

HP 9000 Series Computer Systems

HP-UX Reference
09000-90004

Computer
Museum

Preliminary
November, 1982

Hewlett-Packard Engineering Systems Division

3404 East Harmony Road, Fort Collins, Colorado 80525

Copyright © 1980 Bell Telephone Laboratories, Inc.

UNIX is a Trademark of Bell Telephone Laboratories, Inc.

©Copyright Hewlett-Packard Company, 1982

This document refers to proprietary computer software which is protected by copyright. All rights are reserved. Copying or other reproduction of this program except for archival purposes is prohibited without the prior written consent of Hewlett-Packard Company.

Not for use or disclosure except within Hewlett-Packard Company or by HP-UX licensees.

Hewlett-Packard Engineering Systems Division

3404 East Harmony Road, Fort Collins, Colorado 80525

INTRODUCTION

This manual describes the features of HP-UX. It provides neither a general overview of HP-UX (for that, see A User Guide to the UNIX System, by Rebecca Thomas and Jean Yates), nor details of the implementation of the system (see HP-UX System Manager's Manual).

Not all commands, features, and facilities described in this manual are available in every HP-UX system. When in doubt, consult your system's administrator.

This manual is divided into nine sections, some containing inter-filed sub-classes:

1. Commands and Application Programs:
 1. General-Purpose Commands.
 - 1C. Communications Commands.
 - 1G. Graphics Commands.
 - 1M. System Maintenance Commands.
2. System Calls.
3. Subroutines:
 - 3C. C and Assembler 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 subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory `/bin` (for binary programs). Some programs also reside in `/usr/bin`, to save space in `/bin`. These directories are searched automatically by the command interpreter called the `shell`. Sub-class 1C contains communication programs such as `cu`, `dpr`, `fget`, etc. These entries may differ from system to system. Sub-class 1M contains system maintenance programs such as `fsck`, `mkfs`, etc., which generally reside in the directory `/etc`; these commands are not intended for use by the ordinary user due to their privileged nature.

Section 2 (System Calls) describes the entries into the HP-UX supervisor, 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.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Section 4 (Special Files) discusses the characteristics of each system file 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`.

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

Section 8 (System Maintenance Procedures) discusses crash recovery and boot procedures, etc. Information in this section is not of great interest to most users.

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

Each section 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.

Change bars in the margin indicate changes or additions to the "UNIX User's Manual" from Bell Laboratories. An asterisk in the margin indicates a deletion of text from the "UNIX User's Manual".

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 program being described. A few conventions are used, particularly in Section 1 (Commands):

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

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).

Square brackets **[]** around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a file name.

—Ellipses (...) are used to show that the previous argument prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus -, plus +, or 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.

DESCRIPTION

discusses the subject at hand.

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.

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 derived from that table 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 A User Guide to the UNIX System by Rebecca Thomas and 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 administrator of your system. Keep hitting the "break" or "attention" key until the **login:** message appears.

When a connection (at the speed of the terminal) has been established, the system types **login:** and you then type your user name followed by the return key. If you have a password (and you should!), the system asks for it, but does not print (echo) it on the terminal. After you have logged in, the return, new-line, and line-feed keys will give exactly the same result.

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 you mean all subsequent upper-case input to be treated as lower case. When you have logged in successfully, the shell will type a \$ to you. (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, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a return (or new-line), as described above in Logging in.

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 typing at you. Of course, if you type during output, the output will have interspersed in it the input characters. 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; see etty(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 a DC1 (control-q) or a second DC3 (or any other character, for that matter) is typed. The DC1 and DC3 characters are not passed to any other program when used in this manner.

The ASCII DEL (a.k.a. rubout) character 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 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.

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 function, 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, the etty(1) command will rescue you.

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. Again, the etty(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 a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see The current directory below) for a program with the given name, and if none is there, then in system

directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

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

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

The current directory. HP-UX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as 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 by default assumed to be in that directory. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their respective owners, or by the system administrator. To change the current 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 subdirectory (followed by a /), until finally the file name is reached (e.g., /usr/ae/filex refers to file filex in directory ae, while ae is itself a subdirectory of usr; usr springs directly from the root directory). See intro(2) 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)`. 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 `my(1)` to give it a less vulnerable name). If the program is written in assembly language, you will probably need to load with it library subroutines (see `ld(1)`). Fortran and C call the loader automatically.

When you have finally gone through this entire process without provoking 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; see `exec(2)`.

Text processing. Almost all text is entered through the editor `ed(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 aim them at 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 `$`.

TABLE OF CONTENTS



1. Commands

ar.....	archive and library maintainer
at.....	schedule and execute commands at a later time
basename.....	extract portions of path names
cat.....	concatenate, copy, and print files
cc.....	C compiler
cd.....	change working directory
chmod.....	change mode
chown.....	change file owner or group
chroot.....	change root directory for a command
cmp.....	compare two files
cp.....	copy, link or move files
cpboot.....	copy boot area (operating system)
cpio.....	copy file archives in and out
cron.....	clock daemon
cu.....	call another UNIX system
date.....	print and set the date
df.....	report number of free disk blocks
diff.....	differential file comparator
du.....	summarize disk usage
echo.....	echo (print) arguments
ed.....	text editor
ex.....	text editor
expr.....	evaluate arguments as an expression
find.....	find files
fsck.....	file system consistency check and interactive repair
getopt.....	parse command options
grep.....	search a text file for a pattern
id.....	print user and group IDs and names
install.....	install commands
intro.....	introduction to commands and application programs
kill.....	terminate a process
ld.....	link editor
lifcp.....	copy to or from LIF files
lifinit.....	write a LIF volume header on the file
lifls.....	list content of a LIF directory
lifrn.....	rename LIF files
lifrm.....	remove a LIF file
link.....	exercise link and unlink system calls
login.....	sign on
logname.....	get login name
lpd.....	line printer daemon
lpr.....	line printer spooler
ls.....	list contents of directories
lsdev.....	list device drivers in the system
mail.....	send mail to users or read mail
make.....	maintain, update, and regenerate (compile) groups of programs
msg.....	permit or deny messages to terminal
mkdir.....	make a directory
mknod.....	create special, fifo file

Table of Contents

mount.....	mount and dismount file system
mvdir.....	move a directory
newgrp.....	log in to a new group
news.....	print news items
nice.....	run a command at low priority
nm.....	print name list (symbol table) of object file
od.....	octal file dump
passwd.....	change login password
pr.....	print and paginate files, make columnar listings
ps.....	report process status
pwd.....	working directory name
ranlib.....	convert archives to random libraries
revision.....	get HP-UX revision information
rm.....	remove files or directories
sdfinit.....	initialize Structured Directory Format volume
sed.....	stream text editor
setmnt.....	establish mnttab table
sh.....	shell, the standard command programming language
shutdown.....	terminate all processing
sleep.....	suspend execution for an interval
sort.....	sort and/or merge files
strip.....	remove symbols and relocation bits
stty.....	set the options for a terminal port
su.....	substitute or change users; become super-user or another user
sync.....	update the super block
tabs.....	set tabs on a terminal
tar.....	tape file archiver
tee.....	pipe fitting
test.....	condition evaluation command
time.....	time a command
touch.....	update access, modification times of file; create empty file
true.....	provide truth values
tty.....	get the terminal's name
umask.....	set file-creation mode mask
uname.....	print name of current HP-UX version
uuclean.....	uucp spool directory clean-up
uucp.....	unix to unix copy; file transfer
uustat.....	uucp status inquiry and job control
uusub.....	monitor uucp network
uuto.....	public UNIX-to-UNIX file copy
uux.....	unix to unix command execution
vi.....	screen oriented (visual) display editor based on ex
wait.....	await completion of process
wall.....	write to all users
wc.....	word, line, and character count
what.....	identify files for SCCS information.
who.....	which users are on the system
whodo.....	which users are doing what
write.....	interactively write (talk) to another user
xref.....	cross reference for C programs

Table of Contents

2. System Calls

access.....determine accessibility of a file
alarm.....set a process's alarm clock; signal on timeout
brk.....change data segment space allocation, memory allocation
chdir.....change working directory
chmod.....change access mode of file
chown.....change owner and group of a file
chroot.....change root directory
close.....close a file descriptor
creat.....create a new file or rewrite an existing one
dup.....duplicate an open file descriptor
exec.....execute a file
exit.....terminate process
fcntl.....file control
fork.....create a new process
getdrv.....get driver name corresponding to major device number
getmaj.....get major device number corresponding to driver
getpid.....get process, process group, and parent process IDs
getuid.....get real/effective user, real/effective group IDs
initv.....initialize and optionally format a file system media
intro.....introduction to system calls and error numbers
ioctl.....control device
kill.....send a signal to a process or a group of processes
link.....link to a file
lseek.....move read/write file pointer; seek
mknod.....make a directory, or a special or ordinary file
mount.....mount a file system
nice.....change priority of a process
open.....open for reading or writing
pause.....suspend process until signal
pipe.....create an interprocess channel
read.....read from file
setpgrp.....set process group ID
setuid.....set user and group IDs
signal.....specify what to do upon receipt of a signal
stat.....get file status
stime.....set time and date
stty.....control device
sync.....update super-block
time.....get time
times.....get process and child process times
ulimit.....get and set user limits
umask.....set and get file creation mask
umount.....unmount a file system
uname.....get name of current HP-UX system
unlink.....remove directory entry; delete file
ustat.....get file system statistics

Table of Contents

utime.....set file access and modification times
wait.....wait for child process to stop or terminate
write.....write on a file

3. Subroutines

a64l.....convert between long and base-64 ASCII
abort.....generate an IOT fault
abs.....integer absolute value
assert.....program verification
atof.....convert ASCII to numbers
bessel.....bessel functions
conv.....character translation
crypt.....DES encryption
ctermid.....generate file name for terminal
ctime.....convert date and time to ASCII
ctype.....character classification
cuserid.....character login name of the user
ecvt.....output conversion
end.....last locations in program
exp.....exponential, logarithm, power, square root functions
fclose.....close or flush a stream
ferror.....stream file status inquiries
floor.....absolute value, floor, ceiling, remainder functions
fopen.....open or re-open a stream file; convert file to stream
fread.....buffered binary input/output to a stream file
frexp.....split into mantissa and exponent
fseek.....reposition a stream
gamma.....log gamma function
getc.....get character or word from stream file
getenv.....value for environment name
getgrent.....get group file entry
getlogin.....get login name
getopt.....get option letter from argv
getpass.....read a password
getpw.....get name from UID
getpwent.....get password file entry
gets.....get a string from a stream file
hypot.....Euclidean distance
intro.....introduction to subroutines and libraries
logname.....login name of user
malloc.....main memory allocator
mktemp.....make a unique file name
nlist.....get entries from name list
perror.....system error messages
popen.....initiate pipe I/O to/from a process
printf.....output formatters
putc.....put character or word on a stream
putpwent.....write password file entry

Table of Contents

puts.....	put a string on a stream file
qsort.....	quicker sort
rand.....	random number generator
scanf.....	formatted input conversion, read from stream file
setbuf.....	assign buffering to a stream file
setjmp.....	non-local goto
sinh.....	hyperbolic functions
sleep.....	suspend execution for interval
stdio.....	standard buffered input/output stream file package
string.....	character string operations
swab.....	swap bytes
system.....	issue a shell command
tmpfile.....	create a temporary file
tmpnam.....	create a name for a temporary file
trig.....	trigonometric functions
ttyname.....	find name of a terminal
ungetc.....	push character back into input stream

4. Special Files

console.....	super-user terminal
crt.....	cathode ray tubes for HP 9000 Model 20
cs80.....	Command Set '80 devices
disc.....	direct disc access
intro.....	introduction to special files
magtape.....	magnetic tape interface and controls
null.....	the null file
omni.....	Internal 5 1/4 Inch Flexible Disc Drive
printers.....	HP 2631 line printer
thermal.....	internal thermal printer for HP 9000 Model 20
tty.....	general terminal interface

5. File Formats

a.out.....	assembler and link editor output
ar.....	archive file format
checklist.....	list of file systems processed by fsck
cpio.....	format of cpio archive
dir.....	format of directories
fs.....	format of system volume
fspec.....	format specification in text files
group.....	group file
inittab.....	control information for init
intro.....	introduction to file formats
magic.....	magic numbers for HP-UX implementations
mknod.....	create a special file entry
mnttab.....	mounted file system table
nlist.....	nlist structure format

Table of Contents

passwd.....	password file
profile.....	setting up an environment at login time
sysconf.....	system configuration table file format
utmp.....	Utmp and wtmp entry format

6. Games

No games are currently supported.

7. Miscellaneous Facilities

environ.....	User environment
fcntl.....	file control options
intro.....	introduction to miscellany
regexp.....	regular expression compile and match routines
stat.....	data returned by stat system call
term.....	conventional names
types.....	primitive system data types

8. System Maintenance Procedures

getty.....	set the modes of a terminal
init.....	process control initialization
intro.....	introduction to system maintenance procedures
makekey.....	generate encryption key
mkdev.....	make device files
rc.....	system initialization shell script

NAME

intro - introduction to commands and application programs

DESCRIPTION

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

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

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.

HP-UX COMPATIBILITY

Level: This describes where in the HP-UX compatibility model this capability appears. See _____ for a detailed explanation of the model.

