning
hd	hard disc
mt	tape
fd	flexible disc
lp	printer
dig	digitizer
plt	plotter or graphics CRT

prod_no	the HP product number for the device;
model_initial	the initial (if any) associated with the device product number;
s|d	used with HP 9895A; specifies whether media is single-sided (s) or double-sided (d);
.digit	used when more than one identical device is connected to the system; for example, if two HP 2631G printers are connected, their special files would be named lp2631g.0 and lp2631g.1.

NAME

disc – direct disc access

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	HP
Remarks:	This manual page describes disc information for the Series 500/600/700. Refer to other *disc*(4) entries for information valid for your implementation.

DESCRIPTION

This page describes the actions of the general HP-UX disc drivers when referring to a disc as either a block or character special device.

Block special files access discs via the system's normal buffering mechanism. They may be read and written without regard to physical disc records. Buffering must be done in such a way that concurrent accesses through multiple opens or mounts of the same physical device do not get out of phase.

There is also a "raw" interface via a character special file which provides for direct transmission between the disc and the user's read or write buffer. A single read or write call results in exactly one I/O call. Therefore raw I/O is considerably more efficient when many bytes are transmitted.

In "raw" I/O, each operation is completed to the device before the call returns.

The name of a "raw" device is typically the same as the name of the corresponding block device prefixed with an **r**.

In raw I/O (through character special files), there may be implementation-dependent restrictions on the alignment of the buffer in memory. Each transfer must occur on a sector boundary, and read a whole number of sectors. The sector size is a hardware-dependent value.

Each raw access is independent of other raw accesses and of block accesses to the same physical device. They may or may not be in phase.

SEE ALSO

intro(4), mknod(8).

WARNING

On some systems, having both a mounted file system and a block special file open on the same device is asking for trouble; this should be avoided if possible. This is because it may be possible for some files to have private buffers in some systems.

NAME

hpib – hpib interface information

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP
Remarks:	This information is valid for the Series 500/600/700 only. Refer to other *hpib*(4) manual pages for information valid for other implementations.

DESCRIPTION

HP-IB is Hewlett-Packard's implementation of the Institute of Electrical and Electronic Engineers Standard Digital Interface for Programmable Instrumentation. For more information about the standard, consult any of the following documents:

IEEE Std 488-1978
IEC Pub 625-1
ANSI MC1.1

A read operation on a device connected to an HP-IB configures the computer as "listener" and the device as "talker". The read operation terminates when either the number of bytes requested has been transferred, or the device asserts the EOI (end or identify) line. A write operation configures the computer as "talker" and the device as "listener". The write operation terminates when the number of bytes specified has been transferred.

Devices connected to an HP-IB are addressed using three values. The first value, called the *major value*, is always 12. The second value is called the *select code*. The select code refers to the I/O slot number to which the device is connected. The third value is called the ***HP-IB address***. The HP-IB address is usually set by an in-line or rotary switch on the device itself, and is in the range 0 through 30. Refer to the device reference manual for information on setting the HP-IB address.

HP-UX does not support secondary addressing, or multiple "listeners" at the same select code.

SEE ALSO

section 4, mknod(8), particular device documentation.

NAME

lp – printer information

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP

DESCRIPTION

All file names in /**dev** containing the mnemonic *lp* are special files providing the interface to a particular printer.

If the *lp* mnemonic is preceded by the character **r**, then data is sent to the printer in *raw mode.* In raw mode, no interpretation is done on the data to be printed, and no page formatting is performed. The bytes are simply sent to the printer and printed as is.

If the *lp* mnemonic is **not** preceded by the character **r**, then the data is interpreted according to the following rules:

a form-feed is generated after every 66 lines;

the new-line character is mapped into the carriage-return/line-feed sequence;

tab characters are expanded into the appropriate number of blanks (tab stops are assumed to occur every eight character positions);

lines longer than 132 characters are truncated.

FILES

/dev/lp	default or standard printer used by some HP-UX commands;
/dev/[r]lp*	special files for printers

SEE ALSO

lpr(1), intro(4).

NAME

magtape – magnetic tape interface and controls

HP-UX COMPATIBILITY

Level: Magnetic Tape Support -- HP-UX/RUN ONLY

Origin: UCB and HP

DESCRIPTION

The files /**dev**/**mt*** refer to specific tape drives; the behavior of the specific unit is specified by several bits in the device specifier.

There are three bits controlling the operation of the tape drive. The exact positioning of these bits is implementation dependent. For information concerning how these bits are set, see your system administrator.

rewind When this bit is set, the tape is automatically rewound upon close. This is normally done for units numbered 0-3 and 8-11.

mode When this bit is set, the tape drive behaves like the Berkeley tape drivers; when clear the drive behaves like System III. The details are described below. The *ioctl* operations described below work in both modes on "raw" tapes only.

density When set, the tape drive is run at 1600 bpi; when clear it is run at 800 bpi. The 800 bpi drives are usually numbered 0-7, and 1600 bpi are usually numbered 8-15.

When opened for reading or writing, the tape is assumed to be positioned as desired.

When a file is opened for writing and then closed, a double end-of-file (double tape mark) is written. If the device has the rewind bit set, the tape is rewound; otherwise, the tape is positioned before the second EOF just written.

When a read-only file is closed and the rewind bit is set, the tape is rewound. If the rewind bit is not set, the behavior depends on the mode bit. For System III compatibility, the tape is positioned after the EOF following the data just read. For Berkeley compatibility, the tape is not re-positioned in any way.

The EOF is returned as a zero-length read.

By judiciously choosing **mt** files, it is possible to read and write multi-file tapes.

A tape treated as a block device consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The **mt** files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The raw interface is described below:

The associated files are named **rmt***. Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given.

During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. The number of bytes ignored is available in the mt_resid field of the mtget structure via the MTIOCGET *ioctl*. In raw tape I/O, the buffer and size may have implementation dependent alignment restrictions. Seeks are ignored, instead the *ioctl* operations described below are available. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

Note that the HP 88140L/S cartridge tapes behave like a disc when accessed in raw mode (i.e. the record length is fixed). It is *not* recommended that it be mounted and treated like a disc.

The *ioctl* system call can be used to manipulate magnetic tapes; the following is included from **sys/mtio.h** and describes the possible operations:

```
/* mtio.h */
/* Structures and definitions for mag tape
 * I/O control commands
 */

/* mag tape I/O control commands */

#define     MTIOCTOP      (('m'<<8)|1)        /* do mag tape op */
#define     MTIOCGET      (('m'<<8)|2)        /* get mag tape status */

/* structure for MTIOCTOP - mag tape op command */

struct      mtop {
        short           mt_op;              /* operations defined below */
        daddr_t         mt_count;           /* how many of them */
};

/* operations */

#define     MTWEOF    0         /* write end-of-file record */
#define     MTFSF     1         /* forward space file */
#define     MTBSF     2         /* backward space file */
#define     MTFSR     3         /* forward space record */
#define     MTBSR     4         /* backward space record */
#define     MTREW     5         /* rewind */
#define     MTOFFL    6         /* rewind, put drive offline */
#define     MTNOP     7         /* no-op, sets status only */

/* structure for MTIOCGET - mag tape get status command */

struct      mtget {
        long        mt_type;        /* type of magtape dev. */
        long        mt_resid;       /* residual count */

/* The following two registers are device dependent */

        long        mt_dsreg;       /* drive status register */
        long        mt_erreg;       /* error register */

/* The following two are not yet implemented */

        daddr_t     mt_fileno;      /* file no. of current pos. */
        daddr_t     mt_blkno;       /* blk no. of current pos. */

/* end not yet implemented */
};

/*
 * Constants for mt_type long
 */

#define     MT_ISTS           01
#define     MT_ISHT           02
#define     MT_ISTM           03
#define     MT_IS7970E        04
```

HARDWARE DEPENDENCIES

Series 500/600/700:

Block magnetic tape is not supported.

FILES

/dev/mt*
/dev/rmt*

NAME

null – null file (" bit bucket ")

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

tty – general terminal interface

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

This section describes both a particular special file and the general nature of the terminal interface.

The file /**dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

As for terminals in general: all of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by *getty*(8) and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. This limit is dependent on the particular implementation, but is at least 256. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT (Control-\ or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory if the implementation supports core files.

ERASE	(#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.
KILL	(@) deletes the entire line, as delimited by a NL, EOF, or EOL character.
EOF	(Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NUL) is an additional line delimiter, like NL. It is not normally used.
STOP	(Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a ***hangup*** signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in <**termio.h**>:

```
#define     NCC  8
struct      termio {
        unsigned     short    c_iflag;          /* input modes */
        unsigned     short    c_oflag;          /* output modes */
        unsigned     short    c_cflag;          /* control modes */
        unsigned     short    c_lflag;          /* local modes */
                     char     c_line;           /* line discipline */
        unsigned     char     c_cc[NCC];        /* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	INTR	DEL
1	QUIT	FS
2	ERASE	#
3	KILL	@
4	EOF	EOT
5	EOL	NUL
6	reserved	

7 reserved

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.
IENQAK	0020000	Enable output pacing control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7 bits, otherwise all 8 bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

If IENQAK is set, the system will transmit ASCII ENQ after every 80 characters sent and then wait until the terminal responds with ASCII ACK. The terminal will respond in this way when it has sufficiently emptied its buffer. If the terminal does not respond after 5 seconds, the system will resume transmission anyway. The ASCII ACK that the terminal sends will not get entered into the input queue if it was sent in response to ASCII ENQ.

The initial input control value is all bits clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.

ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000037	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B900	0000011	900 baud
B1200	0000012	1200 baud
B1800	0000013	1800 baud
B2400	0000014	2400 baud
B3600	0000015	3600 baud
B4800	0000016	4800 baud
B7200	0000017	7200 baud
B9600	0000020	9600 baud
B19200	0000021	19200 baud
B38400	0000022	38400 baud
EXTA	0000036	External A
EXTB	0000037	External B
CSIZE	0000140	Character size:
CS5	0	5 bits
CS6	0000040	6 bits
CS7	0000100	7 bits
CS8	0000140	8 bits
CSTOPB	0000200	Send two stop bits, else one.
CREAD	0000400	Enable receiver.
PARENB	0001000	Parity enable.
PARODD	0002000	Odd parity, else even.
HUPCL	0004000	Hang up on last close.
CLOCAL	0010000	Local line, else dial-up.
CRTS	0020000	Assert request-to-send.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

Modem control is defined as asserting data-terminal-ready whenever the device is open, and dropping it on closing the device the last time. Any open to a modem-controlled device will hang until carrier detect is asserted unless the open is asynchronous (O_NDELAY). Without modem control, data-terminal-ready is not asserted, and the state of carrier-detect is ignored.

Asynchronous changes to CLOCAL on an open line directly control the data-terminal-ready modem control line. If CLOCAL is asynchronously turned off on an already open line, subsequent opens (with wait) will hang until carrier-detect is asserted, already open lines will not be affected. Reads to lines which are not in CLOCAL and for which carrier-detect is not asserted will return end-of-file, writes will be discarded as if they had been successful.

If CRTS is set, the request-to-send line is asserted, otherwise it is not. This line may be raised or lowered at any time as needed for controlling the attached device.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase, kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g. 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use*:
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **a**, **n** as **n**, and **N** as **n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA	Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.
TCSETA	Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.
TCSETAW	Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
TCSETAF	Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK	Wait for the output to drain. If *arg* is 0, then send a break (zero bits for at least 0.25 seconds).
TCXONC	Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
TCFLSH	If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

Conversion Aids for stty(2)/gtty(2)

The following conversion information is supplied to help you port programs from pwb/V6 UNIX to your current HP-UX system. Note that these conversions do **not** work for programs ported from Version 7 UNIX, since some Version 7 flags are defined differently.

The following data structure is defined in the include file **sgtty.h**:

```
struct sgttyb {
        char      sg_ispeed;        /* input speed */
        char      sg_ospeed;        /* output speed */
        char      sg_erase;         /* erase character */
        char      sg_kill;          /* kill character */
        short     sg_flags;         /* mode flags */
}
```

The flags, as defined in **sgtty.h**, are:

```
#define     HUPCL     01
#define     XTABS     02
#define     LCASE     04
#define     ECHO      010
#define     CRMOD     020
```

```
#define    RAW       040
#define    ODDP      0100
#define    EVENP     0200
#define    ANYP      0300
#define    NLDELAY   001400
#define    TBDELAY   002000
#define    CRDELAY   030000
#define    VTDELAY   040000
#define    BSDELAY   0100000

#define    CR0       0
#define    CR1       010000
#define    CR2       020000
#define    CR3       030000
#define    NL0       0
#define    NL1       000400
#define    NL2       001000
#define    NL3       001400
#define    TAB0      0
#define    TAB1      002000
#define    NOAL      004000
#define    FF0       0
#define    FF1       040000
#define    BS0       0
#define    BS1       0100000
```

When the *ioctl* **TIOCSETP** (*stty*(2)) command is executed, the flags in the old **sgttyb** structure are mapped into their new equivalents in the **termio** structure. Then the **TCSETA** command is executed as defined above.

The following table shows the mapping between the old **sgttyb** flags and the current **termio** flags. Note that flags contained in the **termio** structure that are not mentioned below are cleared.

HUPCL (if set)
    sets the **termio** HUPCL flag;
HUPCL (if clear)
    clears the **termio** HUPCL flag;
XTABS (if set)
    sets the **termio** TAB3 flag;
XTABS (if clear)
    clears the **termio** TAB3 flag;
TBDELAY (if set)
    sets the **termio** TAB1 flag;
TBDELAY (if clear)
    clears the **termio** TAB1 flag;
LCASE (if set)
    sets the **termio** IUCLC, OLCUC, and XCASE flags;
LCASE (if clear)
    clears the **termio** IUCLC, OLCUC, and XCASE flags;
ECHO (if set)
    sets the **termio** ECHO flag;
ECHO (if clear)
    clears the **termio** ECHO flag;
NOAL (if set)
    sets the **termio** ECHOK flag;
NOAL (if clear)

clears the **termio** ECHOK flag;

CRMOD (if set)
: sets the **termio** ICRNL and ONLCR flags; also, if CR1 is set, the **termio** CR1 flag is set, and if CR2 is set, the **termio** ONOCR and CR2 flags are set;

CRMOD (if clear)
: sets the **termio** ONLRET flag; also, if NL1 is set, the **termio** CR1 flag is set, and if NL2 is set, the **termio** CR2 flag is set;

RAW (if set)
: sets the **termio** CS8 flag, and clears the **termio** ICRNL and IUCLC flags; also, default values of 6 characters and 0.1 seconds are assigned to MIN and TIME, respectively;

RAW (if clear)
: sets the **termio** BRKINT, IGNPAR, ISTRIP, IXON, IXANY, OPOST, CS7, PARENB, ICANON, and ISIG flags; also, the default values control-D and null are assigned to the control characters EOF and EOL, respectively;

ODDP (if set)
: if EVENP is also set, clears the **termio** INPCK flag; otherwise, sets the **termio** PARODD flag;

VTDELAY (if set)
: sets the **termio** FFDLY flag;

VTDELAY (if clear)
: clears the **termio** FFDLY flag;

BSDELAY (if set)
: sets the **termio** BSDLY flag;

BSDELAY (if clear)
: clears the **termio** BSDLY flag.

In addition, the **termio** CREAD bit is set, and, if the baud rate is 110, the CSTOPB bit is set.

When using **TIOCSETP**, the *ispeed* entry in the **sgttyb** structure is mapped into the appropriate speed in the **termio** CBAUD field. The *erase* and *kill* **sgttyb** entries are mapped into the **termio** erase and kill characters.

When the *ioctl* **TIOCGETP** (*gtty*(2)) command is executed, the current **TCGETA** command is first executed. The resulting **termio** structure is then mapped into the **sgttyb** structure, which is then returned to the user.

The following table shows how the **termio** flags are mapped into the old **sgttyb** structure. Note that all flags contained in the **sgttyb** structure that are not mentioned below are cleared.

HUPCL (if set)
: sets the **sgttyb** HUPCL flag;

HUPCL (if clear)
: clears the **sgttyb** HUPCL flag;

ICANON (if set)
: sets the **sgttyb** RAW flag;

ICANON (if clear)
: clears the **sgttyb** RAW flag;

XCASE (if set)
: sets the **sgttyb** LCASE flag;

XCASE (if clear)
: clears the **sgttyb** LCASE flag;

ECHO (if set)
: sets the **sgttyb** ECHO flag;

ECHO (if clear)
: clears the **sgttyb** ECHO flag;

ECHOK (if set)
: sets the **sgttyb** NOAL flag;

ECHOK (if clear)
: clears the **sgttyb** NOAL flag;

PARODD (if set)
: sets the **sgttyb** ODDP flag;

PARODD (if clear)
: clears the **sgttyb** ODDP flag;

INPCK (if set)
: sets the **sgttyb** EVENP flag;

PARODD, INPCK (if both clear)
: sets the **sgttyb** ODDP and EVENP flags;

ONLCR (if set)
: sets the **sgttyb** CRMOD flag; also, if CR1 is set, the **sgttyb** CR1 flag is set, and if CR2 is set, the **sgttyb** CR2 flag is set;

ONLCR (if clear)
: if CR1 is set, the **sgttyb** NL1 flag is set, and if CR2 is set, the **sgttyb** NL2 flag is set;

TAB3 (if set)
: sets the **sgttyb** XTABS flag;

TAB3 (if clear)
: clears the **sgttyb** XTABS flag;

TAB1 (if set)
: sets the **sgttyb** TBDELAY flag;

TAB1 (if clear)
: clears the **sgttyb** TBDELAY flag;

FFDLY (if set)
: sets the **sgttyb** VTDELAY flag;

FFDLY (if clear)
: clears the **sgttyb** VTDELAY flag;

BSDLY (if set)
: sets the **sgttyb** BSDELAY flag;

BSDLY (if clear)
: clears the **sgttyb** BSDELAY flag.

When using **TIOCGETP**, the **termio** CBAUD field is mapped into the *ispeed* and *ospeed* entries of the **sgttyb** structure. Also, the **termio** erase and kill characters are mapped into the *erase* and *kill* **sgttyb** entries.

Note that, since there is not a one-to-one mapping between the **sgttyb** and **termio** structures, unexpected results may occur when using the older **TIOCSETP** and **TIOCGETP** calls. Thus, the **TIOCSETP** and **TIOCGETP** calls should be replaced in all future code by the current equivalents, **TCSETA** and **TCGETA**, respectively.

HARDWARE DEPENDENCIES

Series 500:

The following abbreviations are used for the following dependencies: **ASI** stands for the HP 27128A Asynchronous Serial Interface; **MUX** stands for the HP 27130A 8-Channel Asynchronous Multiplexor; **Model 20** stands for the internal terminal of the HP 9000 Model 20 Series 500/600/700 computer.

[ASI/MUX/Model 20] There is no support for output delays for certain characters, tab expansion, or upper- to lower-case mapping.

[ASI/MUX] The kill character is echoed as <backslash><CR><LF>, and the ECHOK flag is set equal to the ECHO flag.

[ASI/MUX/Model 20] When the type-ahead limit is reached, input is not flushed. Rather, further characters are either ignored (ASI/MUX) or cause a beep (Model 20).

[ASI/MUX] There is no parity error marking on terminal input.

[ASI/MUX] The echoing of carriage-return and new-line characters may not be quite as expected in more obscure driver configurations.

[ASI/MUX] EOF character echoing is not suppressed on terminal input.

[ASI/MUX/Model 20] There is no support for the ONLRET, ONOCR, and OCRNL flags on terminal output.

[ASI/MUX/Model 20] The MIN and TIME parameters for raw terminal input are not supported.

[ASI/MUX/Model 20] The ECHONL flag is not supported.

[ASI/MUX/Model 20] If ECHOE is set and ECHO is not, nothing is echoed for the erase character.

[ASI] The baud rate, number of bits/character, parity, and the CLOCAL flag are governed by switches on the card.

[MUX] The CLOCAL flag is permanently set.

[ASI] On terminals that are hard-wired via direct-connect cables, the carrier-detect line is always false (cleared). Thus, clearing the CLOCAL flag causes a hang-up signal to be sent.

[Model 20] The following *ioctl* flags are not supported, because there is no asynchronous data communication to deal with: IGNPAR, PARMRK, INPCK, IXOFF, IENQAK, CBAUD, CSIZE, CSTOPB, PARENB, PARODD, HUPCL, CLOCAL, CRTS.

FILES

/dev/tty
/dev/tty*
/dev/console

SEE ALSO

stty(1), ioctl(2), stty(2), mknod(8).

BUGS

European modems are not properly supported.

NAME
intro – introduction to file formats

HP-UX COMPATIBILITY
Remarks: Header files are often used to hide hardware incompatibilities.

DESCRIPTION
This section outlines the formats of various files. The C **struct** declarations for the file formats are given where applicable. Usually, these structures can be found in the directories **/usr/include** or **/usr/include/sys**.

NAME

a.out – executable linker output file

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	HP
Remarks:	This manual entry describes the *a.out* file format for the Series 500/600/700. Refer to other *a.out* manual pages for information valid for other implementations.

DESCRIPTION

A.out is the output file of the linker *ld*(1). *Ld* will make *a.out* executable if there are no errors during compilation and linking, and no unresolved external references.

This file has five sections – a file header, a segment table, a segment information section, a symbol table(s) section, and a name pool(s) section. It looks as follows:

File Header
Segment Table
Segment Information – segment image (code/data) – fix-up information (loader) – relocation information (*ld*)
Symbol Tables: – linker symbol table – information for debugger support (not currently implemented)
Name Pool (strings)

Note that the above pictorial representation represents the logical order of the file, not necessarily the physical order. A description of each section of the file follows.

File Header

The *a.out* file header is conceptually divided into two pieces. The first is a section of "scalar" values, and the second is a "file map" containing data pertaining to the rest of the file. The entire file header is made up of 128 bytes of information, 32 of which make up the scalar section. The following is a pictorial representation of the scalar section:

Byte	
0	System ID : File Type
4	Reserved for Future Use
8	Flags
12	Program Entry Point
16	Reserved for Future Use
20	Reserved for Future Use
24	Reserved for Future Use
28	Reserved for Future Use

Each horizontal "slice" represents a word made up of four eight-bit bytes. The first word is called the "magic number", which is made up of two half-words called the system ID and the file type. The system ID identifies the target machine upon which the object code will run. The file type specifies whether or not the file is executable (hex 107), relocatable (hex 106), or shareable (hex 108).

The third word is used to specify the settings of three flags. The left-most three bits of this word are significant; the remainder of the word is ignored. Bit 1, the left-most of the flag bits, marks the program as using a single data segment, if set. You can override this with the **–T** or **–A** *ld* options, which force the program to reside in one or two data segments, respectively. Bit 2 marks the file as relinkable, if set (meaning that the file contains relocation records and a symbol table). Bit 3 marks the file as debuggable, if set.

The *Program Entry Point* word contains an external program pointer (EPP) referencing the starting code for the program. *Ld* normally assigns the starting address of the main program to this word. This can be changed with the **-e** linker option.

The file map portion of the header looks as follows:

Byte	
32	Code Segment Tbl: offset
36	Code Segment Tbl: size
40	Code Seg Images: offset
44	Code Seg Images: size
48	Data Segment Tbl: offset
52	Data Segment Tbl: size
56	Data Seg Images: offset
60	Data Seg Images: size
64	Link Symbol Tbl: offset
68	Link Symbol Tbl: size
72	Reserved for Future Use
76	Reserved for Future Use
80	Reserved for Future Use
84	Reserved for Future Use
88	Reserved for Future Use
92	Reserved for Future Use
96	Name Pool: offset
100	Name Pool: size
104	Reserved for Future Use
108	Reserved for Future Use
112	Reserved for Future Use
116	Reserved for Future Use
120	Reserved for Future Use
124	Reserved for Future Use

Each *offset* entry in the file map shows where the given section starts, relative to the beginning of the *a.out* file. Each *size* entry gives the size, in bytes, for that section.

Segment Table
The segment table collects, in one place, all information about the code and data segments making up the program. The segment table consists of an array of entries. Each entry describes one code or data segment of the program.

The following information is given for both code and data segment table entries:

> a *segment name*, which consists of an offset into the name pool, relative to the beginning of the name pool. This is useful for symbolically referring to code or data segments (not currently implemented).
>
> a *segment type*, which specifies one of three possible types of segments – code, direct data (in GDS), or indirect data (in GDS or EDS).
>
> a list of *segment attributes*. The segments can be paged, virtual, demand loadable, writable, and privileged. The linker sets the attributes for executable files.
>
> a *segment offset*, which references a particular code or data segment within the segment image area. The reference is given relative to the beginning of the segment image area.
>
> a *segment size*, which is the size, in bytes, of the particular code or data segment being described in the entry.
>
> a *segment fixup size*, which specifies the size, in bytes, of the loader fixup area in the particular segment being described.
>
> a *segment relocation information size*, which specifies the number of bytes of relocation records for this segment.

The following information is given for data segment table entries only:

> a *segment limit*, which specifies the maximum number of bytes that the indirect data segment can contain. Attempting to increase the size beyond this stated limit results in an error. The linker assigns a default value of 1.5 megabytes to this field, but it may be changed with the **–m** linker option.
>
> a *segment zero-padding size*, which is a byte count of the uninitialized data area. The linker computes this value from the data relocation records.

The following information is given for code segment table entries only:

> a *segment local procedures count*, which specifies the number of procedures defined in that segment, but only known locally within it.
>
> a *segment external procedures count*, which specifies the number of procedures defined in that segment, but externally known.

Several words are left unused in each segment table entry to allow for future growth.

Segment Information
This section of the file contains the segment images for each segment included in the final, executable file. This section contains a subsection for each program segment. Each subsection is in turn made up of three parts – the contents of the segment (code or data), a list of pointers that the loader must "fix up" in that segment, and the relocation records for that segment. Each subsection looks as follows:

Code/Data Image
Loader Fixup Information
Relocation Records

The code image contains the compiled machine code for each program segment. The data image contains an image of initialized data for the program. Contained in this code are pointers. The loader fixup information area contains offsets that reference these pointers (the offsets are given relative to the beginning of the code/data image area). These offsets must be "fixed up" at run time (i.e., the program loader *exec* must update the segment number fields with the correct values). The linker generates the loader fixup information.

Symbol Tables

The linker symbol table contains data on relocatable symbols relevant to the linker (e.g. name and type for each global symbol). Refer to *nm*(1) (Series 500/600/700 only) for a complete description of each symbol type and the parameters associated with them. The contents of the symbol table may be listed in several different ways with *nm*.

Name Pool

The name pool contains a list of null-terminated strings, which specify the names of the symbols in the program. The symbol table entries contain indexes into the name pool instead of the names themselves. This permits arbitrarily long names to be used instead of fixed-length names. The first string in the name pool is always a null string. This enables zero to be used as an index into the name pool for entities which have no names.

SEE ALSO

magic(5).

NAME

ar – archive file format

SYNOPSIS

#include <ar.h>

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Ar(1) is used to concatenate several files into an archival file. Archives are used mainly as libraries to be searched by the link editor *ld*(1).

A file produced by *ar* has a magic_number at the start, followed by the constituent files, each preceded by a file header. The magic_number is a 32-bit word as defined in **magic.h** (see *magic*(5)). The header of each file is 28 bytes long:

```
        struct   ar_hdr {
                 long    ar_date;
                 long    ar_size;
                 short   ar_mode;
                 char    ar_uid;
                 char    ar_gid;
                 char    ar_name[14];
                 short   ar_fill;
        };
```

Each file begins on an even byte boundary; null bytes are inserted between files if necessary. Nevertheless, the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

All HP-UX versions share the same archive format, which may not match that of other UNIX machines.

SEE ALSO

ar(1), arcv(1), ld(1), magic(5).

NAME

checklist – list of file systems processed by fsck

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Checklist resides in directory /**etc** and contains a list of at most 15 *special file* names. Each *special file* name is contained on a separate line and corresponds to a file system. Each file system will then be automatically processed by the *fsck*(8) command.

HARDWARE DEPENDENCIES

Series 500/600/700:

There is no limit to the number of special file names in *checklist.*

SEE ALSO

fsck(8).

NAME

cpio – format of cpio archive

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

The *header* structure, when the **c** option is not used, is:

```
struct {
          short     h_magic,
                    h_dev,
                    h_ino,
                    h_mode,
                    h_uid,
                    h_gid,
                    h_nlink,
                    h_rdev,
                    h_mtime[2],
                    h_namesize,
                    h_filesize[2];
          char      h_name[h_namesize rounded to word];
} Hdr;
```

When the **c** option is used, the *header* information is described by the statement below:

```
sscanf(Chdr, "%6ho%6ho%6ho%6ho%6ho%6ho%6ho%6ho%11lo%6ho%11lo",
   &Hdr.h_magic,&Hdr.h_dev,&Hdr.h_ino,&Hdr.h_mode,
   &Hdr.h_uid,&Hdr.h_gid,&Hdr.h_nlink,&Hdr.h_rdev,
   &Longtime,&Hdr.h_namesize,&Longfile);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The contents of each file is recorded together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in *stat*(2). The length of the null-terminated path name *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name TRAILER!!!. Directories and the trailer are recorded with *h_filesize* equal to zero.

It will not always be the case that h_dev and h_ino correspond to the results of *stat(2)*, but the values are always sufficient to tell whether two files in the archive are linked to each other.

When a device special file is archived by HP-UX *cpio* (using -x), h_rdev will contain a magic constant which is dependent upon the implementation which is doing the writing. h_rdev flags the device file as an HP-UX 32-bit device specifier, and h_filesize will contain the 32-bit device specifier (see *stat(2)*). If the -x option is not present, special files are not archived or restored. Non-HP-UX device special files are never restored.

SEE ALSO

cpio(1), find(1), stat(2).

NAME

crontab – scheduling file for cron(8)

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

The file *crontab* consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (0-6, with 0 = Sunday). Each of these patterns may contain:

a number in the (respective) range indicated above;
two numbers separated by a minus (indicating an inclusive range);
a list of numbers separated by commas (meaning all of these numbers); or
an asterisk (meaning all legal values).

The sixth field is a string that is executed by the shell at the specified time(s). A **%** in this field is translated into a new-line character. Only the first line (up to a **%** or the end of the line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

EXAMPLES

```
0 0 * * * /etc/backup –fsck
```

This example executes *backup*(8) at midnight every night, all year long.

```
0,15,30,45 2–23 * * * /usr/lib/atrun
```

This example looks for and executes programs scheduled through *at*(1) every quarter hour from 2am until 11pm, every day, all year long.

FILES

/usr/lib/crontab

SEE ALSO

cron(8).

NAME

dir – SDF directory format

SYNOPSIS

#include <types.h>
#include <sys/dir.h>

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP
Remarks:	This entry describes the SDF directory format for the Series 500/600/700. Refer to other *dir*(5) manual pages for information valid for other implementations.

DESCRIPTION

A directory behaves exactly like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a field in its i-node entry (see *inode*(5)). The structure of a directory entry as given in **sys/dir.h** is:

```
#ifndef DIRSIZ
#define DIRSIZ 14
#endif

struct direct
{
        char     d_name[DIRSIZ + 2];     /* 16-char file name */
        short    d_object_type;          /* not referenced by UNIX */
        short    d_file_code;            /* not referenced by UNIX */
        ino_t    d_ino;                  /* use fir # for i-node */
};
```

The SDF implementation of directories eliminates the entries for **.** and **..**. Instead, this information is available as part of the i-node.

File names are stored in directories in a special manner. There are two such cases:

When a file name has **imbedded blanks**, the blanks are represented by the null character on the disc. This is apparent when accessing the disc via raw (character) mode.

When a file name is **blank padded**, all unspecified characters are set to blanks. Again, these are only shown when reading from the disc in raw mode.

Reading from a directory that has been opened via *open*(2), shows file names to be null-terminated and to have imbedded blanks where they belong.