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

<u>S</u> ystem <u>I</u> II	means from Bell UNIX System III.
<u>H</u> P	means written by HP.
<u>U</u> CB	means derived from Berkeley 4.2bsd.
<u>V</u> 7	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.

COMMAND SYNTAX

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

`name [option(s)] [cmdarg(s)]`
 where:

`name` The name of an executable file.

`option` - `noargletter(s)` or,
 - `argletter(<)optarg`
 where < > is optional white space.

`noargletter` A single letter representing an option without
 an argument.

`argletter` A single letter representing an option
 requiring an argument.

`optarg` Argument (character string) satisfying
 preceding `argletter`.

`cmdarg` Path name (or other command argument) not
 beginning with - or, - by itself indicating the
 standard input.

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.

`getopt(1)`, `getopt(3C)`.

Section 6 of this volume for computer games.

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 other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

BUGS

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

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.

Archive file formats are 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 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 are:

- d** Delete the named files from the archive file.
- r** Replace the named files in 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.
- 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.
- t** Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. 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 the archive file.
- v Verbose. Under the verbose option, ar gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with t, it gives a long listing of all information about the files. When used with x, it precedes each file with a name.
- c Create. Normally ar will create afile when it needs to. 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 local directory.

Only the following combinations are meaningful:

d and m:	u and l
p:	none
q:	u and c
r:	u and v and l with a, b, or i
t and x:	v

Other combinations will ignore any modifier - e.g. p will cause v to be ignored.

FILES

/tmp/v* temporaries

SEE ALSO

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

BUGS

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

NAME

`at` - schedule and 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
at 1530 fri week
```

`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.yy
    activity to be performed at hour hhhh of day ddd of
    year yy. yy is a unique number.
/usr/spool/at/lasttimedone
    contains hhhh for last hour of activity.
/usr/spool/at/past
    directory of activities now in progress.
```

AT(1)

AT(1)

/usr/lib/atrun
program that executes activities 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.

BASENAME(1)

BASENAME(1)

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 substitution marks (``) within shell procedures.

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

EXAMPLES

The following example, 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

expr(1), sh(1).

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 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).

CC(1)

CC(1)

NAME

cc - C compiler

SYNOPSIS

cc [option] ... file ...

HP-UX COMPATIBILITY

Level:

C compiler - development

Origin:

System III

DESCRIPTION

Cc is the UNIX 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. See ld(1) for link editor options.

-c Suppress the link edit phase of the compilation, and force an object file to be produced even if only one program is compiled.

-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 (sdb(1)). Not all **HP-UX** systems support a symbolic debugger.

-dn This option is passed through to as(1). (VAX only.)

- 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.
- P Run only the macro preprocessor on the named C programs, and leave the result on corresponding files suffixed .i.
- C Comments are not stripped by the macro preprocessor.
- Dname=def
-Dname Define the name to the preprocessor, as if by **#define**. If no definition is given, the name is defined as 1.
- Uname Remove any initial definition of name.
- Idir Change the algorithm for searching for **#include** files whose names do not begin with / to look in dir before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in "" will be searched for first in the directory of the file 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 file argument is not searched.
- Bstring Find substitute compiler passes in the files named string with the suffixes **cpp**, **c0**, **c1** and **c2**. If string is empty, use a standard backup version.
- t(p02a1) 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**.

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**.

HARDWARE DEPENDENCIES

Series 500:

The **-p**, **-g**, and **-d** 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 **-B** option is supported, but no substitute compiler passes are provided.

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

/lib/cpp is relocated and renamed **/usr/bin/comp**;
/lib/c2 is relocated and renamed **/bin/option**;
/lib/oc*, **/lib/nc***, and **/lib/mcrt0.o** are not currently supported.

FILES

file.c	input file	
file.o	object file	
a.out	linked output	
/tmp/ctm*	temporary	
/lib/cpp	preprocessor	
/lib/ccom	compiler, cc	*
/lib/c2	optional optimizer	
/lib/oc*	backup compiler, gcc	
/lib/nc*	test compiler, gcc	
/lib/crt0.o	runtime startoff	*
/lib/mcrt0.o	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.

B. W. Kernighan, Programming in C-A Tutorial.

D. M. Ritchie, C Reference Manual.

adb(1), **as(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.

LANGUAGE CLARIFICATIONS

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 in all cases.

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.

CD(1)

CD(1)

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

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 <u>chmod(2)</u>
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.

CHMOD(1)

CHMOD(1)

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).

CHOWN(1)

CHOWN(1)

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).

CHROOT(1M)

CHROOT(1M)

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.

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

One should exercise extreme caution when referencing special files in the new root file system.

The command cannot be a shell script.

CMP(1)

CMP(1)

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

cp, ln, mv - copy, link or move files

SYNOPSIS

```
cp [-f] 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 with cp (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 without your intervention. -f is ignored by cp, however, since cp can never overwrite a non-writable file.

SEE ALSO

cpio(1), link(1M), rm(1), chmod(2).

BUGS

If file1 and target lie on different file systems, mv must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Ln will not link across file systems.

NAME

cpboot - copy boot area (operating system)

SYNOPSIS

/etc/cpboot sourcename destname

HP-UX COMPATIBILITY

Level:

Structured Directory Format - HP-UX/NUCLEUS

Origin:

HP

DESCRIPTION

Sourcename and destname refer to character or block special files which point to Structured Directory Format (SDF) volumes containing boot areas. Cpboot copies the valid contents only from the source boot area to the destination boot area. The operating system header is written last to destname, only after the rest of the boot area has been copied successfully. Note that the header on the destination volume is cleared before starting the copy.

If either sourcename or destname represents a block special file, cpboot creates and uses a temporary character special file which points to the same device.

Cpboot normally sends to stdout a series of information lines, some of which only appear if the source volume is one of a set of volumes.

"Total bytes in the system" is the total across all volumes.

"Bytes to transfer" is the actual number to be copied (loadable body only).

"Volume byte offset" is the offset in the system of this volume's chunk of system.

"Volume ID" is a description from this volume.

"Destination boot area size" is the total number of bytes available on destname.

Cpboot does not affect any of the files on the source or destination volumes.

The effective user ID must be zero (super-user).

The magic number in the beginning of the boot area must match one of the magic numbers in a list internal to cpboot.

DIAGNOSTICS

Appropriate error messages are given if:

Either sourcename or destname is the wrong type or does not contain a boot area;

The source boot area does not contain an operating system;

The destination boot area is not large enough;

The copy fails for any other reason.

SEE ALSO

sdfinit(1M)

QUESTIONS

The name on this will change to qscp and it will become part of the qsmgr package. Documentation for qsmgr(8) will be available soon. There will also be minor code changes to this program when the name is changed (in time for demo shipments).

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o [**acBvx**]

cpio -i [**BcdmPrtuvx**] [**patterns**]

cpio -p [**adlmuvx**] **directory**

HP-UX COMPATABILITY

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**, meta-characters **?**, *****, and **[...]** match the slash **/** character. The default for **patterns** is ***** (i.e., select all files).

Cpio -p (pass) copies out and in in a single operation. Destination path names are interpreted relative to the named **directory**.

The meanings of the available options are:

- a** Reset access times of input files after they have been copied.
- B** Input/output is to be blocked 5,120 bytes to the record (does not apply to the **pass** option; meaningful only with data directed to or from **/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). Files copied in this mode are not changed; 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.
- 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).
- 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 file and continue.

HARDWARE DEPENDENCIES

Series 500:

All files with i-nodes greater than or equal to 65535 are unlinkable with the -i option. A separate copy of each file is made instead.

Device files written with the -o option (eg. /dev/mt1) will probably not transport to HP-UX implementations other than the Series 500.

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), 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

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**. 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.

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 line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

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

CRON(1M)

CRON(1M)

`/usr/lib/cronlog`

SEE ALSO

`sh(1)`, `init(8)`.

DIAGNOSTICS

A history of all actions by `cron` are recorded in `/usr/lib/cronlog`, if it exists and is writeable 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. A more efficient algorithm could be used. The overhead in running `cron` is about one percent of the CPU, exclusive of any commands executed by `cron`.

NAME

cu - call another UNIX system

SYNOPSIS

cu [-sspeed] [-aacu] [-lline] [-h] [-o|-e] telno | dir

HP-UX COMPATIBILITY

Level:

Data Communications - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files. Speed gives the transmission speed (110, 150, 300, 1200, 4800, 9600); 300 is the default value. Most of our modems restrict us to choose between 300 and 1200. Directly connected lines may be set to other speeds.

The -a and -l values may be used to specify device names for the ACU and communications line devices. They can be used to override searching for the first available ACU with the right speed. The -h option emulates local echo, supporting calls to other computer systems which expect terminals to be in half-duplex mode. The -e (-o) option designates that even (odd) parity is to be generated for data sent to the remote. Telno is the telephone number, with equal signs for secondary dial tone or minus signs for delays, at appropriate places. The string dir for telno must be used for directly connected lines, and implies a null ACU.

Cu will try each line listed in the file /usr/lib/uucp/L-devices until it finds an available line with appropriate attributes or runs out of entries. After making the connection, cu runs as two processes: the transmit process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the receive process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings.

The transmit process interprets the following:

- ~. terminate the conversation.
- ~! escape to an interactive shell on the local system.

~!cmd... run cmd on the local system (via **sh -c**).

~\$cmd... run cmd locally and send its output to the remote system.

~%take from [to] copy file from (on the remote system) to file to on the local system. If to is omitted, the from argument is used in both places.

~%put from [to] copy file from (on local system) to file to on remote system. If to is omitted, the from argument is used in both places.

~"... send the line **~"...** to the remote system.

~nostop turn off the DC3/DC1 input control protocol for the remainder of the session. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters,

The receive process normally copies data from the remote system to its standard output. A line from the remote that begins with **~>** initiates an output diversion to a file. The complete sequence is:

```
~>[>]:file
zero or more lines to be written to file
~>
```

Data from the remote is diverted (or appended, if **>>** is used) to file. The trailing **~>** terminates the diversion.

The use of **~%put** requires stty(1) and cat(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of echo(1) and cat(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

FILES

```
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null
```

CU(1C)

CU(1C)

SEE ALSO

cat(1), echo(1), stty(1), uucp(1C), dh(4), dn(4), tty(4).

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

BUGS

There is an artificial slowing of transmission by cu during the **~Xput** operation so that loss of data is unlikely.

DATE(1)

DATE(1)

NAME

date - print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

HP-UX COMPATABILITY

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

DATE(1)

DATE(1)

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

EXAMPLE

date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
would generate as output:

DATE: 08/01/76

TIME: 14:45:05

DIAGNOSTICS

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

FILES

/dev/kmem
/usr/adm/wtmp

1

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 (e.g., /dev/HP7908) or by mounted directory name (e.g., /usr). 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.

FILES

/dev/HP79*
/dev/HP91*
/dev/HP98*
/etc/mnttab

SEE ALSO

fsck(1M), fs(5), mnttab(5).

NAME

diff - differential file comparator

SYNOPSIS

diff [-efbh] 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 $*; echo 'l,$p') | ed - $1
```

Except in rare circumstances, diff finds a smallest sufficient set of file differences.

DIFF(1)

DIFF(1)

Option **-h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **-e** and **-f** are unavailable with **-h**.

FILES

/tmp/d?????
/usr/lib/diffh for **-h**

SEE ALSO

cmp(1), comm(1), diff3(1), diffmk(1), dircmp(1), ed(1),
sdiff(1), sccsdiff(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 (.).

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.

ECHO(1)

ECHO(1)

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 `\`:

<code>\b</code>	backspace
<code>\c</code>	print line without new-line
<code>\f</code>	form-feed
<code>\n</code>	new-line
<code>\r</code>	carriage return
<code>\t</code>	tab
<code>\\</code>	backslash
<code>\n</code>	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number <code>n</code> , 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 REs 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 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., [ja-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.



ED(1)

ED(1)

The following rules may be used to construct REs 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 $\backslash(m\backslash)$, $\backslash(m,\backslash)$, or $\backslash(m,n\backslash)$ 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; $\backslash(m\backslash)$ matches exactly m occurrences; $\backslash(m,\backslash)$ matches at least m occurrences; $\backslash(m,n\backslash)$ 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 $\backslash($ and $\backslash)$ is a RE that matches whatever the unadorned RE matches.
- 2.6 The expression $\backslash n$ matches the same string of characters as was matched by an expression enclosed between $\backslash($ and $\backslash)$ earlier in the same RE. Here n is a digit; the sub-expression specified is that beginning with the n -th occurrence of $\backslash($ counting from the left. For example, the expression $\wedge\backslash(.*\backslash)\backslash 1\$$ 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 `q` 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.

• `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`, `c`, 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 file-name command changes the currently-remembered file name to file; otherwise, it prints the currently-remembered file name.

(1,s)g/RE/command list

In the global 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,s)G/RE/

In the interactive Global 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 help command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The Help 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 insert 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 join command joins contiguous lines by removing the appropriate new-line characters. If only one address is given, this command does nothing.

(.)kx

The 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 list 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.

(.,.)ma

The move 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 number 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 print 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, c, 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 quit command causes ed to exit. No automatic write of a file is done (but see DIAGNOSIS below).

Q

The editor exits without checking if changes have been made in the buffer since the last w command.

(\$)r file

The read 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 substitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by 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.

(.,.)ta

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 undo 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, n, s, t, v, G, or V command.

(1,\$)v/RE/command list

This command is the same as the global command g except that the command list is executed with . initially set to every line that does not match the RE.

(1,\$)V/RE/

This command is the same as the interactive global command G except that the lines that are marked during the first step are those that do not match the RE.

(1,\$)w file

The write command writes the addressed lines into the named file. If the file does not

exist, it is created with mode 666 (readable and writable by everyone), unless your `umask` setting (see `sh(1)`) dictates otherwise. The currently-remembered file name is not changed unless `file` is the very first file name mentioned since `ed` was invoked. If no file name is given, the currently-remembered file name, if any, is used (see `e` and `f` commands); `.` is unchanged. If the command is successful, the number of characters written is typed. If `file` 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.

X

A key string is demanded from the standard input. Subsequent `e`, `c`, 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 UNIX 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.

DIAGNOSTICS

? for command errors.
 ?file for an inaccessible file.
 (use the help and Help 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.

SEE ALSO

awk(1), crypt(1), e(1), edit(1), ex(1), grep(1), sed(1),
sh(1), vi(1), view(1).

A Tutorial Introduction to the UNIX Text Editor by B. W. Kernighan.

Advanced Editing on UNIX by B. W. Kernighan.

CAVEATS AND BUGS

A ! command cannot be subject to a g or a y 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

ex, edit, e - text editor

SYNOPSIS

```
ex [-] [-v] [-t tag] [-r] [+ command] [-l] name ...
edit [ex options]
e [ex options]
```

HP-UX COMPATIBILITY

Level:

HP-UX/STANDARD

Origin:

UCB

Remarks:

Ex was written originally by William Joy.
Mark Horton has maintained the editor since version 2.7, adding macros, support for many unusual terminals, and other features such as word abbreviation mode.

DESCRIPTION

Ex is the root of a family of editors: edit, ex, e, and vi.
Ex is a superset of ed, with the most notable extension being a display editing facility. Display based editing is the focus of vi.

If you have 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.

DOCUMENTATION

The document Edit: A tutorial provides a comprehensive introduction to edit, assuming no previous knowledge of computers of the **HP-UX** system.

The Ex Reference Manual - Version 3.5 is a comprehensive and complete manual for the command mode features of ex, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of ex, see the editing documents written by Brian Kernighan for the editor ed; the material in the introductory and advanced documents also works with ex.

An Introduction to Display Editing With Vi introduces the display editor vi, and provides reference material on vi. All of these documents can be found in volume 2c of the

Programmer's Manual. In addition, the Vi Quick Reference card summarizes the commands of vi in a useful, functional way, and is useful with the Introduction.

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/Exnnnnnn	editor temporary
/tmp/Rxnnnnnn	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), vi(1), environ(5), termcap(5).

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.

QUESTIONS

The references - do they exist?



EXPR(1)

EXPR(1)

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 just `"file"`

`expr $a : '.*\/(.*\)' \1 $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`.

SEE ALSO

ed(1), sh(1), test(1).

EXIT CODE

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.

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=
```

FIND(1)

FIND(1)

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 file True if file 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 <u>n</u>	True if the file has been accessed in <u>n</u> days.
-mtime <u>n</u>	True if the file has been modified in <u>n</u> days.
-ctime <u>n</u>	True if the file has been changed in <u>n</u> days.
-exec <u>cmd</u>	True if the executed <u>cmd</u> returns a zero value as exit status. The end of <u>cmd</u> must be punctuated by an escaped semicolon. A command argument () is replaced by the current path name.
-ok <u>cmd</u>	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 <u>device</u>	Write the current file on <u>device</u> in <u>cpio</u> (5) format (5120 byte records).
-newer <u>file</u>	True if the current file has been modified more recently than the argument <u>file</u> .
(expression)	True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).
-inum <u>n</u>	True if the file has inode number <u>n</u> .
-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

FIND(1)

FIND(1)

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

`fsck` - file system consistency check and interactive repair

SYNOPSIS

`/etc/fsck [-y] [-n] [-s] [-d[d[d[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 only.

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 begin 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.
- 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**.

Normal messages from fsck are written to **stderr**. Information generated because of the **-d** option 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.
7. Multiple files claiming the same blocks.

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.

FILES

/etc/checklist contains the default list of file systems to check

DIAGNOSTICS

The diagnostics are intended to be self-explanatory.

FSCK(1)

(Series 500 Only)

FSCK(1)

SEE ALSO

checklist(5).

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.

You should run a backup prior to running fsck for repairs.

GETOPT(1)

GETOPT(1)

NAME

getopt - parse command options

SYNOPSIS

set -- 'getopt optstring \$*'

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 a shell argument.

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in optstring.

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:

GETOPT(1)

GETOPT(1)

```
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).

NAME

grep, egrep, fgrep - search a text 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. Egrep 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 separated by new-lines.

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.

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.

ID(1)

ID(1)

NAME

id - print user and 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), getuid(2), getgid(2).

INSTALL(1M)

INSTALL(1M)

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.

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 <u>dira</u> | Installs a new command in the directory specified in <u>dira</u> . Looks for <u>file</u> in <u>dira</u> and installs it there if it is not found. If it is found, <u>install</u> issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the -s option. |
| -f <u>dirb</u> | Forces <u>file</u> 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 |

INSTALL(1M)

INSTALL(1M)

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 in **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 **OLDfile** 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).

KILL(1)

KILL(1)



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 the link editor for the Series 500 only.

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 the linking process.

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 when it defines a needed symbol. (The linker includes library code as it scans the library). Order is important here: defining elements must follow the corresponding unresolved reference.

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 language) is that 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 language) are predefined (see section 3). User programs cannot define these names for external use. One exception to this occurs when you link a program using `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.

`Ld` recognizes the following options. A colon indicates that the option accepts a modifier.

- `-A` keep the D-data and I-data areas apart (in separate segments).
- `-T` keep the D-data and I-data areas together (in the same GDS segment).
- `-V:` assign a 32-bit integer version number to the resulting output file.
- `-a:` mark all segments (except GDS) in the program as virtual.
- `-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.
- `-h:` for `-r` output, hide the named symbol by clearing the external bit in the symbol table.
- `-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 `libxxx.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 sequential libraries, it may be necessary to specify them 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.

- `-m:` specify a maximum heap memory size for programs with I-data in paged EDS. `Exec` uses this information to allocate the necessary number of consecutively numbered paged data segments.
- `-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`).
- `-p` arrange for the I-data area to reside in paged external data segments (PEDS). This option is not accessible via `cc(1)` because it conflicts with the compiler's profile option.
- `-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 the debugger. 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 symbol 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.

DIAGNOSTICS

The linker issues the warning `DON'T RELY ON -r OUTPUT` when it encounters a `-r` option. `Ld` will continue to process, however.

SEE ALSO

`ar(1)`, `getopt(1)`, `nm(1)`, `pm(1)`, `ranlib(1)`, `strip(1)`, `edata(3)`, `end(3)`, `etext(3)`, `a.out.h(5)`.

LIFCP(1)

LIFCP(1)

NAME

lifcp - copy to or from LIF files.

SYNOPSIS

```
lifcp [-a] [-b] file1 file2
lifcp [-a] [-b] file1 ... fileN
```

HP-UX COMPATABILITY

Level:

HP-UX/NUCLEUS

Origin:

HP

DESCRIPTION

Lifcp copies a LIF file to a UNIX file, a UNIX file to a LIF file or a LIF file to another LIF file. It also copies a list of (UNIX/LIF) files to a (LIF/UNIX) 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 UNIX to LIF copy, a LIF ASCII file is created. The -b option will force binary copy, creating LIF BINARY files.

If neither of the 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.

A LIF file is recognized on the command line by a preceeding `.'. A LIF directory is recognized by a terminating `.'. If `-' is used for file name, standard input or output will be assumed. LIF file names cannot exceed 10 characters. A new file name must consist of uppercase alphanumeric characters (A to Z and 0 to 9) and underscore character. The first character must be a letter. Old file names may consist of any characters. Ones that have special meaning to the shell must be escaped.

EXAMPLES

```
lifcp ../udir/ufile ../test/lvol.LFILE
lifcp ldir.STDOUT -
lifcp - ldir.STDIN
lifcp ufile /dev/rmf0.LFILE
lifcp ldir1.LFILE ldir2.LFILE_A
lifcp ldir.LFILE ufile
lifcp U1_C U2_C U3_C ldir
lifcp *_C ldir
lifcp ldir.L1 ldir.L2 ldir.L3 udir
lifcp ldir.L1 ldir.L2 ldir.L3 Ldir
```

SEE ALSO

lifinit(1) lifrn(1) lifrm(1) lifls(1)

LIFCP(1)

LIFCP(1)

LIF Logical Interchange Format, Hewlett Packard, part no.
5955-2676, January 1982.

DIAGNOSTICS

Lifcp returns exit code 0 if the file is copied
successfully. Otherwise it prints a diagnostic and returns
nonzero.

LIFINIT(1)

LIFINIT(1)

NAME

lifinit - write a LIF volume header on the file.

SYNOPSIS

lifinit [options] file

HP-UX COMPATABILITY

Level:

HP-UX/NUCLEUS

Origin:

HP

DESCRIPTION

Lifinit writes a LIF volume header on a volume or file. Options may appear in any order. Their meanings are:

- vn Sets the volume size to n 256 byte sectors. If n is not a multiple of four, it is increased to next higher number which is a multiple of four. This is done since on DC600 cartridge tapes the logical unit is 1k or four sectors long.
- dn Sets the directory size to n file entries. Each sector can hold up to eight file entries. If n is not a multiple of eight then it will be rounded up to next higher multiple of eight.

The default values for volume size and directory size are listed in the following table. In case the user specifies a volume size between two specified ranges and gives no directory size then the directory size will be set as if the volume was the smaller size. If directory size is the only specified option, then volume size is set to the default value. Smaller volume size will be selected if the directory size is in between two ranges. Volume size must be at least one sector long and strictly greater than the directory size otherwise an error will be generated. The default volume and directory size for UNIX files is the same as for the internal 5" floppies.

MEDIA	VOLUME SIZE(sectors)	DIRECTORY SIZE(# of files)

9826/9836 (built in)	1056	112
8290X	1056	112
9121	1056	112
9895 (single sided)	1890	224
9895 (double sided)	4500	464
9134A (each unit)	4500	464

LIFINIT(1)**LIFINIT(1)**

9134A	18848	752
9134B (10Mb)	37820	1512
7908	64750	2584
7911	109824	4392
7912	256256	10248
7933	1579916	63192

EXAMPLES

```
lifinit -v500 -d10 x
lifinit /dev/rmf0
```

SEE ALSO

lifcp(1) lifrn(1) lifls(1) lifrm(1)
LIF Logical Interchange Format, Hewlett Packard, part no.
5955-2676, January 1982.

DIAGNOSTICS

Lifinit returns exit code 0 if the volume is initialized successfully. Otherwise it prints a diagnostic and returns nonzero.

BUGS

Lifinit does not initialize the media.
In order for Lifinit to know the right size of the volume.
stat(1) should be modified so that it would return the size of the media.

NAME

lifls - list content of a LIF directory.

SYNOPSIS

lifls [option] name

HP-UX COMPATABILITY

Level:

HP-UX/NUCLEUS

Origin:

HP

DESCRIPTION

Lifls with no options lists the content of a LIF directory to the users's terminal. File names are listed horizontally across the screen like ls(1) but are not sorted. Name is a path name naming a LIF volume. 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 multi column output, e.g. to a file or pipe.

EXAMPLES

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

SEE ALSO

lifcp(1) lifrn(1) lifrm(1) lifinit(1)
LIF Logical Interchange Format, Hewlett Packard, part no. 5955-2676, January 1982.

DIAGNOSTICS

Lifls returns exit code 0 if the directory was listed successfully. Otherwise prints a diagnostic and returns nonzero.

NAME

lifrm - remove a LIF file.

SYNOPSIS

lifrm file1 ... fileN

HP-UX COMPATABILITY

Level:

HP-UX/NUCLEUS

Origin:

HP

Computer
Museum

DESCRIPTION

Lifrm removes one or more entries from a LIF directory. LIF files at the command line are recognized by a preceeding ' '. File names may consist of any characters. Ones that have special meaning to the shell must be escaped.

EXAMPLES

lifrm cat.MAN

lifrm /dev/rmf0.F

SEE ALSO

lifcp(1) lifrm(1) lifls(1) lifinit(1)

LIF Logical Interchange Format, Hewlett Packard, part no. 5955-2676, January 1982.

DIAGNOSTICS

Lifrm returns exit code 0 if the file is removed successfully. Otherwise it prints a diagnostic and returns nonzero.

NAME

lifrn - rename LIF files.

SYNOPSIS

lifrn oldfile newfile

HP-UX COMPATABILITY

Level:

HP-UX/NUCLEUS

Origin:

HP

DESCRIPTION

Lifrn changes the name of oldfile to newfile. This operation does not include copy or delete. LIF files are recognized on the command line by a preceding ':'. LIF file name cannot exceed 10 characters. A new file name must consist of uppercase alphanumeric characters (A to Z and 0 to 9) and underscore character. The first character must be a letter. Old file names may consist of any characters. Ones that have special meaning to the shell must be escaped.