SEE ALSO

fs(5), inode(5).

NAME

errfile – system error logging file

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP
Remarks:	This manual page describes *errfile* as implemented on the Series 500/600/700. Refer to other *errfile* manual pages for information valid for other implementations.

DESCRIPTION

Errfile is a logging file containing lines of ASCII text. Each line describes certain system errors that have occurred, or warnings about serious system conditions. Only those system error messages deemed serious enough to be of interest to the system administrator are logged. Urgent messages are also written to /dev/console.

HP-UX creates *errfile* if it does not exist.

The system administrator needs to check the contents of *errfile* periodically and note errors that need his/her attention. Also, *errfile* tends to grow without bounds, so outdated information needs to be removed on a regular basis.

FILES

/usr/adm/errfile

NAME

fs – format of system volume

SYNOPSIS

#include <sys/param.h>
#include <sys/filsys.h>

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP
Remarks:	This manual page describes the format of the system volume as implemented on the Series 500/600/700. Refer to other *fs*(5) manual pages for information valid for other implementations.

DESCRIPTION

Every Structured Directory Format (SDF) volume is divided into logical blocks, the size of which is selected when *init* is executed. Block 0 is the superblock. It has the following format:

```
struct filsys {
        ushort      s_format;         /* disc fmt, should = 0x700 Unix */
        ushort      s_corrupt;        /* non-zero if directory corrupt */
        char        s_fname[16];      /* root dir name, blank padded */
        time_ios    s_init;           /* date initialized / unique id */
        int         s_blksz;          /* no. bytes per block */
        daddr_t     s_boot;           /* boot area starting block */
        int         s_bootsz;         /* size of boot area in blks */
        daddr_t     s_fa;             /* FA file starting block */
        int         s_version;        /* version no., 0 for Unix */
        daddr_t     s_maxblk;         /* largest addressable blk */
        char        s_passwd[16];     /* volume password, Unix unused */
        time_ios    s_bkup;           /* last backup date, Unix unused */
                                      /* rest of blk unused */
};
```

The file attributes file (FA file) begins at the block specified by *s_fa* in the superblock. It has five major sections:

Entry 0: I-node for FA file
Entry 1: I-node for root dir (/)
Entry 2: Unused
Entry 3: Free map . . . Entry n:
Entry n + 1: File entries . . . End of FA File

Each entry consists of 128 bytes. Entry 0 is the i-node of the FA file itself (see *inode*(5) for a description of the i-node structure). Entry 1 is the i-node for the file system's root directory, /.

Entry 3 through entry n consists of the free map, which keeps track of every free (unused) block of memory on the device. The free map contains a bit for each block on the device. If a bit is set, the corresponding block of memory is free; otherwise, the corresponding block is being used. The free map is zero-padded to guarantee that it ends on a 128-byte boundary.

Entry n + 1 through the end of the FA file contains an entry for every file in the system. Each entry is either an i-node, an extent map, or unused. An extent map contains 128 bytes of information, and looks as follows:

```
struct em_rec {
        ushort      e_type;        /* =2 for extent maps */
        ushort      e_exnum;       /* # extents in this rec. */
        int         e_res1;        /* unused */
        ino_t       e_next;        /* next map in list; none = neg */
        ino_t       e_last;        /* last map in list; none = neg */
        ino_t       e_inode;       /* owner i-node no. */
        daddr_t     e_boffset;     /* blk offset of 1st extent */
                                   /* from start of file */
        struct {
                daddr_t            e_startblk;    /* extent start blk */
                int                e_numblk;      /* # blks in extent */
        } e_extent[13];
};
```

FILES

/usr/include/sys/param.h
/usr/include/sys/filsys.h
/usr/include/sys/ino.h

SEE ALSO

inode(5), fsck(8).

NAME

fspec – format specification in text files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

It is sometimes convenient to maintain text files on UNIX with non-standard tabs, (i.e., tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns;
2. a – followed immediately by an integer *n*, indicating tabs at intervals of *n* columns;
3. a – followed by the name of a "canned" tab specification.

Standard tabs are specified by **t–8**, or equivalently, **t1,9,17,25,**etc. The canned tabs which are recognized are defined by the *tabs*(1) command.

s*size* The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t–8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

Several HP-UX commands correctly interpret the format specification for a file. Among them is *gath* (see *send*(1C)) which may be used to convert files to a standard format acceptable to other HP-UX commands.

SEE ALSO

ed(1), reform(1), send(1C), tabs(1).

BUGS

Does not work with *vi*(1) and *ex*(1).

NAME

group, grp.h - group file

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

/etc/group contains for each group the following information:

group name
encrypted password
numerical group ID
comma-separated list of all users allowed in group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.

Actually, /etc/passwd defines the group id for each user. /etc/group exists to supply names for each group, and to support changing groups via *newgrp*(1).

This file resides in directory **/etc**. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

Grp.h describes the group structure returned by getgrent(3), etc:

```
struct      group {             /* see getgrent(3) */
            char        *gr_name;
            char        *gr_passwd;
            int         gr_gid;
            char        **gr_mem;
};
```

FILES

/etc/group

SEE ALSO

newgrp(1), passwd(1), crypt(3C), getgrent(3), passwd(5).

BUGS

There is no tool that helps you ensure that /etc/passwd and /etc/group are compatible.

There is no tool that helps you set group passwords in /etc/group.

NAME

inittab – control information for init(8)

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

When a system state is entered, *init* reads the file /etc/**inittab**. Entries in this file have the format:

state:id:flags:command

The *state* field contains an integer in the range 1 through 9. When *init* reads **inittab**, only those lines beginning with an integer matching *init*'s new state are read and evaluated.

Id is a two-character string supplied by the user. This string is associated with the process created by the **inittab** entry, if any. *Id* can then be used to refer to that process in later entries in **inittab**, without having to know its process number.

Flags can be any of the characters **c**, **o**, **t**, and/or **k**. The **o** flag specifies that *init* is to ignore the entry. The **t** and **k** flags specify that the existing process (if any) associated with *id* is to be terminated (signal 15) or killed (signal 9), or both. They may be specified in either order. Note that the signal is sent to all processes in the process group associated with *id*, at the time the new state is entered. The **c** flag specifies that the process associated with *id* is to be continually reinvoked each time it terminates. This field may be left blank, in which case the created process, if any, is spawned only once and allowed to terminate.

Command is a character string specifying a shell command to be invoked. The string may include any legal arguments for the command, as well as I/O redirection and/or pipeline symbols. *Init* spawns a shell to execute the command. The resulting process is remembered by the specified *id*. All processes run as if invoked by the super-user. Each process is a group leader. This field may be empty, in which case no new process is created.

EXAMPLES

Examine the contents of the /etc/inittab file shipped with your system for examples of *inittab* entries.

FILES

/etc/inittab

SEE ALSO

init(8).

NAME

inode – format of an i-node

SYNOPSIS

#include <sys/types.h>
#include <sys/param.h>
#include <sys/ino.h>

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

Remarks: This entry describes the i-node structure for the Series 500/600/700. Refer to other *inode*(5) manual pages for information valid for other implementations.

DESCRIPTION

An i-node for an ordinary file or directory in a file system has the following structure, as defined in **sys/ino.h**:

```
/*
* I-node structure as it appears on disc. This i-node is actually
* a file information record (FIR) in the HP SDF disc format.
*/

struct dinode {
        ushort      di_type      /* =1 for inodes */
        ushort      di_ftype;    /* file type */
        ushort      di_count;    /* reference count */
        short       di_uftype;   /* user file type (LIF) */
        time_ios    di_ctime;    /* time created */
        unsigned    di_other;    /* public capabilities */
        ino_t       di_protect;  /* file protect rec. none=-1 */
        ino_t       di_label;    /* file label rec. none=-1 */
        int         di_blksz;    /* file size in blocks */
        int         di_max;      /* largest byte writable */
        ushort      di_exsz;     /* recom. extent size */
        ushort      di_exnum;    /* no. i-node extents (1-4) */
        struct   {
                daddr_t    di_startblk;  /* extent start blk */
                int        di_numblk;    /* no. blks in extent */
        } di_extent[4];
        ino_t       di_exmap;    /* inode 1st extent map; none = -1 */
        int         di_size;     /* current size, bytes */

        /* Warning! Next 2 fields apply only to directories */

        ino_t       di_parent;   /* inode of parent */
        char        di_name[16];/* name of this directory */

        /* The remaining fields defined only for local */
        /* implementation of structured directory format. */

        time_t      di_atime;    /* time last accessed */
        time_ios    di_mtime;    /* time last mod. */
        int         di_recsz;    /* logical record size */
        ushort      di_uid;      /* owner's user id */
        ushort      di_gid;      /* owner's group id */
        ushort     di_mode;     /* mode, type of file */
        char       di_res2[2];  /* unused */

        /* The next field used only if file is */
        /* a device file; otherwise it is zero */

        dev_t      di_dev;      /* description of device */
};
```

The meaning of the type declarations included above can be found in *types*(7).

FILES

/usr/include/sys/ino.h

SEE ALSO

dir(5), fs(5), types(7).

NAME

magic – magic numbers for HP-UX implementations

SYNOPSIS

#include <magic.h>

HP-UX COMPATIBILITY

Level: Use: HP-UX/RUN ONLY
Header: HP-UX/DEVELOPMENT

Origin: HP

DESCRIPTION

Magic.h localizes all information about HP-UX "magic numbers" in one file, and thus facilitates uniform treatment of this entity. This file specifies the offset of the number within a file (always the start of the file) and the structure of that field:

```
struct magic_number
 {
  unsigned short int system_id;
  unsigned short int file_type;
 };
typedef struct magic_number MAGIC;
```

Magic.h includes definitions for the system IDs of all HP machines running HP-UX, and file types that are common to all implementations. There may be additional implementation-dependent file types. The predefined file types are:

```
/* for object code files */
#define RELOC_MAGIC  0x106  /* relocatable only */
#define EXEC_MAGIC   0x107  /* normal executable */
#define SHARE_MAGIC  0x108  /* shared executable */


/* for archive files */
#define AR_MAGIC     0xFF65 /* ar format */
```

SEE ALSO

ar(1), chatr(1), ld(1), a.out(5), ar(5)

BUGS

Cpio files use a different form of magic number that is incompatible with ***magic***(5).

NAME

mknod – create a special file entry

SYNOPSIS

#include <mknod.h>

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

DESCRIPTION

Mknod.h provides utilities to pack and unpack device names as used by *mknod(2)*. It contains the macro **dev** = **makedev(major, minor)** which packs the major and minor fields into a form suitable for *mknod(2)*. It also contains **major(dev)** and **minor(dev)** which extract the corresponding fields. The macro MINOR_FORMAT is a *printf* specification that prints the minor field in the format best suited to the particular implementation. The specification given by MINOR_FORMAT must cause the resulting string to indicate the base of the number in the same format as that used for C: no leading zero for decimal, leading zero for octal, and leading zero and 'x' for hexadecimal.

When a minor field is printed in the format specified by MINOR_FORMAT, each sub-field contained in the minor will be wholly contained in the mininum possible number of digits of the resulting string. (Splitting a field across unnecessary digits for the sake of packing is not done.)

SEE ALSO

mknod(2), section 4, mknod(8).

NAME

mnttab – mounted file system table

SYNOPSIS

```
struct mnttab {
        char        mt_dev[MNTLEN];
        char        mt_filsys[MNTLEN];
        short       mt_ro_flg;
        time_t      mt_time;
};
```

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

Mnttab resides in directory /etc and contains a table of devices mounted by the ***mount***(1) command.

Each entry is (2 x MNTLEN + 6) bytes in length (MNTLEN is defined in /usr/include/mnttab.h). The first MNTLEN bytes are the null-padded name of the place where the *special file* is mounted; the next MNTLEN bytes represent the null-padded root name of the mounted special file; the remaining 6 bytes contain the mounted *special file*'s read/write permissions and the date on which it was mounted.

The maximum number of entries in ***mnttab*** is based on the system parameter NMOUNT defined in /usr/include/mnttab.h, which defines the maximum number of mounted special files allowed.

HARDWARE DEPENDENCIES

Series 210:
  MNTLEN = 10, NMOUNT = 20.

Series 500/600/700:
  MNTLEN = 17, NMOUNT = 20.

SEE ALSO

mount(1).

NAME

model – HP-UX machine identification

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: HP

SYNOPSIS

#include <model.h>

DESCRIPTION

There are some distinctions between the implementations of HP-UX due to hardware differences. Where such distinctions exist, conditional compilation or other definitions can be used to isolate the differences. Flags and typedefs to resolve these distinctions are collected in *model.h*. This file contains constants indentifying various HP-UX implementations. The current values are:

```
HP_S_200        /* HP 9000 Series 200 */
HP_S_500        /* HP 9000 Series 500 */
```

Other such constants will be added as HP-UX extends to other machines.

In addition, *model.h* has a statement defining the preprocessor constant *MYSYS* to represent the specific implementation for which compilation is desired. *MYSYS* will be equal to one of the constants above.

Conditional compilation may be used to adapt one file for execution on more than one HP-UX implementation, if it contains implementation- or architecture-dependent features. For instance,

```
#if MYSYS = = HP_S_200
        <statements>
#endif
```

will cause the statements following the if statement to be compiled only for the HP 9000 Series 200.

Model.h also contains typedefs for several predefined types to enhance portability of certain types of code and of files. The meaning of the typedefs listed below is given in cc(1).

int8
u_int8
int16
u_int16
int32
u_int32
int64
u_int64
machptr
u_machptr
longmachptr
u_longmachptr

HARDWARE DEPENDENCIES

Series 210:

A conditional compilation variable, **HP_9000_S_200**, is implemented. It is predefined to the C preprocessor.

Series 500/600/700:

A conditional compilation variable, **HP_9000_S_500**, is implemented. It is predefined to the C preprocessor.

SEE ALSO

cc(1)

NAME

passwd, pwd.h - password file

HP-UX COMPATIBILITY

Level:	Multi-user - HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Passwd contains the following information for each user:

login name
encrypted password
numerical user ID
numerical group ID
comment field
initial working directory
program to use as shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a new-line. If the password field is null, no password is required. If the shell field is null, */bin/sh* is used.

This file resides in directory */etc*. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

The encrypted password consists of 13 characters chosen from a 64 character alphabet (., /, **0–9**, **A–Z**, **a–z**), except when the password is null in which case the encrypted password is also null. Login can be prevented by entering a character that is not part of the above alphabet in the password field (e.g. ∗).

Each character in the above alphabet is part of the base 64 character set, and has a decimal value as shown in the following table:

Base 64 Character:	.	\	0-9	A-Z	a-z
Decimal Equivalent:	0	1	2-11	12-37	38-63

Password aging is in effect for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.) This string defines the "age" needed to implement password aging. The first character of the age denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired will be forced to supply a new one. The next character denotes the minimum period in weeks which must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.)

If the first and second characters are both zero (i.e. first and second characters are both '.'), the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the password file). If the second character has a greater decimal equivalent then the first (e.g. './'), only the super-user will be able to change the password.

Pwd.h designates the broken out password file as obtained by getpwent(3C):

```
struct passwd {
        char                    *pw_name;
        char                    *pw_passwd;
        unsigned int            *pw_uid;
        unsigned int            *pw_gid;
        char                    *pw_age;
        char                    *pw_comment;
        char                    *pw_gecos;
        char                    *pw_dir;
        char                    *pw_shell;
};
```

HARDWARE DEPENDENCIES

Series 210/500/600/700:

The following fields have character limitations as noted:

the login name field can be no longer than 8 characters;

the initial working directory field can be no longer than 63 characters;

the program field can be no longer than 44 characters.

The results are unpredictable if these fields are longer than the limits specified above.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(1), a64l(3C), crypt(3C), getpwent(3C), group(5).

NAME

profile – set up user's environment at login time

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

If the file /etc/profile exists, it is executed by the shell for every user who logs in. /etc/profile should be set up to do only those things that are desirable for *every* user on the system, or to set reasonable defaults.

If your login (home) directory contains a file named *.profile*, that file will also be executed by the shell before your session begins. *.Profile* files are useful for setting various environment parameters, setting terminal modes, or overriding some or all of the results of executing /etc/profile.

FILES

/etc/profile
$HOME/.profile

SEE ALSO

env(1), login(1), mail(1), sh(1), stty(1), su(1), environ(7), term(7).

NAME

ranlib – table of contents format for object libraries

SYNOPSIS

#include <ranlib.h>

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	HP
Remarks:	This manual page describes the table of contents format for the Series 500/600/700. Refer to other *ranlib*(5) manual pages for information valid for other implementations.

DESCRIPTION

Ranlib creates a table of contents for object file libraries, thus allowing the linker *ld* to scan libraries in random (rather than sequential) order.

Ranlib executes the *ar* command to install the table of contents as the first element of the library. The file name of the table of contents is always **__.SYMDEF**.

The table of contents lists each externally known name in the archive, together with the offset of the archive element that defines that name. This offset is useful as an input argument to *lseek*(2) or *fseek*(3).

The **__.SYMDEF** file contains the table of contents and a name pool of strings (the names of external symbols). This allows for symbols with arbitrarily long names. The **rl_hdr** structure defines the layout of the file, and the **rl_ref** structure defines the contents of a table of contents entry. These structures have the following format:

```
struct rl_hdr {
        long int rl_tcbas;              /* offset of table */
        long int rl_tclen;              /* length of table */
        long int rl_nmbas;              /* offset of name pool */
        long int rl_nmlen;              /* length of name pool */
};

struct rl_ref {
        long int name_pos;              /* index into name pool */
        long int lib_pos;               /* offset of defining file */
};
```

SEE ALSO

ar(1), ld(1), ranlib(1), ar(5).

NAME

sccsfile – SCCS file format

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the **ASCII SOH** (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form **DDDDD** represent a five digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

@**hDDDDD**

The value of the checksum is the sum of all characters, except those of the first line. The @**h** provides a *magic number* of octal 000550 (0168 hex).

Delta table

The delta table consists of a variable number of entries of the form:

```
@s DDDDD/DDDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
 .
 .
 .
@c <comments> ...
 .
 .
 .
@e
```

The first line (@**s**) contains the number of lines inserted/deleted/unchanged respectively. The second line (@**d**) contains the type of the delta (currently, normal: **D**, and removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The @**i**, @**x**, and @**g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The @**m** lines (optional) each contain one **MR** number associated with the delta; the @**c** lines contain comments associated with the delta.

The @**e** line ends the delta table entry.

User names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @**u** and @**U**. An empty list allows anyone to make a delta.

Flags

Keywords used internally (see *admin*(1) for more information on their use). Each flag line takes the form:

```
@f <flag>        <optional text>
```

The following flags are defined:

```
@f t    <type of program>
@f v    <program name>
@f i
@f b
@f m    <module name>
@f f    <floor>
@f c    <ceiling>
@f d    <default-sid>
@f n
@f j
@f l    <lock-releases>
@f q    <user defined>
```

The **t** flag defines the replacement for the **%Y%** identification keyword. The **v** flag controls prompting for **MR** numbers in addition to comments; if the optional text is present it defines an **MR** number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the **b** flag is present the **–b** keyletter may be used on the *get* command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the **%M%** identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a *get* command. The **n** flag causes *delta* to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes *get* to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing (*get*(1) with the **–e** keyletter). The **q** flag defines the replacement for the **%Q%** identification keyword.

Comments

Arbitrary text surrounded by the bracketing lines @**t** and @**T**. The comments section typically will contain a description of the file's purpose.

Body The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, represented by:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

admin(1), delta(1), get(1), prs(1).
Source Code Control System User's Guide in *HP-UX: Selected Articles.*

NAME

termcap – terminal capability data base

SYNOPSIS

/etc/termcap

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: UCB

DESCRIPTION

Termcap is a data base describing terminals used by *vi*(1). Terminals are described in *termcap* by giving a set of their capabilities, and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap*.

Entries in *termcap* consist of a number of colon-separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by vertical bar characters. The first name is always 2 characters long. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may contain blanks for readability.

Several examples of *termcap* entries for various terminals are included in the following discussion. Note that these examples do not imply support of the associated terminals. Also note that the *termcap* data base may contain entries for non-HP terminals that have not been tested.

Capabilities

(P) indicates padding may be specified
(P*) indicates that padding may be based on the number of lines affected

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not **^H**
bs	bool		Terminal can backspace with **^H**
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default **^M**)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line

dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give ":ei=:" if **ic**
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default **^L**)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give ":im=:" if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by "other" function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of "keypad transmit" mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of "other" keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in "keypad transmit" mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on "other" function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than **^I** or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines even though it doesn't overstrike

up	str	Upline (cursor up)
us	str	Start underscore mode
vb	str	Visible bell (may not move cursor)
ve	str	Sequence to end open/visual mode
vs	str	Sequence to start open/visual mode
xb	bool	Beehive (f1 = escape, f2 = ctrl C)
xn	bool	A newline is ignored after a wrap (Concept)
xr	bool	Return acts like **ce** \r \n (Delta Data)
xs	bool	Standout not erased by writing over it (HP 264?)
xt	bool	Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the HP series 264X terminals, is among the more complex entries in the *termcap* file as of this writing.

```
h4|h1|hp|hp2645|2645|hp 264x series:\
   :if=/usr/lib/tabset/stdcrt:\
   :al=\EL:am:bs:cd=\EJ:ce=\EK:ch=\E&a%dC:cl=\EH\EJ:cm=6\E&a%r%dc%dY:\
   :co#80:cv=\E&a%dY:da:db:dc=\EP:dl=\EM:ei=\ER:im=\EQ:\
   :kb=^H:ku=\EA:kd=\EB:kl=\ED:kr=\EC:kh=\Eh:ks=\E&s1A:ke=\E&s0A:\
   :li#24:mi:ml=\El:mu=\Em:nd=\EC:pt:se=\E&d@:so=\E&dJ:\
   :us=\E&dD:ue=\E&d@:up=\EA:xs:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability. Capabilities in *termcap* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two-letter codes. For instance, the fact that a terminal has "automatic margins" (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the terminal includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co**, which indicates the number of columns the terminal has, gives the value '80' for the terminal.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either an integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5', to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a **:** in a capability it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability it must be encoded as **\200**. The routines which deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it, or bugs in *ex*. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in */etc/termcap*.

Basic Capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H**, in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on, i.e. **am**.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the model 33 teletype is described as

```
t3 | 33 | tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl | adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor Addressing

Cursor addressing in the terminal is described by a **cm** string capability, containing escapes similar to those used by *printf*(3S) (%x). These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the **%** encodings have the following meanings:

%d	as in *printf*, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output.
%r	reverses order of line and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output.
%D	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is "cm=6\E&%r%2c%2Y". The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, "cm=^T%.%.". Terminals which use "%." need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "cm=\E=%+ %+ ".

Cursor Motions
If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area Clears
If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/Delete Line
If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/Delete Character
There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "abc def" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the "def" which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for "insert null". If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give the sequence **im** to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give the sequence **ei** to leave insert mode (assign an empty value for **ei** if you specified an empty value for **im**). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**; terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, Underlining, and Visible Bells
If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several flavors of standout mode, such as inverse video, blinking, or underlining (half bright is not usually an acceptable "standout" mode unless the terminal is in inverse video mode constantly), the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, then **ug** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex,* this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad
If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local. If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, ":ko=cl,ll,sf,sb:", which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete, and therefore is not recommended for future design. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, **:ma=^Kj^Zk^Xl:** indicates arrow keys left (^H), down (^K), up (^Z), and right (^X). (No home key is defined.)

Miscellaneous
If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, then this can be given as **ta**.

Terminals which don't allow '~' characters to be printed should indicate **hz**. Terminals which echo carriage return/linefeed for carriage return and then ignore a following linefeed should indicate **nc**. Terminals which ignore a linefeed immediately after an **am** wrap should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Terminals where tabs turn all characters moved over to blanks should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is */usr/lib/tabset/std* but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024 characters. Since ***termlib*** routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with **xx@** where xx is the capability. For example, the entry

```
hn|2621nl:ks@:ke@:tc=2621:
```

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/etc/termcap	file containing terminal descriptions

SEE ALSO

ex(1), termcap(3), tset(1), vi(1), ul(1), more(1).

BUGS

Ex allows only 256 characters for string capabilities, and the routines in *termcap(3)* do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

NAME

utmp, wtmp – utmp and wtmp entry format

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

The files **utmp** and **wtmp** hold user and accounting information for use by commands such as *who*(1), *acctcon1* (see *acctcon*(1)), and *login*(1). They have the following structure, as defined by <**utmp.h**>:

```
struct utmp
{
        char    ut_line[8];     /* tty name */
        char    ut_name[8];     /* login name */
        long    ut_time;        /* time on */
};
```

Note that **ut_line** is the tty name, except in the case of date change entries, when it is either a vertical bar (|) or a left brace ({) (see *date*(1)).

Ut_name is valid for login entries only; otherwise the first character is null. There are logout entries in both **utmp** and **wtmp**. In **utmp**, these entries refer to terminals that are not currently logged in; in **wtmp**, they record history.

Note that **wtmp** tends to grow without bound, and should be checked regularly. Information that is no longer useful should be removed periodically to prevent it from becoming too large.

HARDWARE DEPENDENCIES

Series 500/600/700:
Acctcon(1) is not currently supported.

FILES

/etc/utmp
/usr/adm/wtmp
/usr/include/utmp.h

SEE ALSO

acctcon(1), login(1), who(1), write(1).

NAME

intro – introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

NAME

environ – user environment

HP-UX COMPATIBILITY

Level: HP-UX/NUCLEUS

Origin: System III

DESCRIPTION

An array of strings called the "environment" is made available by *exec*(2) when a process begins. By convention, these strings have the form "name=value". The following names are used by various commands:

PATH The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (**:**). *Login*(1) sets **PATH=:/bin:/usr/bin**.

HOME Name of the user's login directory, set by *login*(1) from the password file *passwd*(5).

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm*(1) or *tplot*(1G), which may exploit special capabilities of that terminal.

TZ Time zone information. The format is **xxx***n***zzz** where **xxx** is standard local time zone abbreviation, *n* is the difference in hours from GMT, and **zzz** is the abbreviation for the daylight-saving local time zone, if any; for example, **EST5EDT**.

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh*(1), or by *exec*(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL**, **PS1**, **PS2**, **IFS**.

SEE ALSO

env(1), login(1), sh(1), exec(2), getenv(3C), profile(5), term(7).

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

The *fcntl*(2) function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open*(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/* (The first three can only be set by open(2)) */

#define    O_RDONLY    0
#define    O_WRONLY    1
#define    O_RDWR      2
#define    O_NDELAY    04        /* Non-blocking I/O */
#define    O_APPEND    010       /* append (guaranteed) */

/* Flag values accessible only to open(2) */

#define    O_CREAT     00400     /* open with file create (Uses third open arg) */
#define    O_TRUNC     01000     /* Open with truncation */
#define    O_EXCL      02000     /* exclusive open */

/* fcntl(2) requests */

#define    F_DUPFD     0         /* duplicate fildes */
#define    F_GETFD     1         /* Get close-on-exec flag */
#define    F_SETFD     2         /* set close-on-exec flag */
#define    F_GETFL     3         /* Get file status flags */
#define    F_SETFL     4         /* Set file status flags */
```

SEE ALSO

fcntl(2), open(2).

NAME

INIT, GETC, PEEKC, UNGETC, RETURN, ERROR, compile, step, advance – regular expression compile and match routines

SYNOPSIS

#define INIT <declarations>
#define GETC( ) <getc code>
#define PEEKC( ) <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;

int step(string, expbuf)
char *string, *expbuf;

int advance (string, expbuf)
char *string, *expbuf;

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

This page describes general purpose regular expression matching routines in the form of *ed*(1), defined in **/usr/include/regexp.h**. Programs such as *ed*(1), *sed*(1), *grep*(1), *bs*(1), *expr*(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is complex. Programs that include this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the *compile* routine.

GETC( )	Return the value of the next character in the regular expression pattern. Successive calls to GETC( ) should return successive characters of the regular expression.
PEEKC( )	Return the next character in the regular expression. Successive calls to PEEKC( ) should return the same character (which should also be the next character returned by GETC( )).
UNGETC(*c*)	Cause the argument *c* to be returned by the next call to GETC( ) (and PEEKC( )). No more than one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC( ). The value of the macro UNGETC(*c*) is always ignored.
RETURN(*pointer*)	This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.
ERROR(*val*)	This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression overflow.

The syntax of the *compile* routine is as follows:

```
compile(instring, expbuf, endbuf, eof)
```

The first parameter *instring* is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address that the compiled regular expression may occupy. If the compiled expression cannot fit in (*endbuf–expbuf*) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. For example, in *ed*(1), this character is usually a /.

Each program that includes this file must have a **#define** statement for INIT. This definition will be placed right after the declaration for the function *compile* and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a variable to point to the beginning of the regular expression so that this variable can be used in the declarations for GETC( ), PEEKC( ) and UNGETC( ). Otherwise it can be used to declare external variables that might be used by GETC( ), PEEKC( ) and UNGETC( ). See the example below of the declarations taken from *grep*(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

```
step(string, expbuf)
```

The first parameter to *step* is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter *expbuf* is the compiled regular expression which was obtained by a call of the function *compile*.

The function *step* returns the value one, if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step*. The variable set in *step* is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function *advance*, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* will point to the first character of *string* and *loc2* will point to the null at the end of *string*.

Step uses the external variable *circf* which is set by *compile* if the regular expression begins with ^. If this is set then *step* will only try to match the regular expression to the beginning of the string. If more than one regular expression is to be compiled before the the first is executed, the value of *circf* should be saved for each compiled expression and *circf* should be set to that saved value before each call to *step*.

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns a one indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called, simply call *advance*.

When *advance* encounters a * or \{ \} sequence in the regular expression it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string where the match first occurred at sometime during the backing up process, *advance* will break out of the loop that backs up and will return zero. This is used by *ed*(1) and *sed*(1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like **s/y*//g** do not loop forever.

The routines *ecmp* and *getrange* are trivial and are called by the routines previously mentioned.

EXAMPLES

The following is an example of how the regular expression macros and calls look from an old version of *grep*(1):

```
#define INIT               register char *sp = instring;
#define GETC( )            (*sp++)
#define PEEKC( )           (*sp)
#define UNGETC(c)          (--sp)
#define RETURN(c)          return;
#define ERROR(c)           regerr( )

#include <regexp.h>
...
        compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
        if(step(linebuf, expbuf))
                succeed( );
```

FILES

/usr/include/regexp.h

SEE ALSO

ed(1), grep(1), sed(1).

BUGS

The handling of *circf* is poor.

The routine *ecmp* is equivalent to the Standard I/O routine *strncmp* and should be replaced by that routine.

NAME

stat – data returned by stat/fstat system call

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

The system calls *stat* and *fstat*(2) return data whose structure is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

```
/*
 * Structure of the result of stat/fstat
 */

struct stat {
        dev_t       st_dev;
        ino_t       st_ino;
        ushort      st_mode;
        short       st_nlink;
        ushort      st_uid;
        ushort      st_gid;
        dev_t       st_rdev;
        off_t       st_size;
        time_t      st_atime;
        time_t      st_mtime;
        time_t      st_ctime;
};