EXAMPLES

lifrn cat:F cat:D

SEE ALSO

lifcp(1) lifinit(1) lifls(1) lifrm(1)

LIF Logical Interchange Format, Hewlett Packard, part no. 5955-2676, January 1982.

DIAGNOSTICS

Lifrn returns exit code 0 if the file name is changed successfully. Otherwise it prints a diagnostic and returns nonzero.

LINK(1M)

LINK(1M)

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.

ERROR CODES

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

SEE ALSO

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

LOGIN(1)

LOGIN(1)

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 (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.

If password aging has been invoked by the super-user on your behalf, your password may have expired. In this case, you will be shunted 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 you are notified if there is mail in your mail file (if you have one). Finally, the profiles (i.e., `/etc/profile` and `$HOME/.profile`) (if any) are executed (see `profile(5)`). `Login` initializes the user and group IDs and the working directory, then executes a command interpreter (usually `sh(1)`) according to specifications found in the `/etc/passwd` file. 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`.

LOGIN(1)

LOGIN(1)

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	accounting
/usr/adm/wtmp	accounting
/usr/mail/ <u>your-name</u>	mailbox for user <u>your-name</u>
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
\$HOME/.profile	personal profile

SEE ALSO

mail(1), newgrp(1), sh(1), passwd(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 a UNIX programming counselor.

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.

LOGNAME(1)

LOGNAME(1)

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

lpd - line printer daemon

SYNOPSIS**/usr/lib/lpd****HP-UX COMPATIBILITY**

Level:

HP-UX/STANDARD

Origin:

System III

DESCRIPTIONLpd is the daemon for the line printer.

Lpd uses the directory **/usr/spool/lpd**. The file **lock** is used to prevent two daemons from becoming active. 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.

- 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.

Lpd is automatically initiated by the line printer command, lpr.

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.

Any error encountered will cause the daemon to give up, wait up to 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.

FILES

<code>/usr/spool/lpd/*</code>	spool area for line printer daemon.	*
<code>/etc/passwd</code>	used to get the user's name.	
<code>/dev/lp</code>	line printer device.	*

SEE ALSO

`lpr(1)`.

QUESTIONS

`#IDENT`

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 names appear, the standard input is assumed; thus `lpr` may be used as a filter.

The following options 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.

FILES

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

SEE ALSO

`lpd(1)`.

NAME

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

SYNOPSIS

```
ls [ -abcdfgilanoqrstuxlACFR ] names
l [ls options] names
ll [ls options] names
lsf [ls options] names
lsr [ls options] names
lsx [ls options] names
```

HP-UX COMPATABILITY

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.

There are three major listing formats. The format chosen depends on whether the output is going to a teletype, 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. |
- g The same as -l, except that the owner is not printed.
- 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, list only its name; often used with -l to get the status of a directory.
- 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. |
- l 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.

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;
- p** if the entry is a fifo (a.k.a. 'named pipe') 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.

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 -sllpr'**. On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers. |

Column widths choices are poor for terminals which can tab. |

NAME

lsdev - list device drivers in the system

SYNOPSIS

lsdev [major...]

HP-UX COMPATABILITY

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).

Note: Some device drivers available from the system may be intended for use by other drivers. You may get unexpected results if you try to use them directly from a special file.

DIAGNOSTICS

Lists the drivename as "NO SUCH DRIVER" when appropriate.

SEE ALSO

Sections on specific device drivers.

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 [<u>files</u>]	Save message in the named <u>files</u> (<u>mbox</u> is default).
w [<u>files</u>]	Save message, without its header, in the named <u>files</u> (<u>mbox</u> is default).
m [<u>persons</u>]	Mail the message to the named <u>persons</u> (yourself is default).
q	Put undeleted mail back in the <u>mailfile</u> and stop.
EOT (control-d)	Same as q .
x	Put all mail back in the <u>mailfile</u> unchanged and stop.
!command	Escape to the shell to do <u>command</u> .
*	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 <u>mail</u> to terminate after interrupts. Normally an interrupt only causes the termination of the printing of the current message.
-ffile	

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 *; <u>mailfile</u>
\$HOME/mbox	saved mail
\$MAIL	<u>mailfile</u>
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), news(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**.

QUESTION

Inform on login fails.

NAME

make - maintain, update, and regenerate (compile) groups of programs

SYNOPSIS

make [-f makefile] [-p] [-i] [-k] [-s] [-r] [-n] [-b] [-e]
[-m] [-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:

- 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 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 makefiles.
- e Environment variables override assignments within makefiles.

- m** Print a memory map showing text, data, and stack. This option is a no-operation on systems without the `getu` system call.
- 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.
- 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.
- .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 no **-f** option is present, `makefile`, `Makefile`, `s.makefile`, and `s.Makefile` are tried in order. If `makefile` is `-`, the standard input is taken. 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, then a `,`, then 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). 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 **%X** 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, 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 -fp - 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.o:
    $(CC) -c $(CFLAGS) $(  
    ar rv $@ $*.o  
    rm -f $*.o
```

In fact, the `.c.o` 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.o;
```

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.o:` 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 very cumbersome if the archive library contains a mix of assembly programs and C programs.

HARDWARE DEPENDENCIES

Series 500:

The `-m` option is not currently supported.

FILES

[Mm]akefile
s.[Mm]akefile

SEE ALSO

`sh(1)`, `mk(8)`.

Make-A Program for Maintaining Computer Programs by S. I. Feldman.

An Augmented Version of Make by E. G. Bradford.

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

MAKE(1)

MAKE(1)

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.

QUESTIONS

-m option.

MESG(1)

MESG(1)

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. 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)`.

DIAGNOSTICS

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

NAME

mknod - create special, fifo file

SYNOPSIS

```
/etc/mknod pathname p  
/etc/mknod pathname c b drivername [selcode [addr1 [addr2  
[addr3]]]]
```

HP-UX COMPATIBILITY

Level:

HP-UX/NUCLEUS

Origin:

HP

Remarks:

This is a temporary manual page for the Series 500 only.

DESCRIPTION

The first form of mknod (under SYNOPSIS) creates a fifo special file. The second form creates either a block special file (for discs, tape, and other mountable volumes) or a character special file (for other devices). The special file must be created as pathname (and not already exist).

Drivername is the name of the I/O driver which the new special file will invoke (see lsdev(1M)). If drivername is not in the system, mknod checks to see if that argument is actually a decimal or octal number (the latter must begin with 0). If so, the specified driver number is used. This enables you to create special files for drivers that are not yet installed in the system.

Driver names may be entered in lower-case or upper-case.

Selcode is the select code of the device, as determined by the following formula: $(IOP-1) \times 8 + \text{peripheral address}$. Addr1, addr2, and addr3 specify sub-addresses, such as the HP-IB address, unit, and volume, depending on the particular device driver (see the specific descriptions in section 4).

Numbers may be entered in decimal or octal. If octal is used, the first digit must be 0. Missing arguments default to 0. If a device is not on a bus but has a non-zero unit or volume number, you should explicitly give a bus address of 0.

Note that the range of address values is not checked. Numbers out of range may alter other address values.

Device files are created with a mode of 0666, modified by the current value of umask. For fifo files, mknod also changes the user and group ownerships of the file to the current real user and group ID's.

Any user can create fifo files, but to create character or block special files, your real user ID must be that of the super-user (0).

Each special file gives access to a specific type of device on a specific address. These details cannot be changed for existing device files.

DIAGNOSTICS

Must be super-user

You do not have permission to create block or character special files.

Bad number

You specified an invalid driver number, select code, or address.

Usage ...

where "..." is a syntax description of mknod. You passed mknod an invalid argument list.

Other errors may be generated by perror(3) if mknod fails.

SEE ALSO

lsdev(1M), umask(1), specific device drivers.

MOUNT(1M)

MOUNT(1M)

NAME

mount, umount - mount and dismount 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

Mount issues a warning if the file system to be mounted is currently mounted under another name.

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.

MOUNT(1M)

MOUNT(1M)

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 having the effect of -r. |

An error will occur if mnttab does not exist. |

Names are truncated to MNTLEN bytes (see mnttab(5)). |



MVDIR(1M)

MVDIR(1M)

NAME

`mvdir` - move a directory

SYNOPSIS

`/etc/mvdir` `dirname` `name`

HP-UX COMPATIBILITY

Level:

HP-UX/NUCLEUS

Origin:

System III

DESCRIPTION

`Mvdir` renames directories within a file system. `Dirname` must be a directory; `name` must not exist. Neither name may be a sub-set of the other (`/x/y` cannot be moved to `/x/y/z`, nor vice versa).

Only the super-user can use `mvdir`.

BUGS

The restriction on names above helps to prevent the 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.

SEE ALSO

`mkdir(1)`.

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 newgroup 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.

When most users log in, they are members of the group named **other**.

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

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 <u>stdin</u> is a non-tty file, this message is given.
You have no shell:	Exec of the shell failed.

SEE ALSO

login(1), group(5).

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.

NICE(1)

NICE(1)

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.

NM(1)

NM(1)

NAME

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

SYNOPSIS

nm [-gnoprsu] [file ...]

HP-UX COMPATIBILITY

Level:

HP-UX/DEVELOPMENT

Origin:

System III

Remarks:

This command is dependent on the **a.out** format, which is in turn architecture dependent. Thus, there may be variations in the results of this command across HP-UX systems.

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.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

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.
- s Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next highest value). This difference is the value printed.

This flag turns on **-g** and **-n** and turns off **-u** and **-p**.

-u Print only undefined symbols.

HARDWARE DEPENDENCIES

Series 500:

The **-s** option is not currently implemented.

The output format described above has been changed.

The heading for the output is as follows:

U	X	TYPE	CONTENTS	NAME
=	=	====	=====	=====

where **U** specifies whether the symbol is defined (.) or undefined (U), **X** specifies whether the symbol is non-external (.) or external (X), **TYPE** specifies the linker symbol type (as defined in `a.out.h`), and **CONTENTS** lists the variant part of the symbol table entry. The variant part is determined by the symbol type (shown in capital letters), as follows:

```

ABS
value
FUNC or ENTRY
ptr_type segment offset stt_index
SYSTEM
entry_type segment stt
LABEL
ptr_type segment offset
DDATA
base_reg displacement
IDATA or UDATA
displacement
DCOMM or ICOMM
blocksize address
PTR
ptr_to_common target address
SEGMENT
seg_number
FILENAME
sequence

```

NAME is the name of the symbol.

DIAGNOSTICS

Nm generates an error message for the following conditions:

- invalid option;
- cannot open file;
- bad magic number
- read error

NM(1)

NM(1)

SEE ALSO

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

QUESTIONS

Are member names given for archives?

**NAME**

od - octal file dump

SYNOPSIS

od [-bcdox] [file] [[+ loffset [.] [b]]]

HP-UX COMPATIBILITY

Level:

HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Od dumps file in one or more formats as selected by the first argument. If the first argument is missing, **-o** is default. The meanings of the format options are:

- b** Interpret bytes in octal.
- 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 hex.

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, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

SEE ALSO

adb(1).

PASSWD(1)

PASSWD(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 monospace. 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. 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).

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.

PASSWD(1)**PASSWD(1)**

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.

FILES

/etc/passwd

SEE ALSO

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

NAME

`pr` - print and paginate files, make columnar listings

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.
- `-eck` 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).
- `-ick` 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).