#define     S_IFMT      0170000     /* type of file */
#define     S_IFDIR     0040000     /* directory */
#define     S_IFCHR     0020000     /* character special */
#define     S_IFBLK     0060000     /* block special */
#define     S_IFREG     0100000     /* regular */
#define     S_IFIFO     0010000     /* fifo */
#define     S_IFSRM     0150000     /* SRM special */
#define     S_IFNWK     0110000     /* network special */
#define     S_ISUID     04000       /* set user ID on ex. */
#define     S_ISGID     02000       /* set group ID on ex. */
#define     S_ISVTX     01000       /* save swapped text */
#define     S_IREAD     00400       /* read perm., owner */
#define     S_IWRITE    00200       /* write perm., owner */
#define     S_IEXEC     00100       /* ex/search perm., owner */
```

FILES

/usr/include/sys/types.h
/usr/include/sys/stat.h

SEE ALSO

stat(2).

NAME

term – conventional names

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

These names are used by certain commands (e.g., *nroff*(1), *mm*(1), *man*(1), *tabs*(1)) and are maintained as part of the shell environment (see *sh*(1), *profile*(5), and *environ*(7)) in the variable **$TERM** (note that publication of these names does not constitute support of these devices by Hewlett-Packard):

1520	Datamedia 1520
1620	Diablo 1620 and others using the HyType II printer
1620–12	same, in 12-pitch mode
2621	Hewlett-Packard HP2621 series
2631	Hewlett-Packard 2631 line printer
2631–c	Hewlett-Packard 2631 line printer - compressed mode
2631–e	Hewlett-Packard 2631 line printer - expanded mode
2640	Hewlett-Packard HP2640 series
2645	Hewlett-Packard HP264n series (other than the 2640 series)
300	DASI/DTC/GSI 300 and others using the HyType I printer
300–12	same, in 12-pitch mode
300s	DASI/DTC/GSI 300s
382	DTC 382
300s–12	same, in 12-pitch mode
3045	Datamedia 3045
33	TELETYPE® Model 33 KSR
37	TELETYPE Model 37 KSR
40–2	TELETYPE Model 40/2
4000A	Trendata 4000A
4014	Tektronix 4014
43	TELETYPE Model 43 KSR
450	DASI 450 (same as Diablo 1620)
450–12	same, in 12-pitch mode
735	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
tn1200	General Electric TermiNet 1200
tn300	General Electric TermiNet 300

Up to 8 characters, chosen from [–a–z0–9], make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a –. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept arguments of the form –T*term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **$TERM**, which, in turn, should contain *term*.

SEE ALSO

mm(1), nroff(1), tplot(1G), sh(1), stty(1), tabs(1), profile(5), environ(7).

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

The data types defined in the include file are used in HP-UX system code; some data of these types are accessible to user code:

```
#define    NREGS_S    13  /* no. of regs saved */

typedef    struct { int r[1]; }            *physadr;
typedef    long                            daddr_t;
typedef    char                            *caddr_t;
typedef    unsigned short                  ushort;
typedef    unsigned long                   ino_t;
typedef    short                           cnt_t;
typedef    long                            time_t;
typedef    int                             label_t[NREGS_S];
typedef    long                            dev_t;
typedef    long                            off_t;
typedef    long                            paddr_t;
```

The form *daddr_t* is used for disc addresses except in an i-node on disc, see *fs*(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

HARDWARE DEPENDENCIES

Series 500/600/700:

The types NREGS_S and *label_t* are not implemented.

SEE ALSO

fs(5).

NAME

intro – introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions on such topics as boot procedures, recovery from crashes, file backups, etc.

NAME

backup – backup or archive file system

SYNOPSIS

/lbin/backup [**–archive**] [**–fsck**]

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP

DESCRIPTION

Backup uses *find*(1) and *cpio*(1) to save on the default tape drive (/**dev**/**mt**) a *cpio* archive of all files which have been modified since the modification time of /**etc**/**archivedate**. *Backup* should be periodically invoked by *cron*(8) at night, or when the system is otherwise idle.

The **–archive** option causes *backup* to save all files, regardless of their modification date, and then update /**etc**/**archivedate** using *touch*(1).

Backup prompts you to mount a new tape and continue if there is no more room on the current tape. Note that this prompting does not occur if you are running *backup* from *cron*(8).

The **–fsck** option causes *backup* to start a file system consistency check (without correction) after the backup is complete. This is the normal mode of nightly operation. For correct results, it is important that the system be effectively single-user while *fsck* is running, especially if it is allowed to automatically fix whatever inconsistencies it finds. *Backup* does not ensure that the system is single-user.

You should edit /**lbin**/**backup** to "customize" it for your system. For example, *backup* uses *tcio*(1) by default. You will need to modify *backup* to use *cpio*(1) if you want to access a raw device.

Several parameters are used which can be customized:

backupdirs	specifies which directories to recursively back up (usually /, meaning all directories);
backuplog	file name where start and finish times, block counts, and error messages are logged;
archive	file name whose date is the date of the last archive;
remind	file name that is checked by /**etc**/**profile** to remind the next person who logs in to change the backup tape;
rootdev	list of places for *fsck* (usually a character special file that points to the root device);
fscklog	file name where start and finish times and *fsck* output is logged.

You may want to make other changes, such as whether or not *fsck* does automatic correction (according to its arguments), where *cpio* output is directed, other information logging, etc.

In all cases, the output from *backup* is a normal *cpio* archive file (or volume) which can be read using *cpio* with the **c** and **B** options.

FILES

/etc/archivedate
parameterized file names

SEE ALSO

cpio(1), find(1), touch(1), cron(8), fsck(8).

BUGS

When *cpio* runs out of tape, it sends an error to *stderr* (which is logged, so it does not appear on your CRT), and demands a new special file name from /dev/tty. To continue, rewind the tape, mount the new tape, type the name of the new special file at the system console, and press **RETURN**.

If *backup* is left running overnight and the tape runs out, *backup* terminates, leaving the *find* process still waiting. You need to kill this process when you return.

NAME

catman – create the cat files for the manual

SYNOPSIS

/etc/**catman** [–**p**] [–**n**] [–**w**] [sections]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	UCB

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the *nroff* input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* will recreate the /**usr**/**lib**/**whatis** database.

If there is one parameter not starting with a '–', it is taken to be a list of manual sections to look in. For example

catman 123

will cause the updating to only happen to manual sections 1, 2, and 3.

Options:

–**n**	prevents creation of /**usr**/**lib**/**whatis**.
–**p**	prints what would be done instead of doing it.
–**w**	causes only the /**usr**/**lib**/**whatis** database to be created. No manual reformatting is done.

FILES

/usr/man/man?/*.*	raw (*nroff* input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/mkwhatis	commands to make whatis database

SEE ALSO

man(1).

NAME

chsys – change to different operating system or version

SYNOPSIS

/lbin/chsys sysname

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Chsys* is implemented on the Series 500/600/700 only.

DESCRIPTION

Chsys is a shell script that enables you to boot a different operating system, or a different version of the same operating system, using only one boot area on one disc. *Sysname* is one of a number of operating system names defined within *chsys*. *Chsys* uses *oscp*(8) to rebuild the boot area on */dev/rhd* with the selected system, reading from ordinary files containing operating system code. *Chsys* then invokes *osck*(8) to confirm that the new system is "healthy". (Note that *osck* performs a redundant check, so its invocation in *chsys* may be removed if you want to save time.)

Chsys invokes *oscp* as quietly as possible. *Chsys* causes *oscp* to read the new system ID string from a file selected by the *sysname* given, and redirects the output from *oscp* to */dev/null*. If *oscp* and *osck* are successful, *chsys* calls *stopsys –r* to switch to the new operating system. Note that *oscp* and *osck* together can take longer than a minute to run. During this time, *chsys* keeps you informed as to what actions are being taken.

If you simply want to reboot the operating system already in the boot area, do **not** use *chsys*. Instead, invoke *stopsys –r* directly.

If you want to allocate and use several boot areas on several discs, see *osmgr*(8).

You should modify *chsys* to localize it for your system. You may want to add or delete available *sysname*s, change the names or meanings of *sysname*s, change the name of the character special file (*/dev/rhd*) which points to the boot volume, etc. *Chsys* recognizes four default *sysname*s. They stand for:

HP-UX Model 20 single-user minimal system;
HP-UX Model 20 single-user complete system;
BASIC minimal system;
BASIC complete system.

These default *sysname*s serve as examples for any others you may want to add. They may or may not be useful to you. To see the actual *sysname*s, invoke *chsys* with no argument.

Chsys should only be invoked by the effective super-user unless both of the following are true:

the special file which points to the boot device must be readable and writable by whoever invokes *chsys*;

the *stopsys*(8) command must be owned by root and have the set-user-ID bit set.

If either of the above are not true, either the *oscp* or the *stopsys* command will fail.

RETURN VALUES

If any of the invoked commands fails, *chsys* writes a message to standard error and exits with the same return value as that returned by the unsuccessful command. *Chsys* returns 1 if invoked improperly.

SEE ALSO

sh(1), osmgr(8), shutdown(8), stopsys(8), sync(8).

WARNINGS

Chsys does not check that the system is idle, and it does not notify all users that the system is going down. You should usually execute *shutdown*(8) before executing *chsys*.

Chsys does not ask you to confirm that the intended operating system or version has been selected before the system is rebooted. However, *osck* ensures that the system is rebootable, and *stopsys –r* performs a *sync*(8). Note that new operating systems built in the boot area by *oscp* are always marked as loadable (see *osmark*(8)).

NAME

cron – clock daemon

SYNOPSIS

/etc/cron

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file */usr/lib/crontab* (see *crontab*(5)). Because *cron* never exits, it should be executed only once. This is best done by running *cron* from the initialization process through the file /**etc**/**rc** (see *init*(8)). *Cron* forks a copy of itself, thus explicit asynchronous execution (using **&**) is unnecessary.

Cron examines **crontab** once a minute to see if it has changed; if it has, *cron* reads it. Thus it takes only a minute for entries to become effective.

FILES

/usr/lib/crontab
/usr/lib/cronlog

SEE ALSO

sh(1), crontab(5), init(8).

DIAGNOSTICS

A history of all actions by *cron* is recorded in **/usr/lib/cronlog**, if it exists and is writable when *cron* starts.

BUGS

Cron reads **crontab** only when it has changed, but it reads the in-core version of that table once a minute.

NAME

fsck – file system consistency check, interactive repair

SYNOPSIS

/etc/fsck [**–y**] [**–n**] [**–s**] [**–d**] [*file system*] ...

HP-UX COMPATIBILITY

Level:	SDF File System – HP-UX/RUN ONLY
Origin:	HP
Remarks:	This manual page describes *fsck* as implemented on the Series 500/600/700. Refer to other *fsck* manual pages for information valid for other implementations.

DESCRIPTION

Fsck checks and interactively repairs inconsistent conditions for SDF file systems. If the file system is consistent, then the number of files, the number of blocks used, the number of blocks free, and the percent of volume unused are reported. If the file system is inconsistent, the operator is prompted for concurrence before each correction is attempted. Note that many corrective actions will result in some loss of data. The amount and severity of the loss 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, *fsck* will default to a **–n** action.

Fsck makes multiple passes over the file system, so care should be taken to ensure that the system is quiescent. You should unmount the file system being checked, if possible.

The following flags are interpreted by *fsck*:

–y Assume a **yes** response to all questions asked;

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

–s Ignore the actual free list and unconditionally reconstruct a new one. This option is useful in correcting multiply claimed blocks when one of the claimants is the free list. When using this option, the number of unclaimed blocks reported by *fsck* includes all the blocks in the free map. This can produce extensive output if **–d** is also selected.

–**s** should only be selected after a previous *fsck* indicates a conflict between a file and the free map. After *fsck –s* has executed, the integrity of the conflicting file(s) should be checked.

If **–s** is used to correct a problem on a virtual memory device, there is a high probability that the final step in *fsck* will fail and you will be forced to reboot. Should this occur, an appropriate error message will be printed.

–d Dump additional information. The more **d**'s that are present, the more information that is dumped. You may specify up to five **d**'s. Using more than two, however, can result in an overwhelming amount of output.

Fsck also recognizes, and ignores, the –**S** and –**t** options found in other versions of *fsck*. An appropriate warning is printed.

If no *file system*(s) are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Error messages from *fsck* are written to *stderr*. Information generated because of the **–d** option and normal output is written to *stdout*. Both are unbuffered.

Inconsistencies checked include:

1. Blocks claimed by more than one i-node, or by 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. Blocks not accounted for anywhere;
5. Bad i-node format;

6. Directory checks:
   Files pointing to unallocated i-nodes;
   I-node numbers out of range;
   Multiply linked directories;
   Link to the parent directory.

Orphaned files (allocated but unreferenced) with non-zero sizes are, with the operator's concurrence, reconnected by placing them in the *lost+found* directory. The name assigned is the i-node number. The only restriction is that *lost+found* must 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 creating *lost+found*, copying a number of files to it, and then removing them (before *fsck* is executed).

Orphaned directories and files with zero size are, with the operator's concurrence, returned directly to the free list. This will also happen if the *lost+found* directory does not exist.

You should run a backup prior to running *fsck* for repairs.

FILES

/etc/checklist	contains the default list of file systems to check

SEE ALSO

checklist(5), fs(5).

DIAGNOSTICS

The diagnostics are intended to be self-explanatory.

BUGS

All file systems must be described by a character special device file.

Do not redirect *stdout* or *stderr* to a file on the device being checked.

Fsck cannot check devices with a logical block size greater than 1024.

NAME

fwtmp, wtmpfix – manipulate wtmp records

SYNOPSIS

fwtmp [–**ic**]
wtmpfix [files]

HP-UX COMPATIBILITY

Level: Multi-user - HP-UX/EXTENDED

Origin: System III

DESCRIPTION

Fwtmp

Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formatted ASCII records. The ASCII version is useful to enable you to edit bad records or perform general purpose maintenance of the file.

The argument **–ic** is used to denote that input is in ASCII form, and output is to be written in binary form. (The arguments **i** and **c** are independent, respectively specifying ASCII input and binary output, thus **–i** is an ASCII to ASCII copy and **–c** is a binary to binary copy).

Wtmpfix

Wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A – can be used in place of *files* to indicate the standard input. If time/date corrections are not made, *acctcon1* will fault when it encounters certain date change records.

Each time the date is set while operating in multi-user mode, a pair of date change records are written to **/usr/adm/wtmp**. The first record is the old date denoted by | in the name field. The second record specifies the new date and is denoted by a { in the name field. *Wtmpfix* uses these records to synchronize all time stamps in the file.

FILES

/usr/adm/wtmp
/usr/include/utmp.h

SEE ALSO

utmp(5).

DIAGNOSTICS

Wtmpfix generates these diagnostics:

Cannot make temporary: xxx
  failed to make temp file

Input truncated at offset: xxx
  missing half of date pair

New date expected at offset: xxx
  missing half of date pair

Cannot read from temp: xxx
  read error

Bad file at offset: xxx
  ut_line entry not digit, alpha, nor "|" or "{" (first character only checked)

Out of core
  malloc fails. (Saves table of date changes)

No dtab
  software error (not seen yet)

BUGS

Fwtmp generates no errors, even on garbage input.

NAME

getty – set the modes of a terminal

SYNOPSIS

/etc/**getty** name [type [delay]]

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Getty is normally invoked by *init*(8) as the first step in allowing users to login to the system. Lines in /etc/**inittab** tell *init* to invoke *getty* with the proper arguments.

Name should be the name of a terminal in /**dev** (e.g., **tty03**); *type* should be a single character chosen from –, **0**, **1**, **2**, **3**, **4**, **5**, **6**, or **7**, which selects a speed table in *getty*, or **!**, which tells *getty* to update /etc/**utmp** and exit. *Type* defaults to zero.

Delay is relevant for dial-up ports only. It specifies the time in seconds that should elapse before the port is disconnected if the user does not respond to the **login:** request. *Delay* defaults to zero.

First, *getty* types the **login:** message. The **login:** message depends on the speed table being used. Then the user's login name is read, a character at a time.

While reading, *getty* tries to adapt to the terminal, speed, and mode that is being used. If a null character is received, it is assumed to be the result of a "break" ("interrupt"). The speed is then changed based on the speed table that *getty* is using, and **login:** is typed again. Subsequent breaks cause a cycling through the speeds in the speed table being used.

The user's login name is terminated by a new-line or carriage-return character (the characters you type in must match those specified for your login name exactly, including case). The latter results in the system being set to treat carriage returns appropriately. If the login name contains only upper-case alphabetic characters, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, *login*(1) is called with the user's login name as argument.

Speed sequences for the speed tables:

–	B110; for 110 baud console TTY.
0	B300–B150–B110–B1200; normal dial-up sequence starting at B300.
1	B150; no sequence.
2	B2400; no sequence.
3	B1200–B300–B150–B110; normal dial-up sequence starting at B1200.
4	B300; for console DECwriter.
5	B9600; no sequence.
6	B4800–B9600; for Tektronix 4014.
7	B4800;

HARDWARE DEPENDENCIES

Series 210:

The character **U** may be specified as a *type*; it represents a speed of 1200 baud.

Series 500/600/700:

The character **H** may be specified as a *type*; it represents a speed of 9600 baud for HP terminals.

Upper-case letters are mapped into lower-case during *getty* only.

SEE ALSO

login(1), tty(4), inittab(5), utmp(5), init(8).

BUGS

Ideally, the speed tables would be read from a file, not compiled into *getty*.

NAME

init – process control initialization

SYNOPSIS

/**etc**/**init** [state]

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

Init is invoked inside HP-UX as the last step in the boot procedure. It is process number one, and is the ancestor of every other process in the system. As such, it can be used to control the process structure of the system. If *init* is invoked with an argument by the super-user, it will cause an asynchronous change in the state of process one.

Init has 9 states, 1 through 9. It is invoked by the system in state 1, and it performs the same functions on entering each state. When a state is entered, *init* reads the file /**etc**/**inittab** (see *inittab*(5)).

After reading /**etc**/**inittab** and signaling running processes as required, but before invoking any processes under the new state, /**etc**/**rc** is invoked (see *rc*(8)). *Init* will also execute /**etc**/**rc** at the request of the operating system (e.g., when recovering from power failure). In this last case, the first argument has an **x** appended to it.

When /**etc**/**rc** has finished executing (or after five minutes), *init* invokes all *commands* waiting to be executed. (A *command* is waiting to be executed if there is no process currently running that has the same *id* as the command.) The *flag* **c** (continuous) requires the *command* to be continuously reinvoked whenever the process with that *id* dies. The *flag* **o** (off) causes the *command* to be ignored. This is useful for turning lines off without extensive editing. Otherwise, the *command* is invoked a maximum of one time in the current state.

Init invokes the *command* field read from /**etc**/**inittab** by opening / for reading on file descriptors 0, 1, and 2, resetting all signals to system default, setting up a new process group (*setpgrp*(2)), and *exec*ing:

/bin/sh –c "exec *command*"

Whenever *init* changes state, all commands saved for execution in the previous state are discarded.

Init inherits all orphan processes (those whose parent process died before the child). In such cases, *init* notices them when they terminate, but otherwise does nothing.

HARDWARE DEPENDENCIES

Series 500/600/700:

Init does not execute */etc/rc* at the request of the operating system, so *rc*'s first argument never has an **x** appended to it.

FILES

/etc/inittab
/etc/rc
/bin/sh
/dev/console

SEE ALSO

login(1), sh(1), exec(2), setpgrp(2), inittab(5), getty(8), rc(8).

DIAGNOSTICS

When *init* can do nothing else because of a missing /**etc**/**inittab** or when it has no children left, it will try to execute a shell on /**dev**/**console**. When the problem has been fixed, it is necessary to change states, and terminate the shell.

BUGS

Init does not complain if the state id pairs in /**etc**/**inittab** are not unique. For any given pair, the last one in the file is valid.

NAME

install – install commands

SYNOPSIS

/**etc/install** [**–c** dira] [**–f** dirb] [**–i**] [**–n** dirc] [**–o**] [**–s**] file [dirx ...]

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: System III

DESCRIPTION

Install is a command most commonly used in "makefiles" (see *make*(1)) to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

Install is useful for installing new commands, or new versions of existing commands, in the standard directories (i.e. /**bin**, /**etc**, etc.).

If no options or directories (*dirx* ...) are given, *install* will search (using *find*(1)) a set of default directories (/**bin**, /**usr**/**bin**, /**etc**, /**lib**, and /**usr**/**lib**, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

–c *dira*	Installs a new command in the directory specified in *dira*. Looks for *file* in *dira* and installs it there if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **–s** option.
–f *dirb*	Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to **755** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **–o** or **–s** options.
–i	Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options other than **–c** and **–f**.
–n *dirc*	If *file* is not found in any of the searched directories, it is put in the directory specified by *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options other than **–c** and **–f**.
–o	If *file* is found, this option saves the "found" file by moving it to **OLD***file* in the directory in which it was found. May be used alone or with any other options other than **–c**.
–s	Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

mk(8).

BUGS

Install cannot create alias links for a command (e.g. *vi* is an alias for *ex*).

NAME

killall – send signal to all user processes

SYNOPSIS

/**etc**/**killall** [signal]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Killall sends the specified *signal* to all user processes in the system, with the following exceptions:

the *init* process;

all processes (including background processes) associated with the terminal from which *killall* was invoked;

any *ps –ef* process, if owned by *root*;

any *sed –e* process, if owned by *root*;

any *shutdown* process;

any *killall* process;

any */etc/rc* process.

Killall obtains its process information from *ps*(1), and thus may not be able to perfectly identify which processes to signal.

If no *signal* is specified, signal 9 (kill) is sent by default.

Killall is invoked automatically by *shutdown*(8). The use of *shutdown* is recommended over using *killall* by itself.

SEE ALSO

kill(1), ps(1), signal(2), chsys(8), shutdown(8), stopsys(8).

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

HP-UX COMPATIBILITY

Level: HP-UX/RUN ONLY

Origin: System III

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