- ngk** 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).
- wk** Set the width of a line to k character positions (default is 72 for equal-width multi-column output, no limit otherwise).
- ok** 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.
- lk** 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.
- sc** 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
```

PR(1)

PR(1)

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).

NAME

ps - report process status

SYNOPSIS

ps [options]

HP-UX COMPATIBILITY

Level:

HP-UX/NUCLEUS

Origin:

System III

Remarks:

There are various implementations of the kernel, and thus the information returned by ps may differ across implementations. This page reflects the Bell-derived version.

DESCRIPTION

Ps prints certain information about active processes. Without 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 llist.
 - 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 llist. 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 llist.

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** (1) 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** (1) The state of the process:
 - 0 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** (1) The priority of the process; higher numbers mean lower priority.
- NI** (1) Nice value; used in priority computation.
- ADDR** (1) The memory address of the process, if resident; otherwise, the disk address.

SZ (1) The size in blocks of the core image of the process.

WCHAN (1) The event for which the process is waiting or sleeping; if blank, the process is running.

TTY (all) The controlling terminal for the process.

TIME (all) The cumulative execution time for the process.

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.

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 close approximation to reality. Some data printed for defunct processes are irrelevant.

PWD(1)

PWD(1)

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:

System III

DESCRIPTION

Ranlib converts each named archive to a form which can be loaded more rapidly by the loader. It does this by adding a table of contents named __SYNDEF 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. If a ranlib library is updated by ar, you must re-execute ranlib on the library again to update the table of contents.

SEE ALSO

ar(1), ld(1)

BUGS

Because generation of a library by ar and randomization by ranlib are separate, phase errors are possible. The loader 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.

REVISION(1)

(Series 500 Only)

REVISION(1)

NAME

revision - get HP-UX revision information

SYNOPSIS

/etc/revision

HP-UX COMPATIBILITY

Level:

HP-UX/EXTENDED

Origin:

HP

DESCRIPTION

This command prints five lines to the standard output. Those five lines consist of four line from uname(2) which give information on the kernel, followed by the command level revision.

Typical output looks like this:

Sysname: HP-UX

Nodename: sww

Release: QA10

Version: 9030/40

Unicorn Revision 3

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 antisocial consequences of inadvertently doing something like:

`rm -r .*`

NAME

sdfinit - initialize Structured Directory Format volume

SYNOPSIS

```
/etc/sdfinit [-i] pathname [blocksize [bootsize  
[interleave]]]
```

HP-UX COMPATIBILITY

Level:

Structured Directory Format - HP-UX/NUCLEUS

Origin:

HP

Remarks:

This command is valid for the Series 500 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.

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).

DIAGNOSTICS

Appropriate error messages are given if the argument list is incorrect, pathname cannot be initialized, or any other

SDFINIT(1M)

(Series 500 Only)

SDFINIT(1M)

error occurs.

SEE ALSO

osmgr(8), section 4.

SETMNT(1M)

SETMNT(1M)

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(1M) and umount(1M) commands. Setmnt reads standard input and creates a mnttab entry for each line. Input lines have the format:

filesystem node

where filesystem is the name of the file system's special file and node is the root name of that file system. Thus filesystem and node become the first two strings in the mnttab(5) entry. *

FILES

/etc/mnttab

SEE ALSO

mnttab(5).

BUGS

Filesystem 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(1M) and umount(1M).

NAME

sh - shell, the standard command programming language

```
sh - ( ) " ' : . break continue eval exec exit export read
readonly set shift times trap for case if while if elif esac
do done $$ $* $@ $? $- $! $$
```

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 |. 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.
- 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.

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 <u>word</u> as standard input (file descriptor 0).
>word	Use file <u>word</u> 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 <u>word</u> 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 <u>word</u> , or to an end-of-file. The resulting document becomes the standard input. If any character of <u>word</u> is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) <code>\new-line</code> is ignored, and <code>\</code> must be used to quote the characters <code>\</code> , <code>\$</code> , <code>'</code> , and the first character of <u>word</u> . If <code>-</code> is appended to <code><<</code> , then all leading tabs are stripped from <u>word</u> and from the document.
<&digit	The standard input is duplicated from file descriptor <u>digit</u> (see <u>dup(2)</u>). Similarly for the standard output using <code>></code> .
<&-	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`.
- 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 names 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 names are marked readonly and the values of the these names 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 \$! 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 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.

EXIT STATUS

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).

FILES

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

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.

SHUTDOWN(1M)

SHUTDOWN(1M)

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 UNIX 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(1M)). 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(1M).

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 very quickly and gives the user very little time to log out.

SEE ALSO

sync(1M), umount(1M).

SLEEP(1)

SLEEP(1)

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 65536 seconds.

SORT(1)

SORT(1)

NAME

sort - sort and/or merge files

SYNOPSIS

sort [-cmubdfinrtx] [+pos1 [-pos2]] ... [-o output]
[names]

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.
- tx** "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.

SORT(1)

SORT(1)

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 **-tx** 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.

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???

SEE ALSO

comm(1), **join(1)**, **uniq(1)**.

SORT(1)

SORT(1)

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions and for disorder discovered under option -c.

BUGS

Very long lines are silently truncated.

STRIP(1)

STRIP(1)

NAME

strip - remove symbols and relocation bits

SYNOPSIS

strip name ...

HP-UX COMPATIBILITY

Level:

HP-UX/DEVELOPMENT

Origin:

System III

Remarks:

This command is dependent on the **a.out** format, which is in turn architecture dependent. Thus, there may be variations in the results of this command across HP-UX systems.

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. This is useful to save space after a program has been debugged.

The effect of strip is the same as use of the **-s** option of ld.

If name is an archive file, strip will remove the local symbols from any **a.out** format files it finds in the archive. Certain libraries, such as those residing in **/lib**, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

FILES

/tmp/stm* temporary file

SEE ALSO

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

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 sanity 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.

clocal (-clocal)

assume a line without (with) modem control.

Input Modes

ignbrk (-ignbrk)

ignore (do not ignore) break on input.

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.
iuc1c (**-iuc1c**)
 map (do not map) upper-case alphabets 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 alphabets 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 ERASEd 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**, **iuc1c**, 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**, or **tek**.

SEE ALSO

tabs(1), **ioctl(2)**, **tty(4)**.

QUESTIONS

What about **tty33** ...

NAME

`su` - substitute or change users; become super-user or 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 the super-user to run shell procedures with restricted privileges (an `arg` of the form `-c string` executes `string` via the shell). 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, except that if `$PATH` is not found (not exported), `$PATH` will be set to `:/bin:/usr/bin`. The super-user's `$PATH` is unconditionally set to `/bin:/etc:/usr/bin`. `$PATH` will always be reset to the above values if additional arguments are given. Note that the `.profile` can check `arg0` for `-sh` or `-su` to determine how it was invoked.

`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 `-`.

SU(1)

SU(1)

FILES

/etc/passwd	system's password file
\$HOME/.profile	user's profile

SEE ALSO

env(1), login(1), sh(1), environ(7).

SYNC(1M)

SYNC(1M)

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).

TABS(1)

TABS(1)

NAME

`tabs` - set tabs on a terminal

SYNOPSIS

`tabs [ tabspec ] [ +mn ] [ -Ttype ]`

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.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the left margin on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

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, e.g., the DASI 300, DASI 300s, and DASI 450.

`-code` Gives the name of one of a set of ``canned'' tabs. The legal codes 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. Note that such a setting leaves a left margin of n columns on TerminoNet terminals only. 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 `nc(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:

-Ttype `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.

+mn 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. For a TerminoNet, the first value in the tab list should be 1, or the margin will move even further to the right. 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.

DIAGNOSTICS

<code>illegal tabs</code>	when arbitrary tabs are ordered incorrectly.
<code>illegal increment</code>	when a zero or missing increment is found in an arbitrary specification.
<code>unknown tab code</code>	when a "canned" code cannot be found.
<code>can't open</code>	if <code>--file</code> option used, and file can't be opened.
<code>file indirection</code>	if <code>--file</code> option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

`nroff(1)`, `tset(1)`, `environ(7)`, `term(7)`.

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.

QUESTIONS

GE TerminoNet
DASI
S/370
UNIVAC
PL/1

NAME

tar - tape file archiver

SYNOPSIS

tar [key] [files]

HP-UX COMPATABILITY

Level:

HP-UX/STANDARD

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. 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 tape. 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. This command implies the **r** function.
- o** On output, tar normally places information specifying owners and modes of directories in the archive. Former versions of tar, when encountering this information, will give error message 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.

The following characters may be used in addition to the letter that selects the desired function:

- 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.

If a tape drive is used as the output device, it must be configured in Berkeley compatibility mode; see **magtape(4)**.

FILES

/dev/rmt?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.

Complaints 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 disk file, the b option should not
ever be used, because updating an archive stored on disk 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. |

TEE(1)

TEE(1)

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.

TEST(1)

TEST(1)

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.

TEST(1)

TEST(1)

`-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

`find(1)`, `sh(1)`, `eval(1)`.

WARNING

In the second form of the command (i.e., the one that uses `[]`, rather than the word `test`), the square brackets must be delimited by blanks.

TIME(1)

TIME(1)

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.

SEE ALSO

timex(1), times(2), times command in sh(1).



TOUCH(1)

TOUCH(1)

NAME

touch - update access and modification times of a file;
create empty file

SYNOPSIS

touch [-amc] [mmddhhmm[yy]] files

HP-UX COMPATIBILITY

Level:
HP-UX/NUCLEUS

Origin:
System III

DESCRIPTION

Touch causes the access and modification 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).

TRUE(1)

TRUE(1)

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.

TTY(1)

TTY(1)

NAME

tty - get the terminal's name

SYNOPSIS

tty [-s]

HP-UX COMPATIBILITY

Level:

HP-UX/STANDARD

Origin:

System III

DESCRIPTION

It prints the path name of the user's terminal. The -s option inhibits printing, allowing one to test just the exit code.

EXIT CODES

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

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.

Umask is recognized and executed by the shell.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

UNAME(1)

UNAME(1)

NAME

uname - print name of current HP-UX version

SYNOPSIS

uname [-snrva]

HP-UX COMPATIBILITY

Level:

HP-UX/NUCLEUS

Origin:

System III

DESCRIPTION

Uname prints the current system name of UNIX on the standard output file. It is mainly useful to determine what system one is 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 to a communications network). (e.g. uucp).
- r print the operating system release.
- v print the operating system version.
- a print all the above information.

SEE ALSO

uname(2).

NAME

uuclean - uucp spool directory clean-up

SYNOPSIS

uuclean [options] ...

HP-UX COMPATIBILITY

Level:

Data Communications - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Uuclean will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

-ddirectory

Clean directory instead of the spool directory.

-ppre

Scan for files with pre as the file prefix. Up to 10 **-p** arguments may be specified. A **-p** without any pre following will cause all files older than the specified time to be deleted.

-ntime

Files whose age is more than time hours will be deleted if the prefix test is satisfied. (default time is 72 hours)

-m

Send mail to the owner of the file when it is deleted.

This program will typically be started by cron(1M).

FILES

/usr/lib/uucp

directory with commands used by uuclean
internally

/usr/spool/uucp

spool directory

SEE ALSO

uucp(1C), uux(1C).

NAME

uucp, uulog, uuname - unix to unix copy; file transfer

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

uuname

HP-UX COMPATIBILITY

Level:

Data Communications - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where system-name is taken from a list of system names which uucp knows about. Shell metacharacters ?*[] appearing in path-name will be expanded on the appropriate system.

Path names may be one of:

- (1) a full path name;
- (2) a path name preceded by "user where user is a login name on the specified system and is replaced by that user's login directory;
- (3) a path name preceded by "/user where user is a login name on the specified system and is replaced by that user's directory under PUBDIR (see FILES);
- (4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see chmod(2)).

The following options are interpreted by uucp:

-d Make all necessary directories for the file copy

(default).

- f Do not make intermediate directories for the file copy.
- c Use the source file when copying out rather than copying the file to the spool directory (default).
- C Copy the source file to the spool directory.
- m Send mail to the requester when the copy is complete.
- nuser Notify user on the remote system that a file was sent.
- sys Send the uucp command to system sys to be executed there. (Note - this will only be successful if the remote machine allows the uucp command to be executed by /usr/lib/uucp/uuxqt.)

Uulog maintains a summary log of uucp and uux(1C) transactions in the file /usr/spool/uucp/LOGFILE by gathering information from partial log files named /usr/spool/uucp/LOG.*.?. (These files will only be created if the LOGFILE is being used by another process.) It removes the partial log files.

The options cause uucp to print logging information:

- sys Print information about work involving system sys.
- user Print information about work done for the specified user.

Uuname lists the uucp names of known systems. The -l option returns the local system name.

FILES

/usr/spool/uucp
 spool directory
/usr/spool/uucppublic
 public directory for receiving and sending
 (PUBDIR)
/usr/lib/uucp/*
 other data and program files

SEE ALSO

mail(1), uux(1C).
Uucp Implementation Description by D. A. Nowitz.

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin `/usr/spool/uucppublic` (equivalent to `~nuucp` or just `~`).

BUGS

All files received by `uucp` will be owned by `uucp`. The `-m` option will only work sending files or receiving a single file. (Receiving multiple files specified by special shell characters `?*[]` will not activate the `-m` option.)

QUESTIONS

`~nuucp` in last line of WARNING is inconsistent with page 1. |

NAME

uustat - uucp status inquiry and job control

SYNOPSIS

uustat [option] ...

HP-UX COMPATIBILITY

Level:

Data Communications - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Uustat will display the status of, or cancel, previously specified uucp commands, or provide general status on uucp connections to other systems. The following options are recognized:

- mch Report the status of accessibility of machine mch. If mch is specified as all, then the status of all machines known to the local uucp are provided.
- kjobn Kill the uucp request whose job number is jobn. The killed uucp request must belong to the person issuing the uustat command unless he is the super-user.
- chour Remove the status entries which are older than hour hours. This administrative option can only be initiated by the user uucp or the super-user.
- user Report the status of all uucp requests issued by user.
- sys Report the status of all uucp requests which communicate with remote system sys.
- ohour Report the status of all uucp requests which are older than hour hours.
- yhour Report the status of all uucp requests which are younger than hour hours.
- jall Report the status of all the uucp requests.
- v Report the uucp status verbosely. If this option is not specified, a status code is printed with each uucp request.

When no options are given, uustat outputs the status of all uucp requests issued by the current user. Note that only one of the options -j, -m, -k, -c, or the rest of other options may be specified.

For example, the command

```
uustat -uhdc -smhtsa -y72 -v
```

will print the verbose status of all uucp requests that were issued by user hdc to communicate with system mhtsa within

the last 72 hours. The meanings of the job request status are:

```

job-number user remote-system command-time status-time
status

```

where the status may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
00001	the copy failed, but the reason cannot be determined
00002	permission to access local file is denied
00004	permission to access remote file is denied
00010	bad <u>uucp</u> command is generated
00020	remote system cannot create temporary file
00040	cannot copy to remote directory
00100	cannot copy to local directory
00200	local system cannot create temporary file
00400	cannot execute <u>uucp</u>
01000	copy succeeded
02000	copy finished, job deleted
04000	job is queued

The meanings of the machine accessibility status are:

```

system-name time status

```

where time is the latest status time and status is a self-explanatory description of the machine status.

FILES

```

/usr/spool/uucp      spool directory
/usr/lib/uucp/L_stat  system status file
/usr/lib/uucp/R_stat  request status file

```

SEE ALSO

uucp(1C).
Uustat - A UUCP Status Inquiry Program, by H. Che.

NAME

uusub - monitor uucp network

SYNOPSIS

uusub [options]

HP-UX COMPATIBILITY

Level:

Data Communication - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Uusub defines a uucp subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

- asys Add sys to the subnetwork.
- dsys Delete sys from the subnetwork.
- l Report the statistics on connections.
- r Report the statistics on traffic amount.
- f Flush the connection statistics.
- uhr Gather the traffic statistics over the past hr hours.
- csys Exercise the connection to the system sys. If sys is specified as all, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

sys #call #ok time #dev #login #nack #other

where sys is the remote system name, #call is the number of times the local system tries to call sys since the last flush was done, #ok is the number of successful connections, time is the latest successful connect time, #dev is the number of unsuccessful connections because of no available device (e.g. ACU), #login is the number of unsuccessful connections because of login failure, #nack is the number of unsuccessful connections because of no response (e.g. line busy, system down), and #other is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

where sfile is the number of files sent and sbyte is the number of bytes sent over the period of time indicated in the latest uusub command with the -uhr option. Similarly, rfile and rbyte are the numbers of files and bytes received.

UUSUB(1M)

UUSUB(1M)

The command:

uusub -c all -u 24

is typically started by cron(1M) once a day.

FILES

/usr/spool/uucp/SYSLOG
 system log file
/usr/lib/uucp/L_sub
 connection statistics
/usr/lib/uucp/R_sub
 traffic statistics

SEE ALSO

uucp(1C), uustat(1C).

NAME

uuto, uupick - public UNIX-to-UNIX file copy

SYNOPSIS

uuto [options] source-files destination
uupick [-s system]

HP-UX COMPATIBILITY

Level:

Data Communication - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Uuto sends source-files to destination. Uuto uses the uucp(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!user

where system is taken from a list of system names that uucp knows about (see uname(1C)). User is the login name of someone on the specified system.

Two options are available:

- p Copy the source file into the spool directory before transmission.
- m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on system, where PUBDIR is a public directory defined in the uucp source. Specifically the files are sent to

PUBDIR/receive/user/mysystem/files.

The destined recipient is notified by mail(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, uupick searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from system: [file file-name] [dir dirname] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line> Go on to next entry.

d Delete the entry.

m [dir] Move the entry to named directory dir
(current directory is default).

a [dir] Same as **m** except moving all the files sent
from system.

p Print the content of the file.

q Stop.

EOT (control-d) Same as **q**.

!command Escape to the shell to do command.

***** Print a command summary.

Uupick invoked with the **-ssystem** option will only search the
PUBDIR for files sent from system.

FILES

PUBDIR /usr/spool/uucppublic public directory

SEE ALSO

mail(1), uuclean(1M), uucp(1C), uulog(1C), uuname(1C),
uustat(1C), uux(1C).

NAME

uux - unix to unix command execution

SYNOPSIS

uux [-] command-string

HP-UX COMPATIBILITY

Level:

Data Communication - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from uux. Many sites will permit little more than the receipt of mail (see mail(1)) via uux.

The command-string is made up of one or more arguments that look like a Shell command line, except that the command and file names may be prefixed by system-name!. A null system-name is interpreted as the local system.

File names may be one of

- (1) a full path name;
- (2) a path name preceded by "xxx" where xxx is a login name on the specified system and is replaced by that user's login directory;
- (3) anything else is prefixed by the current directory.

The - option will cause the standard input to the uux command to be the standard input to the command-string. For example, the command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"
```

will get the f1 files from the "usg" and "pwba" machines, execute a diff command and put the results in f1.diff in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

Uux will attempt to get all files to the execution system.

For files which are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!uucp b!/usr/file \(!/usr/file\)
```

will send a `uucp` command to system `'a'` to get `/usr/file` from system `'b'` and send it to system `'c'`.

`Uux` will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

FILES

<code>/usr/lib/uucp/spool</code>	spool directory
<code>/usr/lib/uucp/*</code>	other data and programs

SEE ALSO

`uuclean(1M)`, `uucp(1C)`.
`Uucp Implementation Description` by D. A. Nowitz

BUGS

Only the first command of a shell pipeline may have a `system-name!`. All other commands are executed on the system of the first command.
The use of the shell metacharacter `*` will probably not do what you want it to do. The shell tokens `<<` and `>>` are not implemented.

NAME

`vi`, `view` - screen oriented (visual) display editor based on `ex`

SYNOPSIS

`vi [-t tag] [-r] [+ command] [-l] [-wn] name ...`

HP-UX COMPATIBILITY

Level:

`vi` editor - HP-UX/STANDARD

Origin:

UCB

Remarks:

William Joy wrote the original `vi` editor. Mark Horton added macros to `visual` mode and is maintaining version 3.

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 Vi Quick Reference card and the Introduction to Display Editing with Vi provide full details on using `vi`.

FILES

See `ex(1)`.

SEE ALSO

`e(1)`, `edit(1)`, `ex(1)`, "Vi Quick Reference" card, "An Introduction to Display Editing with Vi".

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.

VI(1)

VI(1)

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.

WAIT(1)

WAIT(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 can't be waited for.

WALL(1M)

WALL(1M)

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.

WHAT(1)

WHAT(1)

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

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/ada/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)`.

WHODO(1M)

WHODO(1M)

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).

WRITE(1)

WRITE(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-username 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

intro - introduction to system calls and error numbers

SYNOPSIS

#include <errno.h>

HP-UX COMPATIBILITY**Level:**

This describes where in the HP-UX compatibility model this capability appears. The level given for each system call is the one for which the full semantics must be implemented. There may be sections entitled REDUCED SEMANTICS defining a reduced capability set that may be implemented if certain conditions are met. In a REDUCED SEMANTICS section, the first paragraph will summarize the reduction and the second will give the conditions under which the reduced semantics may or will be present.

Origin:

System III, HP, and UCB

DESCRIPTION

This section describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always -1; the individual descriptions specify the details. An error number is also made available in the external variable `errno`. `Errno` is not cleared on successful calls, so it should be tested only after an error has been indicated.

All of the possible error numbers are not listed in each system call description because many errors are possible for most of the calls. 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)`).
- 9 EBADF Bad file number
Either a file descriptor refers to no open file, or a read (respectively write) request is made to a file which is open only for writing (respectively reading).
- 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.
- 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 write on a directory.
- 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 pure-procedure program that is being
executed.
- 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.

DEFINITIONS**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.

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.

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 signaling of related processes; see `kill(2)`.

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 process upon termination of the process group leader; see `exit(2)` and `signal(2)`.

Real User ID and Real Group ID

Each user allowed on the system is identified by a positive integer called a real user ID.

Each user is also a member of a group. The group is identified by a positive integer called the real group ID.

An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

Effective User ID and Effective Group ID

An active process has an effective user ID and an effective group ID that are used to determine file access permissions (see below). The effective user ID and effective group ID are equal to the process's real user ID and real group ID respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group ID bit set; see `exec(2)`.

Super-user

A process is recognized as a super-user process and is granted special privileges if its effective user ID is 0.

Special Processes

The processes with a process ID of 0 and a process ID of 1 are special processes and are referred to as `proc0` and `proc1`.

`Proc0` is the scheduler. `Proc1` is the initialization process (`init`). `Proc1` is the ancestor of every other process in the system and is used to control the process structure.

Note that some implementations of HP-UX may not have an identifiable `proc0`. For these implementations, statements about `proc0` should be ignored.

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 0 (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)`.

Path Name and Path Prefix

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.

More precisely, a path name is a null-terminated character string constructed as follows:

```
<path-name>::=<file-name>|<path-prefix><file-name>|
<path-prefix>::=<rtprefix>|/<rtprefix>
<rtprefix>::=<dirname>|/<rtprefix><dirname>/
```

where <file-name> is a string of 1 to 14 characters other than the ASCII slash and null, and <dirname> is a string of 1 to 14 characters (other than the ASCII slash and null) that names a directory.

If a path name begins with a slash, the path search begins at the `root` directory. Otherwise, the search begins from the current working directory.

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.

Directory.

Directory entries are called links. By convention, a directory contains at least two links, `.` and `..`, referred to as `dot` and `dot-dot` respectively. `Dot` refers to the directory itself and `dot-dot` refers to its parent directory.

Root Directory and Current Working Directory.

Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. A process's root directory need not be the root directory of the root file system.

File Access Permissions.

Read, write, and execute/search permissions on a file are granted to a process if one or more of the following are

true:

The process's effective user ID is super-user.

The process's effective user ID matches the user ID of the owner of the file and the appropriate access bit of the "owner" portion (0700) of the file mode is set.

The process's effective user ID does not match the user ID of the owner of the file, and the process's group ID matches the group of the file and the appropriate access bit of the "group" portion (070) of the file mode is set.

The process's effective user ID does not match the user ID of the owner of the file, and the process's effective group ID does not match the group ID of the file, and the appropriate access bit of the "other" portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied.

HARDWARE DEPENDENCIES

Series 500:

Description of error 12 (ENOMEM) is not completely true, since Series 500 does not implement swap space.

The definition for **Special Processes** should be:

The process with a process ID of 1 is a special process and is referred to as proc1. Proc1 is the initialization process (init). Proc1 is the ancestor of every other process in the system and is used to control the process structure.

SEE ALSO

intro(3).

ACCESS(2)

ACCESS(2)

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. [EACCESS]

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. [EACCESS]

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.

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)`.

ALARM(2)

ALARM(2)

NAME

alarm - set a process's alarm clock; signal on timeout

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).

QUESTIONS

The use of alarm values greater than 765335 is not portable.

NAME

`brk`, `sbrk` - change data segment space allocation, memory 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:

`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 intrinsic.

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 causes it to increase by one page, and decreasing it by one byte does nothing. If your process' data area is not paged,

BRK(2)

BRK(2)

then the size of the process data area changes similarly in increments of 32 bytes.

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).

CHMOD(2)

CHMOD(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:



CHMOD(2)

CHMOD(2)

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. [EACCESS]

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:

Chmod only changes the mode of files created 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).

CHOWN(2)

CHOWN(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.

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. [EACCESS]

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:

Chown changes the owner and group of files created only

CHOWN(2)

CHOWN(2)

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

`chown(1)`, `chmod(2)`.

CHROOT(2)

CHROOT(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, .. can not 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.

CHROOT(2)

CHROOT(2)

SEE ALSO

chroot(1), chdir(2).

1

CLOSE(2)

CLOSE(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).

CREAT(2)

CREAT(2)

NAME

`creat` - create a new file or rewrite an existing one

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).

DUP(2)

DUP(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).

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

creat(2), close(2), exec(2), fcntl(2), open(2), pipe(2).

NAME

execl, execv, execl, 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 execl (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 transforms the calling process into a new process. The new process is constructed from an ordinary, executable file called the new process file. 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 process is overlaid by the new process.

Path points to a path name that identifies the new process file.

File points to the new process file. 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)). File may be a shell script, in which case /bin/sh is executed with file as the input.

Arg0, arg1, ..., argn are pointers to null-terminated character strings. These strings constitute the argument

list available to the new process. 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 process. 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 for the new process. Envp is terminated by a null pointer.

File descriptors open in the calling process remain open in the new process, 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 process will be set to terminate the new process. Signals set to be ignored by the calling process will be set to be ignored by the new process. Signals set to be caught by the calling process will be set to terminate new process; see signal(2).

If the set-user-ID mode bit of the new process file is set (see chmod(2)), exec sets the effective user ID of the new process to the owner ID of the new process file. Similarly, if the set-group-ID mode bit of the new process file is set, the effective group ID of the new process is set to the group ID of the new process file. The real user ID and real group ID of the new process remain the same as those of the calling process.

Profiling is disabled for the new process; see profil(2).

The new process also inherits the following attributes from the calling process:

- nice value (see nice(2))
- process ID
- parent process ID
- process group ID
- tty group ID (see exit(2) and signal(2))
- trate 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, cuptime, and cstime (see times(2))

Exec will fail and return to the calling process if one or more of the following are true:

One or more components of the new process file's path name do not exist. [ENOENT]

A component of the new process file's path prefix is not a directory. [ENOTDIR]

Search permission is denied for a directory listed in the new process file's path prefix. [EACCESS]

The new process file is not an ordinary file. [EACCESS]

The new process file mode denies execution permission. [EACCESS]

The new process file has the appropriate access permission, but has an invalid magic number in its header. [ENOEXEC]

The 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 process requires more memory than is allowed by the system-imposed maximum MAXMEM. [ENOMEM]

The number of bytes in the new process's argument list is greater than the system-imposed limit of 5120 bytes. [E2BIG]

The new process file is not as long as indicated by the size values in its header. [EFAULT]

Path, argv, or envp point to an illegal address. [EFAULT]

Path is null. [ENOENT]

RETURN VALUE

If exec returns to the calling process 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).

EXIT(2)

EXIT(2)

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 `stdio` buffers are flushed.

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 (see `init(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 processes that has a process group ID equal to that of the calling process.

EXIT(2)

EXIT(2)

_exit is equivalent to exit except that stdio buffers are not flushed.

HARDWARE DEPENDENCIES

Series 500:

An accounting record is **not** written on the accounting file, since acct is not currently supported.

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 cmds 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:

Filedes 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)*.

FORK(2)

FORK(2)

NAME

fork - create a new process

SYNOPSIS

int fork ()

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, cuptime, and cstime are set to 0; see times(2).

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]

HARDWARE DEPENDENCIES

Series 500:

Fork will also fail if there is not enough memory to create the new process. [ENOMEM]

Series 200:

FORK(2)

FORK(2)

Fork will also fail if there is not enough memory to create the new process. [ENOSPC]

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), vfork(2), wait(2).

NAME

getdrv - get driver name corresponding to major device number

SYNOPSIS