Makekey is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output will be pipes.

SEE ALSO

crypt(1), ed(1), passwd(5).

NAME

mkdev - make device files

SYNOPSIS

/etc/mkdev

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP

DESCRIPTION

This shell script helps the super-user install and maintain an HP-UX system. It consists of a machine-dependent list of commands which create one of each typical type of device file, with suggested default device addresses. It also creates mount directories for mountable volumes and changes permissions as appropriate for the device files.

This command sets a precedent for name and address conventions, and makes it easier to build (or rebuild) special files all at once.

Mkdev automatically changes the working directory (using *cd*) to /**dev** before starting execution.

Mkdev is specifically intended for modification before (each) use. Command lines for non-desired devices should be commented out with "#" so that they are still available for later use. You may want to use shorter device names than those suggested, especially for default devices.

SEE ALSO

chmod(1), mkdir(1), mknod(8).

DIAGNOSTICS

Each command line in *mkdev* is echoed as it is executed. Error messages, if any, are generated by the commands invoked.

Since the super-user must modify this script before using it the first time, an error is given if it has not been modified.

NAME

mknod – create special, fifo, network file

SYNOPSIS

/etc/**mknod** name [**c**|**b**|**s**] major minor
/etc/**mknod** name **p**
/etc/**mknod** name **n** nodename

HP-UX COMPATABILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. *Name* is the path name of the special file.

In the first synopsis shown, the second argument should be **b** if the special file is block-type (discs, tape), or **c** if it is character-type (other devices). The **s** option is used to create a Shared Resource Management (SRM) special file. *Major* and *minor* are numbers specifying the major device type (e.g. device driver number) and the minor device number (typically, but not exclusively, the unit, drive, HP-IB bus address and/or line number). *Major* and *minor* may be specified in hex, octal, or decimal, using the C language conventions (decimal numbers must not have a leading zero, octal must have a leading zero, and hex must have a leading zero followed by 'x').

Mknod can also be used to create fifo's (named pipes) (second synopsis shown).

The third synopsis shown is used to create a network special file, which addresses another node on a local area network. *Nodename* is the name by which the node is known on the network.

A real ID of 0 (super-user) is required on all except the second synopsis shown above.

The newly created file has a mode of 0666, minus the current setting of the user's *umask*.

SEE ALSO

lsdev(1), mknod(2), section 4, mknod(5).

NAME

optinstall, optupdate – install/update optional HP-UX products

SYNOPSIS

optinstall productnumber
optupdate productnumber

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Optinstall* and *optupdate* are implemented on the Series 500/600/700 only.

DESCRIPTION

Optinstall is used to configure an optional HP-UX software product to its initial state prior to use.

Optupdate is used to receive a periodic update of an HP-UX optional product when a prior release of the same product is already present.

Productnumber uniquely identifies the particular HP-UX software product to be installed or updated. It is a number of the form 97XXXA and is the same number that appears on the media on which the product is supplied.

Both routines are interactive. They print information about the addresses of the mass storage devices to be used, and accept input from the user which can change these addresses. Other variables which control the operation can also be interactively set.

Note that *optinstall* and *optupdate* work only on the 88140L/S tape cartridge media.

FILES

/dev/[r]hd*	
/dev/[r]mt*	device files enabling access of source and destination devices;
/optinstall.dir	directory on which destination disc is mounted (if not root).

SEE ALSO

unpack(1).

BUGS

Only one product may be installed or updated at a time, even if several products reside on the same tape.

There is no simple way to use *optinstall* from a shell script.

There is no way to prevent accidentally installing or updating an older version of a product over a newer one.

The routines fail if either the source media or the destination disc is currently mounted (see *mount*(1)). This restriction does not apply if the destination disc is the root device.

Only the super-user can execute *optinstall.*

It cannot be guaranteed that valid *productnumbers* will always be in the range 97XXXA.

NAME
oscк – check integrity of OS in SDF boot area(s)

SYNOPSIS
/lbin/osck [**–v**] volume

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Osck* is implemented on the Series 500/600/700 only.

DESCRIPTION
Osck checks one operating system in the boot area on the volume specified by *volume* (usually a block or character special file).

The OSF must be the first section of an *n*-section operating system. If *n* is greater than one, *osck* prompts for additional volumes as needed. The volumes must be mounted in order.

The **–v** (verbose) option causes *osck* to print additional information about each volume and each code segment as they are encountered. If **–v** is not specified, it is silent except for warnings, errors, and prompts for new volumes.

Osck checks the following:

OSF headers are valid and consistent across multiple volumes;

the first code segment is a power-up segment;

the code segment chain contains correct headers and lengths;

all segment checksums are correct;

the system terminates correctly after the last segment.

SEE ALSO
oscp(8), osmark(8), osmgr(8), sdfinit(8).

DIAGNOSTICS
Osck gives an appropriate error message and returns a non-zero value if *volume* cannot be accessed, or is not an SDF volume, there is no boot area, or the boot area contents appear invalid. Error messages are also given if any integrity violation is found. See *osmgr*(8) for a complete list of return values.

NAME
oscp – copy, create, append to, split operating system

SYNOPSIS
/lbin/oscp [**–o**] [**–v**] fromvolume tovolume
/lbin/oscp –m [**–v**] file ... tovolume
/lbin/oscp –a [**–v**] file ... tovolume
/lbin/oscp –s [**–v**] fromvolume

HP-UX COMPATIBILITY
Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Oscp* is implemented on the Series 500/600/700 only.

DESCRIPTION
Oscp enables you to perform:

boot-to-boot copy
Copy an operating system from the boot areas on one or more SDF volumes to the boot area on one SDF volume;

files-to-boot copy (**–m**, **–a** options)
Create a new operating system or append to an existing operating system from a list of ordinary files, and put the resulting system in one boot area;

boot-to-files copy (**–s** option)
Split up the segments in an operating system from one or more boot areas to one or more ordinary files.

Fromvolume and *tovolume* are usually special files.

Boot-to-Boot Copy
If **–m**, **–a**, and **–s** are not specified, *oscp* does boot-to-boot copy. For normal, multi-volume boot-to-boot copy, *oscp* requires that the operating system file (OSF) on the first *fromvolume* be the first section of an *n*-section operating system. If *n* is greater than one, *oscp* prompts you for additional volumes as required. The additional volumes must be mounted in order.

Before starting the copy, *oscp* clears the OSF header on *tovolume.* The OSF header values are corrected on *tovolume* after the copy is done. This new header may include a new system ID string that you enter when you are prompted (the same ID string displayed by the boot loader).

The **–o** (one volume only) option tells *oscp* to copy only one OSF (which may be part or all of a system) from *fromvolume* to *tovolume*, without changing the OSF header.

The **–v** (verbose) option tells *oscp* to print additional information about each volume as it is encountered. Otherwise, *oscp* is silent except for warnings, errors, and prompts for new volumes and new system ID strings.

Files-to-Boot Copy
If the **–m** (merge) option is given, *oscp* does a files-to-boot copy from the specified *files.* The source files may be BASIC/9000 BIN files or HP-UX ordinary files. The *files* must all be accessible and contain valid code segments. The code segments must all be of the same system type. The last code segment in each file must be followed by two null bytes.

Note that segments of unknown type, and old power-up segments (before February 1983) are "generic donors", and may be merged with any other type. Also note that, when creating a new system, *oscp* uses the first OSF header magic number in its internal list (i.e. 0xE9C28206).

Once you enter the new system ID string, *oscp* destroys the old OSF (if any) in the boot area before writing the new system.

The **–a** (append) option allows you to append code segments from ordinary files to an existing OSF on *tovolume*. There must be enough unused space in the boot area after the OSF, and the OSF must be a complete system in itself (i.e. volume 1 of 1). The existing OSF is not invalidated until the last segment is copied to the boot area.

In conjunction with **–m** or **–a**, the **–v** (verbose) option gives you additional information about the boot area and each segment as it is encountered.

Boot-to-Files Copy
The **–s** (split) option allows you to split an operating system into one or more ordinary files (HP-UX ordinary files only, not BASIC BIN files). For each code segment in the operating system, you are prompted for a file name to which the code segment is appended. If you enter a null line, the code segment is appended to the same file as was used in the previous append operation.

If the size of the specified file is greater than zero, *oscp* backs up two bytes from the end of the file to overwrite the previous terminator before appending the code segment to the file.

The **–v** (verbose) option gives you additional information about the boot area and each segment as it is encountered.

Note that the resulting ordinary files may be owned by the owner of the *oscp* command, depending on its permissions.

Copying to Boot Areas
Before beginning the copy, *oscp* prompts you for the 80-character operating system ID string to use for all volumes.

Before writing to *tovolume*, *oscp* first checks that it contains a boot area with sufficient unused space.

SEE ALSO

osck(8), osmark(8), osmgr(8), sdfinit(8).

DIAGNOSTICS

Oscp prints an appropriate error message and returns a non-zero value if *fromvolume* or *tovolume* cannot be accessed or is not an SDF volume, there is no boot area, the boot area contents appear invalid, or the source OSF is not section 1 of an *n*-section system.

Errors are also given if:
fromvolume and *tovolume* are the same (by name);
*fromvolume*s are mounted out of order;
a specified ordinary file is inaccessible or has invalid contents;
the first segment is not a power-up segment;
any segment has a mismatching system type.

See *osmgr*(8) for the exact list of return values.

BUGS

Oscp –a checks that all appended segments are mutually compatible, but it does not check them against the segments in the existing OSF.

Performing an *oscp –a* to a boot area with less than 1024 free bytes results in an error before the copy completes.

Before appending, *oscp –s* backspaces over the existing two-null-byte terminator at the end of each ordinary file, but it does not check that the bytes overwritten were actually two null bytes.

A boot area of less than 1024 bytes, at the end of a volume, results in a read error.

NAME

osmark – mark SDF OS file as loadable/unloadable

SYNOPSIS

/etc/osmark [**–m** | **–u**] [**–v**] volume

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Osmark* is implemented on the Series 500/600/700 only.

DESCRIPTION

Osmark marks an operating system file (OSF) in a boot area as loadable (**–m** option) or unloadable (**–u** option). *Volume* is usually a special file specifying the SDF volume on which the boot area is found.

If neither **–m** nor **–u** are specified, *osmark* reports the status of the OSF.

The **–v** (verbose) option causes *osmark* to print additional information about the volume in the same format as that used by *osck* and *oscp*.

When dealing with a multi-volume operating system, be sure that each OSF in the system is properly marked, not just the first.

SEE ALSO

osck(8), oscp(8), osmgr(8).

DIAGNOSTICS

Osmark outputs an appropriate error message and returns a non-zero value if *filespec* cannot be accessed or is not an SDF volume, there is no boot area, or the boot area contents appear invalid. Refer to *osmgr*(8) for a list of possible return values.

NAME

osmgr – operating system manager package description

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	This entry describes the operating system manager package, which is implemented on the Series 500/600/700 only.

DESCRIPTION

This group of three commands helps you manage the operating systems which reside in the boot areas on your Structured Directory Format (SDF) volumes. The package includes:

oscp	copy systems or create them from ordinary files;
osck	check operating system integrity;
osmark	mark an operating system file as loadable or not loadable, or inquire about current state of operating system file.

Oscp, *osck*, and *osmark* are multiple links to a single program.

Boot Areas:

Each SDF volume has one boot area consisting of zero or more contiguous logical blocks. The boot area is completely outside the file area. Its size is determined when the volume is initialized. To change the size of a boot area, you must re-initialize the volume.

Each boot area may contain at most (one part of) one operating system.

The logical block size for a boot area is the same as that for the rest of the volume (i.e., whatever size you request when you initialize the volume).

Operating Systems:

Every HP 9000 operating system consists of a series of code segments. An operating system may reside in the boot area on one volume, or it may be distributed in sections over several volumes (not necessarily with a whole number of segments per volume).

An operating system can also reside in a number of ordinary files, each containing a whole number of segments, and terminated by two null bytes. This is the same format used for BASIC/9000 BIN files. In this form, the system is not loadable, but its files can be combined into a loadable system by *oscp.*

Operating System Files:

Each boot area contains zero or one operating system files (OSF's). If an operating system resides in sections in several boot areas, each section occupies one OSF on one SDF volume.

Operating System File Headers:

Each OSF starts with a header that includes a "loadable" flag, a volume number, and the total number of volumes over which this operating system is distributed. The loader only boots an OSF if it is marked loadable. If required, it requests additional volumes until it has loaded from all volumes in the set. You should ensure that all parts of a multi-volume operating system are marked loadable.

Each OSF header also includes an 80-character identification string. The loader displays this string before it starts to load from each volume.

RETURN VALUES

The following list contains all the possible return values, mnemonics, and meanings given by OS manager commands:

0		no error;
1	USAGE	bad argument list;
2	FILESYS	error during file system access;
3	VOLSEQ	volumes mounted out of order;
4	VOLCONT	bad volume (not SDF, no boot area, etc.);

5	HEADER	invalid or inconsistent OSF header(s);
6	FIRSTSEG	first segment is not a power-up segment;
7	SEGTYPE	incompatible segment system types or revisions;
8	SEGLEN	segment length out of range or not whole words;
9	CHECKSUM	segment checksum does not match reference value;
10	TERM	system terminator ("−1 " word) missing.

SEE ALSO

osck(8), oscp(8), osmark(8), sdfinit(8).

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

pwck [file]
grpck [file]

HP-UX COMPATIBILITY

Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION

Pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The default password file is **/etc/passwd**.

Grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(5), passwd(5).

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

rc – system initialization shell script

SYNOPSIS

/etc/rc arg1 arg2 arg3

HP-UX COMPATIBILITY

Level:	HP-UX/RUN ONLY
Origin:	System III

DESCRIPTION

The **/etc/rc** shell script is executed by *init*(8) whenever the *init* state is changed. *Arg1* is the new state of *init*, *arg2* is the number of times that state has been previously entered, and *arg3* is the previous state (for example, 1 0 0 for system boot-up). These arguments are supplied by *init*.

Rc performs housekeeping functions, such as setting the hostname of your computer, creating the *mnttab* table, mounting volumes, starting *cron*, preserving editor temporary files, and checking for the existence of any temporary files. Much of *rc* is only executed the first time (for *arg1* = 1, *arg3* = 0). Comments are inserted in the shell script to guide you in "customizing" *rc* for your particular system.

SEE ALSO

init(8).

NAME

revck – check internal revision numbers of HP-UX files

SYNOPSIS

/**lbin**/**revck** ref_files

HP-UX COMPATIBILITY

Level: HP-UX/STANDARD

Origin: HP

Remarks: *Revck* is implemented on the Series 500/600/700 only.

DESCRIPTION

Revck checks the internal revision numbers of lists of files against reference lists. Each *ref_file* must contain a list of absolute path names (each beginning with "/") and *whatstrings* (revision information strings from *what*(1)). Path names begin in column one of a line, and have a colon appended to them. Each path name is followed by zero or more lines of *whatstrings*, one per line, each indented by at least one tab (this is the same format in which *what*(1) outputs its results).

For each path name, *revck* checks that the file exists, and that executing *what*(1) on the current path name produces results identical to the *whatstrings* in the reference file. Only the first 1024 bytes of *whatstrings* are checked.

Ref_files are usually the absolute path names of the *revlist* files shipped with HP-UX. Each HP-UX software product includes a file named /system/*product*/revlist (for example, /system/97070A/revlist). The *revlist* file for each product is a reference list for the ordinary files shipped with the product, plus any empty directories on which the product depends.

FILES

/system/*product*/revlist	lists of HP-UX files and revision numbers

SEE ALSO

what(1).

DIAGNOSTICS

Revck is silent except for reporting missing files or mismatches. If a *ref_file* is not in the right format, you will get unpredictable results.

NAME

rootmark – mark/unmark volume as HP-UX root volume

SYNOPSIS

/lbin/rootmark [**-m** | **-u**] filespec

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Rootmark* is implemented on the Series 500/600/700 only.

DESCRIPTION

Rootmark enables you to control which mass storage device contains your HP-UX root (/) directory. The HP-UX operating system searches mass storage devices and uses the first root volume it finds.

Filespec is usually a character special file which points to a mass storage volume initialized with Structured Directory Format (SDF). If invoked with no option, *rootmark* tells the current state of the specified volume. If **–m** is specified, then the specified volume is marked as a root volume. If **–u** is specified, the specified volume is marked as not a root volume. *Rootmark* is silent if successful.

RETURN VALUE

Rootmark sends an error message to standard error and returns a non-zero value if it cannot read or write a volume, or if a volume is not SDF. *Rootmark* returns 1 for incorrect syntax, 2 for a file system problem, and 3 for a volume that is not in SDF.

SEE ALSO

mount(1), osmgr(8), sdfinit(8).

WARNINGS

A volume must not be marked as a root volume unless it contains all the directories and files that HP-UX requires for system initialization.

Never mark any media shipped from Hewlett-Packard as not a root volume, in case you need to reinstall HP-UX from that media.

NAME

sdfinit – initialize Structured Directory Format volume

SYNOPSIS

/lbin/sdfinit [**–i**] pathname [blocksize [bootsize [interleave]]]

HP-UX COMPATIBILITY

Level:	Structured Directory Format - HP-UX/NUCLEUS
Origin:	HP
Remarks:	*Sdfinit* is implemented on the Series 500/600/700 only.

DESCRIPTION

Sdfinit initializes a volume on a special file in the Structured Directory Format (SDF).

Pathname refers to a character or block special file, which must be accessible and not mounted.

Blocksize is the number of bytes per logical block. It is rounded up, if necessary, to the next multiple of the physical record size for the volume. If absent or less than 1, the system sets a reasonable default for you.

Bootsize is the number of bytes to allocate for the boot area on the volume. It is rounded up to a whole number of logical blocks. It defaults to 0 (no boot area).

Interleave is the sector interleave factor. It defaults to 1 (not necessarily the best value for all devices).

The **–i** option inhibits formatting and certification, so the volume is only initialized. That is, only a directory structure is written. This saves a considerable amount of time in most cases. However, the **–i** option is **not recommended** for most removable media, unless it was recently formatted and certified in the same type of drive.

Sdfinit does not return until the operation is completed. This can require considerable time. For example, formatting can take 47 minutes on an HP 7933 hard disc, or up to 67 minutes on an HP 88140L cartridge tape. Initialization takes an additional one to five minutes.

The root directory on the newly initialized volume is always owned by the super-user, and has a mode of 777.

Note that your effective user ID must be that of the super-user (0).

HARDWARE DEPENDENCIES

Series 500/600/700:

During the formatting and initialization process, you are prevented from doing anything else on the same select code.

SEE ALSO

lifinit(1), section 4, osmgr(8).

DIAGNOSTICS

Appropriate error messages are given if the argument list is incorrect, *pathname* cannot be initialized, or any other error occurs.

WARNING

Aborting *sdfinit* prematurely requires that you re-execute *sdfinit* to format and initialize the media.

Serious problems arise if you power down the HP 9000 Series 500 while *sdfinit* is formatting and initializing the internal 10 megabyte Winchester disc. This is because *sdfinit* writes critical records to the disc that specify what records are spared. When you power down, these tables become garbage, and cause the disc to fail its read/write self-test when you power up again. If this problem occurs, call your local HP Sales and Support Office for assistance.

NAME

setmnt – establish mnttab table

SYNOPSIS

/etc/setmnt

HP-UX COMPATIBILITY

Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION

Setmnt creates the /**etc**/**mnttab** table (see *mnttab*(5)), which is needed for both the *mount*(1) and *umount*(1) commands. *Setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