```
getdrv (major, drivename)
char drivename[16]
```

HP-UX COMPATIBILITY

Level:
HP-UX/?

Origin:
HP

DESCRIPTION

Getdrv returns the ASCII driver name of the driver that refers to the given major device number. This is the driver name that can be used in the mknod(1) command. Drivename will contain the null-terminated name of the driver, if it exists in the system. Major must be a number in the range 1 to 255.

BUGS

This intrinsic is available only on the Series 500. Its use will make your code non-transportable across HP-UX.

DIAGNOSTICS

-1 is returned if major is out of range, or if there is no driver in the system by the name drivename.

SEE ALSO

mknod(1), getmaj(2), mknod(2).

GETMAJ(2)

(Series 500 Only)

GETMAJ(2)

NAME

getmaj - get major device number corresponding to driver

SYNOPSIS

```
getmaj (drivername)  
char drivername[16]
```

HP-UX COMPATIBILITY

Level:
HP-UX/?

Origin:
HP

DESCRIPTION

Getmaj returns the major device number that refers to the named driver. This is the major device number that is to be used in mknod(2) to specify which driver is to be used for that device. Valid driver numbers are in the range 1 to 255. Getmaj returns -1 if the named driver is not currently resident in the system.

BUGS

This intrinsic is available only on the Series 500. Its use will make your code non-transportable across HP-UX.

DIAGNOSTICS

-1 is returned if drivername does not contain a valid driver name.

SEE ALSO

mknod(1), getdrv(2), mknod(2).

GETPID(2)

GETPID(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), intro(2), setpgrp(2), signal(2).

GETUID(2)

GETUID(2)

NAME

getuid, geteuid, getgid, getegid - get real user, effective user, real group, and 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

intro(2), setuid(2).

NAME

initv - initialize and optionally format a file system media

SYNOPSIS

```
initv(path,blocksize,bootsize,interleave,format);  
char *path;  
long blocksize,bootsize,interleave,format;
```

HP-UX COMPATIBILITY

Level:

HP-UX/?

Origin:

HP

Remarks:

This intrinsic is implemented only on the Series 500.



DESCRIPTION

This intrinsic allows for magnetic media to be optionally formatted (each physical record of the media is tested and spared if necessary) and initialized with the HP Structured Directory Format (SDF).

Path is a pointer to a null-terminated file name which is either a block or character special file describing the device that is to be initialized. This device must not be mounted.

Blocksize is the number of bytes per block (smallest amount of space transferred to or from the device). This should be a multiple of the physical record size.

Bootsize is the number of bytes which is to be allocated on the media for the operating system bootstrapping area. Systems which load the operating system off magnetic media can initialize the media with whatever boot size is necessary.

Interleave specifies which interleave factor is used for media formatting. Not all disc drives can have a variable interleave factor. Check the disc reference manual. Specifying a different interleave factor can improve throughput for some applications.

Format specifies that the media is to be tested for defective physical records and spared as necessary. This will happen whenever the value of format is non-zero. Formatting is normally not necessary for media that have been previously initialized.

This intrinsic is only executable by a process whose

INITV(2)

((Series 500 Only))

INITV(2)

effective user ID is that of the super-user.

RETURN VALUE

A successful execution will return zero (0). Failure will return -1, with errno set to indicate the nature of the failure. Refer to intro(2) for a list of possible errno settings.

SEE ALSO

sdfinit(1M).

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]

Include files other than sys/ioctl.h are appropriate for some devices.

RETURN VALUE

If an error has occurred, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

tty(4), Section 4 generally.

KILL(2)

KILL(2)

NAME

kill - send a signal to a process or a group of processes

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 whose process group ID is equal to the absolute value of pid.

KILL(2)

KILL(2)

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]

HARDWARE DEPENDENCIES

Series 500:

The process proc0 is not implemented.

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).

LINK(2)

LINK(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. [EACCESS]

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. [EACCESS]

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]

LINK(2)

LINK(2)

Path1 or path2 is null. [ENOENT]

1

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

link(1M), ln(1), unlink(2).

1

LSEEK(2)

LSEEK(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.

LSEEK(2)

LSEEK(2)

SEE ALSO

creat(2), dup(2), fcntl(2), open(2).

NAME

mknod - make a directory, or a special or ordinary file

SYNOPSIS

```
#include <mknod.h>
int mknod (path, mode, dev)
char *path;
int mode;
dev_t dev;
```

HP-UX COMPATABILITY

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. [EACCESS]

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), **mknod(1M)**, **chmod(2)**, **exec(2)**, **umask(2)**, **fs(5)**, **mknod(5)**.

MOUNT(2)

MOUNT(2)

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 file 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]

MOUNT(2)

MOUNT(2)

The device associated with spec is currently mounted.
[EBUSY]

Spec or dir 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(1M), umount(2).

NICE(2)

NICE(2)

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]

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 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 (only one of the first three flags below may 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]

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).

PAUSE(2)

PAUSE(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).

PIPE(2)

PIPE(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(3).

READ(2)

READ(2)

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:

File 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).)

SETPGRP(2)

SETPGRP(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.

RETURN VALUE

Setpgrp returns the value of the new process group ID.

SEE ALSO

exec(2), fork(2), getpid(2), intro(2), kill(2), signal(2).

QUESTIONS

Check to see if the new process group ID returned is constrained in any way.

SETUID(2)

SETUID(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), intro(2).

SIGNAL(2)

SIGNAL(2)

NAME

signal - specify what to do upon receipt of a signal

SYNOPSIS

```
#include <signal.h>

int (*signal (sig, func))()
int sig;
int (*func)();

func(signo, [ , code, scp ] )
int signo, code;
struct sigcontext *scp;
```

HP-UX COMPATABILITY

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
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 <u>WARNING</u> below)
SIGPWR	19	power fail (see <u>WARNING</u> 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 of 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 (see exit(2) for 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 (see intro(2)) 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 two additional (optional) parameters to signal handler routines. The parameters are:

signo signal number (same as standard UNIX).

code a word of information usually provided by the hardware.

scp a pointer to a machine dependent structure.

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. They are documented with the specific documentation for each implementation. It is permissible for **code** and **signo** to always be zero and 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.

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, including `SIGKILL`.
[EINVAL]

`Func` points to an illegal address. [EFAULT]

HARDWARE DEPENDENCIES

Series 500:

An accounting record will **not** be written on the accounting file, since accounting is not currently supported.

The Series 500 does not support making a "core image" in the current working directory of the receiving process.

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)`, `pause(2)`, `ptrace(2)`, `wait(2)`, `setjmp(3C)`.

WARNING

Two signals that behave differently than 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)

There is no guarantee that, in future releases of HP-UX, these signals will continue to behave as described below; they are included only for compatibility with other versions of UNIX. Their use in new programs is strongly discouraged.

For these signals, func is assigned one of three values: **SIG_DFL**, **SIG_IGN**, or a function address. The actions prescribed by these values of 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 proceeding 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**.

PORTABILITY CONSIDERATIONS

The optional information passed to the user's signal-catching function is machine dependent. Portable software should not use this feature.

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 COMPATABILITY

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 */
                        /* directory 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; this entry defined */
                        /* only for char or 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 system calls: creat(2), mknod(2), pipe(2), utime(2), and read(2).

st_mtime Time when data was last modified. Changed by the following system calls: creat(2), mknod(2), pipe(2), utime(2), and write(2).

st_ctime Time when file status was last changed. Changed by the following system calls: 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

chmod(2), chown(2), creat(2), link(2), mknod(2), time(2), unlink(2).

STIME(2)

STIME(2)

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 7 Compatibility - HP-UX/STANDARD

Origin:

Version 7

Remarks:

This call is preserved for backward compatibility with Bell Version 7. It provides as close an approximation as possible to the old Version 7 function. All new code should use the corresponding `ioctl` calls described in `tty(4)`.

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.

SYNC(2)

SYNC(2)

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 disk 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.

REDUCED SEMANTICS

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(1M).

TIME(2)

TIME(2)

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]

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).

TIMES(2)

TIMES(2)

NAME

times - get process and child process times

SYNOPSIS

```
long times (buffer)
struct tbuffer *buffer;
struct tbuffer {
    long utime;
    long stime;
    long ctime;
    long cstime;
}
```

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.

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.

Ctime is the sum of the utimes and cetimes of the child processes.

Cstime is the sum of the stimes and cstimes of the child processes.

Times will fail if buffer points to an illegal address.
[EFAULT]

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.

TIMES(2)

TIMES(2)

SEE ALSO

exec(2), fork(2), time(2), wait(2).

QUESTIONS

SYNOPSIS needs to be fixed to reflect reality.

ULIMIT(2)

ULIMIT(2)

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)`.

RETURN VALUE

Upon successful completion, a non-negative value is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

SEE ALSO

`brk(2)`, `write(2)`.

QUESTIONS

Is 512-byte blocks (under DESCRIPTION) correct?

UMASK(2)

UMASK(2)

NAME

umask - set and get 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 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.

RETURN VALUE

The previous value of the file mode creation mask is returned.

SEE ALSO

mkdir(1), mknod(1M), sh(1), chmod(2), creat(2), mknod(2), open(2).

UMOUNT(2)

UMOUNT(2)

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.

UMOUNT(2)

UMOUNT(2)

SEE ALSO

mount(1M), mount(2).

NAME

uname - get name of current HP-UX system

SYNOPSIS

```
#include <sys/utsname.h>
```

```
int uname (name)
struct utsname *name;
```

HP-UX COMPATABILITY

Level:

HP-UX/RUN ONLY

Origin:

System III

DESCRIPTION

Uname stores information identifying the current UNIX system in the structure pointed to by name.

Uname uses the structure defined in <sys/utsname.h>:

```
struct utsname {
    char sysname[9];
    char nodename[9];
    char release[9];
    char version[9];
};
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 sethostname(1). Release contains the release number of the operating system, e.g. "1.0" or "3.0.1". Version contains the product or product family identification for the system on which the operating system runs.

Uname will fail if name points to an invalid address.
[EFAULT]

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

sethostname(1), sethostname(2), uname(1).

QUESTIONS

Get new version of structure defined in <sys/utsname.h>,

UNAME(2)

UNAME(2)

under DESCRIPTION.

1

UNLINK(2)

UNLINK(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 be 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]

UNLINK(2)

UNLINK(2)

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

The super-user cannot unlink a directory unless it is empty and has a link count of 1.

NAME

ustat - get file system statistics

SYNOPSIS

```
#include <sys/types.h>
#include <ustat.h>
```

```
int ustat (dev, buf)
int 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 that includes the following elements:

```
daddr_t f_tfree;          /* Total free blocks */
ino_t    f_tinode;         /* Number of free inodes */
char     f_fname[61];      /* Filsys name */
char     f_fpack[61];      /* Filsys pack name */
```

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]

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. [EACCESS]

The effective user ID is not super-user and not the owner of the file and times is not **NULL**. [EPERM]

UTIME(2)

UTIME(2)

The effective user ID is not super-user and not the owner of the file and times is **NULL** and write access is denied. [EACCESS]

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).

WAIT(2)

WAIT(2)

NAME

wait - wait for child process to stop or 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 and if the child process terminated, status identifies the cause of termination and pass 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 0177.

If the child process terminated due to an 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

WAIT(2)

WAIT(2)

process is set to 1. This means the initialization process inherits the child processes; see inrq(2).

Wait will fail and return immediately if one or more of the following are true:

The calling process has no existing unwaited-for child processes. [ECHILD]

Sigl_log points to an illegal address. [EFAULT]

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).

WRITE(2)

WRITE(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]

WRITE(2)

WRITE(2)

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. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes will give a failure return (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.

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)).

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 UNIX 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`. They are automatically loaded as needed by the FORTRAN compiler `f77(1)`. 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), f77(1), ld(1), nm(1), intro(2), 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 `intro(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,
 $a0 = 38 \times 64 \text{ sup } 0 + 2 \times 64 \text{ sup } 1 = 166$

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.

ABORT(3C)

ABORT(3C)

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.

ABS(3C)

ABS(3C)

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 200:
The largest negative integer returns itself.

Series 500:
The largest negative integer returns itself.

SEE ALSO

fabs(3M).

*

ASSERT(3X)

ASSERT(3X)

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.

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);
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 a huge negative value, and sets errno to EDOM

NAME

toupper, tolower, toupper, tolower, toascii - character translation

SYNOPSIS

```
#include <ctype.h>
```

```
int toupper (c)
int c;
```

```
int tolower (c)
int c;
```

```
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.