```
filesys node
```

where *filesys* is the name of the file system's *special file* and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the *mnttab*(5) entry.

FILES

/etc/mnttab

SEE ALSO

mnttab(5).

BUGS

Filesys and *node* are truncated to MNTLEN bytes.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.
It is unwise to use *setmnt* to create false entries for *mount*(1) and *umount*(1).

NAME
shutdown – terminate all processing

SYNOPSIS
/etc/shutdown [grace]

HP-UX COMPATIBILITY
Level:	HP-UX/STANDARD
Origin:	System III

DESCRIPTION
Shutdown is part of the HP-UX operation procedures. Its primary function is to terminate all currently running processes in an orderly and cautious manner. The procedure is designed to interact with the operator (i.e., the person who invoked *shutdown*). *Shutdown* may instruct the operator to perform some specific tasks, or to supply certain responses before execution can resume. *Shutdown* goes through the following steps:

- All users logged on the system are notified to log off the system by a broadcasted message. The operator may display his/her own message at this time. Otherwise, the standard file save message is displayed.
- If the operator wishes to run the file-save procedure, *shutdown* unmounts all file systems.
- All file systems' super blocks are updated before the system is to be stopped (see *sync*(8)). This must be done before re-booting the system, to insure file system integrity. The most common error diagnostic that will occur is *device busy*. This diagnostic happens when a particular file system could not be unmounted. See *umount*(1).

Grace specifies, in seconds, a grace period for users to log off before shutting down. The default is 60 seconds. If *grace* is zero, *shutdown* runs more quickly and gives the user very little time to log out.

HARDWARE DEPENDENCIES
Series 500/600/700:
A file-save procedure is not implemented.

SEE ALSO
mount(1), sync(8).

NAME

stopsys – stop operating system with optional reboot

SYNOPSIS

/lbin/stopsys [**–r**]

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Stopsys* is implemented on the Series 500/600/700 only.

DESCRIPTION

Stopsys dumps all system I/O buffers to mass storage volumes (i.e. performs a *sync*(8)), and shuts down all virtual memory activity. Then, *stopsys* either stops the operating system so that the hardware may be powered down (no option), or it reboots the system (resets the machine's processor(s) to the power-on state) (**–r** option). The reboot (**–r**) option results in the activation of the system boot loader, almost exactly as if the power was just turned on, except that I/O cards are not power-cycled.

Just before it stops the system, *stopsys* writes a message to /dev/console indicating that the system is stopped and can be safely powered down.

Stopsys may be invoked only by the effective super-user. However, it may be made public by setting the set-user-ID bit and assigning ownership to *root.*

Stopsys does not ensure that the system is idle. If any user processes are running, the *sync*(8) may be ineffective. You should execute *shutdown*(8), or at least kill all non-essential processes, prior to running *stopsys.*

SEE ALSO

chsys(8), killall(8), shutdown(8), sync(8).

DIAGNOSTICS

Stopsys returns only if a non-fatal error occurs, in which case it writes a message to standard error and returns 1. Non-fatal errors include:

invocation with improper arguments;
invocation by other than the effective super-user;
any failure to stop the system, as long as the system is still usable.

If *stopsys* fails to stop the system for any reason, but the system is then not in a usable state, *stopsys* writes an error message to /dev/console and then attempts to reboot (if **–r** was specified). If **–r** was not specified, or if the reboot attempt fails, *stopsys* writes "system stopped" on /dev/console, and you must reboot the system yourself (using the power switch or the front panel).

Note that if the reboot fails it indicates a hardware problem with the HP 9000 Model 20 keyboard on select code 6, or the HP 9000 Model 30/40 system control module on select code 7.

BUGS

At this time, *stopsys* does not shut down Local Area Net (LAN) activity.

NAME
sync – update the super block

SYNOPSIS
sync

HP-UX COMPATIBILITY
Level:	HP-UX/NUCLEUS
Origin:	System III

DESCRIPTION
Sync executes the *sync* system intrinsic. If the system is to be stopped, *sync* must be called to insure file system integrity. See *sync*(2) for details.

SEE ALSO
sync(2).

NAME

uconfig – system reconfiguration

SYNOPSIS

/lbin/uconfig [option boot_device]

HP-UX COMPATIBILITY

Level:	HP-UX/NON-STANDARD
Origin:	HP
Remarks:	*Uconfig* is implemented on the Series 500/600/700 only.

DESCRIPTION

Uconfig enables you to reconfigure certain system parameters. When invoked with no arguments, *uconfig* lists the current system configuration. The following *options* are recognized:

–f file reconfigures the system parameters in the boot area according to the specifications given in *file*. *File* may contain any combination of system parameters. Each line in *file* has the following format:

id value [#comment]

where *id* is a pre-defined system parameter name, *value* is one or more values associated with the parameter, and *comment* is a descriptive comment for that line. All characters between the comment delimiter (#) and a new-line are ignored. The *id*, *value*, and *comment* fields are delimited by one or more blanks and/or tabs.

The valid *id*s and *value*s are:

vm_device driver_name addr1 addr2 addr3 addr4
where *driver_name* is an integer specifying the virtual device driver, and *addr1* – *addr4* are integers specifying the device select code, HP-IB address, unit, and volume, respectively.

cache_buf_size size
where *size* is an integer in the range 256 to (maximum memory) divided by (minimum number of cache buffers), specifying the number of bytes in each individual cache buffer. *Size* is rounded down to the closest multiple of 256.

cache_buf_num num
where *num* is an integer in the range 1 to (maximum memory) divided by (minimum size of cache buffers), specifying the number of individual cache buffers forming the cache.

read_ahead_level level
where *level* is an integer in the range 1 to the value of **cache_buf_num**, specifying the number of buffers that can be filled in one sequential read operation.

swap_time time
where *time* is an integer in the range of 1 to 32767 ticks (a tick equals 10msec), specifying the time a virtual segment remains memory resident before being swapped to disc.

page_size size
where *size* is an integer in the range 256 to 8192, specifying the size of paged data in bytes. If *size* is an odd number, it is rounded down to the next even number.

vm_pool_size size
where *size* is an integer in the range 16384 to maximum memory, specifying the maximum size in bytes of the virtual memory page pool. *Size* is rounded down to the nearest 16 Kbyte boundary.

scroll_pages num_pages
where *num_pages* is an integer in the range 1 to 10, specifying the number of pages of display buffering (one page = 24 lines of display). The actual number of pages allocated depends on current available memory. This parameter applies to the Model 20 only.

local_time_zone time_zone
where *time_zone* is an integer in the range –1440 to 1440, specifying the local time zone in minutes relative to GMT.

stack_size size
where *size* is an integer in the range 16384 to maximum memory, specifying the maximum stack size in bytes for any partition.

segment_num num_segments
where *num_segments* is an integer in the range 200 to 950, specifying the maximum number of code and data segments in the system.

work_set_ratio ratio
where *ratio* is a floating-point number in the range 0 to 1, specifying the virtual memory working set ratio.

–d reconfigures the system parameters in the boot area to their default values. The default values, as contained in the file /etc/**uconfigtab**, are:

vm_device	0 0 0 0 0; root device as determined by the system at power-up;
cache_buf_size	1024 bytes;
cache_buf_num	50;
read_ahead_level	4;
swap_time	20 ticks; (one tick = 10 msec)
page_size	4096 bytes;
vm_pool_size	0; this value is dynamically computed;
scroll_pages	2;
local_time_zone	420 (Fort Collins time zone);
stack_size	2097152;
segment_num	700;
work_set_ratio	0.00375.

The **–f** and **–d** options are mutually exclusive.

Boot_device is the path name of a character special file containing a boot area. The new configuration is written out to the boot area on *boot_device*, and takes effect the next time the system is booted.

FILES

/etc/uconfigtab
list of default system configuration parameters

WARNING

Do not use *uconfig* to change the system parameters of an operating system in a boot area unless that operating system is identical to the operating system you are currently running. If the two operating systems differ, *uconfig* will execute successfully, but the new operating system will either fail to boot, or, if it boots successfully, exhibit strange behavior.

absolute path name	path name beginning with a slash. It indicates that the file's location is given relative to the current root directory, and that the search begins there.
a.out	the format of object code files on HP-UX. The format is machine dependent. *A.out* is also the default output file name for *ld*(1).
archive	a file which is made up of the contents of other files (such as a group of library files to be used by the linker *ld*(1)). An archive file is created and maintained by *ar*(1). Similar programs (*tar*(1) or *cpio*(1)) create archives also, but not in a form compatible with *ld*(1).
block size	see *logical block size.*
block special file	a special file associated with discs and other volumes (such as CS80 tape cartridges) that transfer data in units of blocks. Block special files are mountable.
boot	the process of searching for, loading, and running an operating system.
boot area	the portion of an SDF mass storage medium which contains an executable module which boots the operating system in some implementations.
boot ROM	see *system loader.*
character special file	a special file associated with devices which transfer data character-by-character. Examples are printers, terminals, nine-track magnetic tapes, and discs accessed in raw mode.
child	a process spawned via the *fork*(2) system call.
command	a program which is executed through the shell command interpreter. Arguments following the command name are passed on to the command program. You can write your own command programs, either as compiled programs, or as shell scripts (written in the shell programming language).
command interpreter	a program which reads lines typed at the keyboard (or redirected from a file), and interprets them as requests to execute other programs. The command interpreter for HP-UX is called the shell. See *sh*(1), *rsh*(1).
control character	a member of a character set which may produce some action in a device other than printing or displaying a character. In ASCII, control characters are those in the code range 0 through 31, and 127. Control characters can be generated by simultaneously pressing a displayable character key and [CTRL], [CONTROL], or [CNTL].
crash	the unexpected shutdown of a system, usually requiring the system to be rebooted.
current directory	see *working directory.*
current working directory	see *working directory.*
data encryption	a method for encoding information in order to protect sensitive or proprietary data.
default search path	the sequence of directory prefixes that *sh*(1), *time*(1), and other HP-UX commands apply in searching for a file known by an incomplete path name. It is defined by the environment variable **PATH** (see *environ*(7)). *Login*(1) sets **PATH** equal to **:/bin:/usr/bin**, which means that your working directory is the first directory searched, followed by /**bin**, followed by /**usr**/**bin**. You can redefine the search path by modifying **PATH**, usually done in /**etc**/**profile** and in **.profile**.
directory	provides the mapping between the names of files and the files themselves. Each user may have a directory of his own files. An HP-UX directory is named and behaves exactly like an ordinary file, except that it can be written

	on only by the file system.
effective group ID	an active process has an effective group ID that is used to determine file access permissions. The effective group ID is equal to the process's real group ID, unless the process or one of its ancestors evolved from execution of a file that had the set-group-ID bit set. See *exec*(2).
effective user ID	an active process has an effective user ID that is used to determine file access permissions (and other permissions with respect to system calls, if the effective user ID is 0). The effective user ID is equal to the process's real user ID, unless the process or one of its ancestors evolved from execution of a file that had the set-user-ID bit set. See *exec*(2).
environment	the set of conditions (such as your working directory, home directory, and the type of terminal you are using) that exist for any given process.
end-of-file character	(1) the character returned when attempting to read past the logical end of a file (via *stdio*); (2) the character which indicates end of data when using *read*(2). An end-of-file indication is entered from the keyboard by typing control-d.
file	an HP-UX file is simply a stream of bytes – there is no system-defined record structure. Files may be accessed serially or randomly (indexed by byte offset). The interpretation of file contents and structure is up to the programs that access the file. Text files are sequences of ASCII characters.
file descriptor	a unique identifier of an open file, returned by the file system when a file is opened for reading or writing. The opened file must be identified by its descriptor when using system calls to read or write the file.
file name	names consisting of up to 14 characters may be used to name an ordinary file, special file, or directory. These characters may be selected from the set of all character values excluding null and the ASCII code for / (slash). Note that it is generally unwise to use "*", "?", "[", "!", or "]" as part of file names because of the special meaning attached to these characters by the shell. See *sh*(1). It is also not wise to begin a file name with a "–", " + ", or " = ", since these symbols are used to indicate that a command argument follows. Beware of putting blanks in quoted file names.
file system	the directory structure and all associated files.
filter	a program which reads data from the standard input, performs a transformation on the data, and writes it to the standard output.
fork	an HP-UX system call which spawns a new process as an exact copy of the current process. This is the only means by which new processes are created in HP-UX. See *fork*(2).
group	a distinction given to any number of users who may be permitted to access the same set of files. The users in any given group are listed in the "group file", */etc/group*.
hierarchical directory	a directory structure in which each directory may contain other directories as well as files.
home directory	the directory defined by the shell variable **HOME**. You are placed in the

	home directory when you log in. **HOME** defaults to your login directory unless you specify otherwise.
interleave factor	a number which determines the order in which disc sectors on a disc are accessed to make data acquisition more efficient.
interrupt signal	the signal sent by *SIGINT*. It is the ASCII DEL ("rubout") character or the BREAK key. This signal generally causes whatever program you are running to terminate. The key which sends this signal (see *signal*(2)) can be redefined via *ioctl*(2) or *stty*(1).
intrinsic	see *system call.*
I/O redirection	a mechanism provided by the HP-UX shell for changing the source of standard input data and the destination of standard output and standard error.
kernel	the most basic part of the HP-UX operating system. The kernel supports the file system, as well as task synchronization, scheduling, communication, I/O, process creation, and memory allocation activities (everything that is included in Section 2).
library	an archive file containing a set of subroutines and variables which may be accessed by a user program.
LIF	an abbreviation for Logical Interchange Format.
link	a directory entry for a file. One physical file may have several links – it may appear in several different directories (possibly under different names).
linker	combines several object programs into one, and searches libraries to resolve user program references, building a file for the loader (*exec*(2)) in *a.out* format. The *ld* (link editor) command performs this function.
logical block size	the smallest unit of memory which can be allocated on an SDF volume; a multiple of the physical sector size.
login directory	the directory which becomes the defualt directory when you first log in. It is specified in */etc/passwd*, on the line beginning with your user name. It is usually the same as that specified in the environment variable **HOME**.
login procedure	the procedure required before a user can begin using an HP-UX system. He must enter his user name and his password (if any).
magic number	the first word of an *a.out*–format or archive file. It contains the system ID (tells what machine the file will run on) and the file type.
metacharacter	a character which has special meaning to the HP-UX shell. The set of metacharacters includes: * ? ! [ ] < > ; | ' " &.
mode	the 16 bits which determine access and execute permission for a file, determine whether to save text image after execution, set effective user and group IDs, and indicate file type.
mountable file system	a removable file system with its own root directory and independent hierarchy of directories and files.
new-line	the character which usually separates lines of characters. It is the same character as line-feed: ASCII (octal) code 12. It is represented by "\n" in many utilities and in C. The terminal driver (see *tty*(4)) normally recognizes the carriage-return/line-feed sent by a terminal as the new-line character, and translates it accordingly.
ordinary file	a type of HP-UX file, created by the user, and containing a program or text.
orphan process	When a parent process is terminated via *exit*(2), its child processes are also terminated. However, any processes that may have been spawned by the

	child processes are not killed and become "orphans". *Init*(8) inherits all orphan processes.
OSF	an operating system file in SDF boot area.
owner	the owner of a file is usually the creator of that file. However, the ownership of a file can be changed by the super-user or the current owner with the *chown* command.
parent directory	a directory's parent directory is the directory one level above it in the file hierarchy.
parent process ID	a new process is created by a currently active process; see *fork*(2). The parent process ID of a process is the process ID of its creator.
password	a string of characters used to verify the identity of a user. Passwords can be associated with users and groups. They can be encrypted, and they appear in files in encrypted form (see the second field of each line in */etc/passwd*).
path name	a path name is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a file name. A slash by itself names the root directory. Unless specifically stated otherwise, the null path name is treated as if it named a non-existent file.
permission bits	the nine low-order bits of a file's mode. These bits determine read, write, and execute permissions for the file owner, group, and others.
pipe	an interprocess communication channel used to pass data between two processes. It is commonly used by the shell to transfer data from the standard output of one process to the standard input of another.
proc1	a special process with a process ID of 1. This process is the initialization process (*init*). It is the ancestor of every other process in the system and is used to control the process structure.
process	an invocation of a program. There may be several processes, all running the same program, but with different data and in different stages of execution.
process group ID	each active process is a member of a process group that is identified by a positive integer called the *process group ID*. This ID is the process ID of the group leader. This grouping permits the signalling of related processes; see *kill*(2).
process ID	each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 0 to 30 000.
program	a sequence of instructions to the computer, either in the form of a compiled high-level language or a sequence of shell command language instructions in a text file.
prompt	the character displayed by the shell on the CRT indicating that the previous command has been completed and the system is ready for another command. It is usually a "$" or "#", but the user can re-define it to be any string by setting **PS1** in his *.profile* file.
quit signal	the signal sent by *SIGQUIT*. See *signal*(2). The quit signal is generated by typing the ASCII FS character. It not only causes a running program to terminate, but also generates a file with the "core image" of the terminated process. Quit is useful for debugging.
randomized library	an archive file as processed by *ranlib*(1). It contains a table of contents of

	externally known symbols as its first element. See *ranlib*(1) and *ld*(1).
raw disc	a character special file which provides for direct transmission between the disc and the user's read or write buffer. A single read or write call results in exactly one I/O call.
real group ID	each user allowed on the system is a member of a group. The group is identified by a positive integer called the *real group ID*. An active process has a real group ID set to the real group ID of the user responsible for the creation of the process.
real user ID	each user allowed on the system is identified by a positive integer called a *real user ID*. An active process has a real user ID set to the real user ID of the user responsible for the creation of the process.
regular expression	used within an editor as a short-cut for specifying a set of character strings. This notation is used in addresses to specify lines, and in some editor commands to specify portions of a line that are to be substituted. For some examples of commands that use regular expressions, see *grep*(1), *ed*(1), and *sh*(1).
relative path name	path name not beginning with a slash. It indicates that the file's location is given relative to the current working directory, and that the search begins there.
root directory	the highest level directory of the hierarchical file system, in which other directories are contained. In HP-UX, the "/" character refers to the root directory.
root volume	the mass storage volume which includes the root directory.
SDF	an abbreviation for Structured Directory Format. SDF provides tree-structured access to files through the root directory of the volume.
shell	the shell is a command language, a command interpreter, and a programming language that provides the user-interface to the HP-UX operating system.
shell program	see *shell script*.
shell script	a sequence of shell commands and shell programming language constructs, usually stored in a file, for invocation as a user command (program) by the shell.
signal	signals are software interrupts sent to processes, informing them of special situations or events. HP-UX currently defines 19 types of signals. See *signal*(2).
special file	a file associated with an I/O device. Special files are read and written just like ordinary files, but requests to read or write result in activation of the associated device. Entries for each file normally reside in the */dev* directory.
standard error	file descriptor 2 – the default destination of error and special messages from a program if a destination file is not specified. The default standard error is the user's login device, but it is possible to redirect it to another destination.
standard input	file descriptor 0 – the default source of input data for a program if a source file is not specified. The default standard input is the user's login device, but the parent process may redirect the standard input to be from a file or a pipe.
standard output	file descriptor 1 – the default destination of output data from a program, if a destination file is not specified. The default standard output is the user's login device, but the parent process may redirect the standard output to be a file or

	a pipe.
super block	a block on each file system storage volume which describes the volume. It includes such items as the size of the entire volume, the logical block size, and the locations of the file attributes file and the boot area. See *fs*(5).
super-user	the HP-UX system manager. This user has access to all files, and can perform privileged operations. He has a user ID of 0. The super-user can read or write any file without regard to the setting of the mode of the file.
system call	an HP-UX operating system (kernel) function available to the user through a high-level language (such as FORTRAN, Pascal, or C). Also called an "intrinsic" or a "system intrinsic".
system loader	a piece of object code that permanently resides in the computer. When the computer is powered up, the system loader is automatically loaded and run. Its function is to find and load an operating system. In the Series 210, when this object code is contained in a ROM (Read Only Memory) chip, it is called a "boot ROM".
tty group ID	each active process can be a member of a terminal group that is identified by a positive integer called the *tty group ID*. This grouping is used to terminate a group of related processes upon termination of one of the processes in the group; see *exit*(2) and *signal*(2).
working directory	the directory in which you currently reside. Also, the default directory in which path name searches begin, when a given path name does not begin with "/". (It is sometimes referred to as the "current directory", or the "current working directory".)
zombie process	if the parent process of a calling process is not executing a *wait*, the calling process is transformed into a *zombie process*. This is a process that only occupies a slot in the process table; it has no other space allocated either in user or kernel space.

Manual Comment Sheet Instruction

If you have any comments or questions regarding this manual, write them on the enclosed comment sheets and place them in the mail. Include page numbers with your comments wherever possible.

If there is a revision number, (found on the Printing History page), include it on the comment sheet. Also include a return address so that we can respond as soon as possible.

The sheets are designed to be folded into thirds along the dotted lines and taped closed. Do not use staples.

Thank you for your time and interest.

MANUAL COMMENT SHEET

HP-UX Reference

09000-90004 June 1983

Update No. ________________

(See the Printing History in the front of the manual)

Name: ________________________________

Company: ________________________________

Address: ________________________________

____

Phone No: ________________

fold - fold

fold - fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Engineering Systems Division
Attn: Cust. Documentation/Dept. 4231
3404 East Harmony Road
Fort Collins, Colorado 80525

MANUAL COMMENT SHEET

HP-UX Reference

09000-90004 June 1983

Update No. ________________

(See the Printing History in the front of the manual)

Name: ______________________________

Company: ______________________________

Address: ______________________________

____

Phone No: ________________

fold - fold

fold - fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Engineering Systems Division
Attn: Cust. Documentation/Dept. 4231
3404 East Harmony Road
Fort Collins, Colorado 80525

MANUAL COMMENT SHEET

HP-UX Reference

09000-90004 June 1983

Update No. ________

(See the Printing History in the front of the manual)

Name: ________

Company: ________

Address: ________

____

Phone No: ________

fold -- fold

fold -- fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Engineering Systems Division
Attn: Cust. Documentation/Dept. 4231
3404 East Harmony Road
Fort Collins, Colorado 80525

MANUAL COMMENT SHEET

HP-UX Reference

09000-90004 June 1983

Update No. ____________

(See the Printing History in the front of the manual)

Name:	
Company:	
Address:	
Phone No:	

fold - fold

fold - fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS   PERMIT NO. 37   LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Engineering Systems Division
Attn: Cust. Documentation/Dept. 4231
3404 East Harmony Road
Fort Collins, Colorado 80525

MANUAL COMMENT SHEET

HP-UX Reference

09000-90004 June 1983

Update No. ________________

(See the Printing History in the front of the manual)

Name: ____

Company: ____

Address: ____

____

Phone No: ____________________

fold - fold

fold - fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Engineering Systems Division
Attn: Cust. Documentation/Dept. 4231
3404 East Harmony Road
Fort Collins, Colorado 80525

HEWLETT
PACKARD

Part No. 09000-90004
E0683
Microfiche No. 09000-99004

Printed in U.S.A.
First Edition, June 83