CONV(3C)

CONV(3C)

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.

CRYPT(3C)

CRYPT(3C)

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>**.

NOTES

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\n0

Localtime and gmtime return pointers to structures containing the broken-down time. Localtime corrects for the time zone and possible daylight savings time; gmtime converts directly to GMT, which is the time the UNIX 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(3) */
```

```

    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 Savings 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.

CTYPE(3C)

CTYPE(3C)

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)).

<u>isalpha</u>	<u>c</u> is a letter
<u>isupper</u>	<u>c</u> is an upper case letter
<u>islower</u>	<u>c</u> is a lower case letter
<u>isdigit</u>	<u>c</u> is a digit [0-9]
<u>isxdigit</u>	<u>c</u> is a hexadecimal digit [0-9], [A-F] or [a-f]
<u>isalnum</u>	<u>c</u> is an alphanumeric
<u>isspace</u>	<u>c</u> is a space, tab, carriage return, new-line, vertical tab, or form-feed
<u>ispunct</u>	<u>c</u> is a punctuation character (neither control nor alphanumeric)
<u>isprint</u>	<u>c</u> is a printing character, code 040 (space) through 0176 (tilde)
<u>isgraph</u>	<u>c</u> is a printing character, like <u>isprint</u> except false for space

CTYPE(3C)

CTYPE(3C)

isctrl

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/RUN ONLY

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>**.

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.

SEE ALSO

getlogin(3C), getpwuid(3C).

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.

The name cuserid is rather a misnomer.

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;
```

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.

Ecvt is identical to ecvt, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by *_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.

END(3C)

END(3C)

NAME

`end`, `etext`, `edata` - last locations in program

SYNOPSIS

```
extern end;
extern etext;
extern 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.

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 if it is remembered that they should be prefixed by `_`.

HARDWARE DEPENDENCIES

Series 500:

`end` is the lowest data area address available to the user. `etext` is the lowest unused code segment number. (This value is of little, if any, use to anyone.) `edata` is the first address following the initialized (indirect) data.

SEE ALSO

`brk(2)`, `malloc(3C)`.

NAME

`exp`, `log`, `pow`, `sqrt` - exponential, logarithm, power, square root functions

SYNOPSIS

```
#include <math.h>
```

```
double exp (x)
double x;
```

```
double log (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`.

`Pow` returns `xy`.

`Sqrt` returns the square root of `x`.

SEE ALSO

`intro(2)`, `hypot(3M)`, `sinh(3M)`.

DIAGNOSTICS

`Exp` returns a huge value when the correct value would overflow. A truly outrageous argument 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**.

EXP (3M)

EXP (3M)

Sqrt returns 0 and sets errno to **EDOM** when x is negative.

QUESTIONS

log10??

FCLOSE(38)

FCLOSE(36)

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)`.

FERROR(3S)

FERROR(3S)

NAME

error, 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).

FOPEN(3S)

FOPEN(3S)

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 ill-formed.

Fdopen returns a **NULL** if there are too many open files, or if the arguments are ill-formed.

FREAD(3S)

FREAD(3S)

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), putc(3S), gets(3S), puts(3S), printf(3S), scanf(3S).

FREXP(3C)

FREXP(3C)

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 COMPATABILITY

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 such that value = x*2**n indirectly through eptr.

Ldexp returns the quantity value*(2**exp). If this quantity overflows then +HUGE or -HUGE 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, it is a magic cookie 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.

GAMMA(3M)

GAMMA(3M)

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 $\ln|\lambda(x)|$. The sign of $\lambda(x)$ is returned in the external integer signgam. The following C program fragment might be used to calculate λ :

```
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; this is wiser than, but incompatible with, the older getchar(3S). Because it is implemented as a macro, getc treats *

GETC(38)

GETC(38)

incorrectly a stream argument with side effects. In particular, **getc(*f++)**; doesn't work sensibly.

GETENV(3C)

GETENV(3C)

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/RUN ONLY

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(3) */
    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

GETGRENT(3C)

GETGRENT(3C)

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.

GETLOGIN(3C)

GETLOGIN(3C)

NAME

getlogin - get login name

SYNOPSIS

```
char *getlogin ( );
```

HP-UX COMPATIBILITY

Level:

HP-UX/RUN ONLY

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

Returns **NULL** if name not found.

BUGS

The return values point to static data whose content is overwritten by each call.

GETOPT(3C)

GETOPT(3C)

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. Because optind is external, it is normally initialized to zero 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.

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.

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;
                bufsize = 512;
                break;
            case '?':
                errflg++;
        }
    if (errflg) {
        fprintf (stderr, "usage: . . . ");
        exit (2);
    }
    for( ; optind < argc; optind++) {
        .   if (access (argv[optind], 4)) {
        .
    }

```

BUGS

An option letter cannot be a colon or a question mark. |

GETPASS(3C)

GETPASS(3C)

NAME

getpass - read a password

SYNOPSIS

```
char *getpass (prompt)
char *prompt;
```

HP-UX COMPATIBILITY

Level:
HP-UX/RUN ONLY

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.

GETPW(3C)

GETPW(3C)

NAME

getpw - get name from UID

SYNOPSIS

```
getpw (uid, buf)
int uid;
char *buf;
```

HP-UX COMPATIBILITY

Level:
HP-UX/RUN ONLY

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

Non-zero return on error.

GETPWENT(3C)

GETPWENT(3C)

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/RUN ONLY

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;
    int  pw_uid;
    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

GETPWENT(3C)

GETPWENT(3C)

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 seipwent.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

GETS(3S)

GETS(3S)

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.

NOTE

Gets deletes the new-line ending its input, but fgets keeps it.

HYPOT(3M)

HYPOT(3M)

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).

LOGNAME(3X)

LOGNAME(3X)

NAME

logname - login name of user

SYNOPSIS

char *logname();

HP-UX COMPATIBILITY

Level:

HP-UX/RUN ONLY

Origin:

System III

DESCRIPTION

Logname returns a pointer to the null-terminated login name; it extracts the **\$LOGNAME** variable from the user's environment.

This routine is kept in **/lib/libPW.a**.

FILES

/etc/profile

SEE ALSO

env(1), login(1), profile(5), environ(7).

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, esize)

unsigned elem, esize;



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, grave disorder will result if the space assigned by malloc is overrun or if some random number is handed to free.

Malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing 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.

MALLOC(3C)

MALLOC(3C)

Calloc allocates space for an array of n_{elem} elements of size el_{size}. 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.

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 name with six trailing Xs, 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 an existing file. If there are less than 6 Xs, 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.

NLIST(3C)

NLIST(3C)

NAME

nlist - get entries from name list

SYNOPSIS

```
#include <nlist.h>
nlist (file-name, nl)
char *file-name;
struct nlist nll[];
```

HP-UX COMPATABILITY

Level:

HP-UX/RUN ONLY

Origin:

System III

Remarks:

This is dependent on the **a.out** format, which is in turn architecture dependent. Thus, there may be variations in the results of this routine across HP-UX systems.

DESCRIPTION

Nlist selectively extracts the values corresponding to the elements of a list of symbols from the designated file. The file must have the organization and symbol table described for an **a.out** file in **a.out.h(5)**. The information is extracted from the symbol table used by the loader, **ld(1)**.

The list of names provided by the caller is in an array of structures, each containing a pointer to a string giving the name for which information is desired. The rest of the structure contains a type field, a value field, and possibly other machine specific information which will be filled in by the **nlist** call. The list is terminated with a null name pointer. The structure updated by this call is described in **nlist(5)**. The name pointer in the **nlist** structure will always be the first field.

When **nlist** is called, the file's symbol table is searched for each name in the **nlist** array. If the name is found, all symbol table information, including the type and value of the name, is inserted into the appropriate **nlist** fields.

On machines which have such a file, this subroutine is useful for examining the system name list kept in the file **/unix**. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5), **nlist(5)**.

DIAGNOSTICS

All nlist structure fields are set to 0 if the file cannot
be found or if it is not a valid object file containing a
linker symbol table.

HP-UX STANDARDIZATION ISSUES

This utility deals with the symbol table as used by the
linker; extended symbol tables for debugging are not
addressed.

NAME

perror, sys_errlist, sys_nerr, errno - system error messages

SYNOPSIS

```
perror (s)
char *s;

int sys_nerr;
char *sys_errlist[];

int errno;
```

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.

SEE ALSO

intro(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 "popened" 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).

PRINTF(3S)

PRINTF(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 COMPATABILITY

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 args 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 args. The results are undefined if there are insufficient args for the format. If the format is exhausted while args remain, the excess args 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 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 args 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 `[-lddd.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 `[-ld.ddd[+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 `pi` 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;

putc (c)

fputc (c, stream)
FILE *stream;

putw (w, stream)
int w;
FILE *stream;
```

HP-UX COMPATABILITY

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(3B)`.

SEE ALSO

`ferror(3B)`, `fopen(3B)`, `fwrite(3B)`, `getc(3B)`, `printf(3B)`, `puts(3B)`.

DIAGNOSTICS

These functions return the constant `EOF` upon error. Since this is a good integer, `ferror(3B)` should be used to detect `putc` errors.

Line buffering may cause confusion or malfunctioning of programs which use standard i/o routines but use `read(2)` themselves to read from standard input. 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 going off and computing 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.

PUTPWENT(3C)

PUTPWENT(3C)

NAME

putpwent - write password file entry

SYNOPSIS

```
#include <pwd.h>
```

```
int putpwent (p, f)  
struct passwd *p;  
FILE *f;
```

HP-UX COMPATIBILITY

Level:

HP-UX/RUN ONLY

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)), putpwuid 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.

PUTS(3S)

PUTS(3S)

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.

DIAGNOSTICS

Both routines return **EOF** on error.

SEE ALSO

ferror(3S), fopen(3S), fwrite(3S), gets(3S), printf(3S),
putc(3S).

NOTES

Puts appends a new-line, fputs does not.

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 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.

SEE ALSO

sort(1), bsearch(3C), lsearch(3C), strcmp(3C).

BUGS

If width is zero, a divide-by-zero error is generated.

RAND(3C)

RAND(3C)

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 232 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 you like as argument.

HARDWARE DEPENDENCIES

Series 200:

Rand returns successive pseudo-random numbers in the range from 0 to $2^{15} - 1$.

Series 500:

Rand returns successive pseudo-random numbers in the range from 0 to $2^{31} - 1$.

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.
- 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 **%ls**. 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.

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

atoi(3C), **getc(3S)**, **printf(3S)**.

NOTE

Trailing white space (including a new-line) is left unread unless matched in the control string.

DIAGNOSTICS

These functions return **EOF** on end of input and a short count for missing or illegal data items.

SCANF (3S)

SCANF (3S)

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

SETBUF(3S)

SETBUF(3S)

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.

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, with the exception of register variables, which have an undefined value. Relying on the value of a register variable after a long jump renders the users's code non-portable.

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.

SLEEP(3C)

SLEEP(3C)

NAME

sleep - suspend execution for interval

SYNOPSIS

unsigned sleep (seconds)
unsigned 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 alarm signal, the process sleeps only until the alarm signal would have occurred, and the caller's alarm catch routine is executed just before the sleep routine returns, but if the sleep time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening sleep.

SEE ALSO

alarm(2), pause(2), signal(2).

NAME

stdio - standard buffered input/output stream file package

SYNOPSIS

```
#include <stdio.h>
FILE *stdin, *stdout, *stderr;
```

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

`open(2)`, `close(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 usually cause grave disorder,
possibly including program termination. Individual function
descriptions describe the possible error conditions.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen,
strchr, strchr, 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. The comparison is done using (signed) char type.

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*.

Strlen returns the number of non-null characters in *s*. (The "length" of a string does not include the null terminator.)

Strchr (**strchr**) 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 (**strspn**) 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.

STRING(3C)

STRING(3C)

When no token remains in s1, a **NULL** is returned.

HARDWARE DEPENDENCIES

Series 500:

n is limited to about 500 Mbytes.

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 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.

SYSTEM(3S)

SYSTEM(3S)

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

tmpfile - create a temporary file

SYNOPSIS

```
#include <stdio.h>
```

```
FILE *tmpfile ()
```

HP-UX COMPATIBILITY

Level:

HP-UX/RUN ONLY

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/RUN ONLY

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 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 .

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.

TRIG(3M)

TRIG(3M)

The approximate limits are:

HP 9000 Series 500	1.49E08
HP 9000 Series 200	1.49E08

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 can't 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 drivers directly are at higher than average risk of not being portable. HP will attempt to document variations in behavior, but this is not always feasible, thus HP cannot guarantee portability of such programs.

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 HP model numbers, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding HP-UX device driver are discussed where applicable.

BUGS

While the names of the entries generally refer to model numbers, in certain cases these names are seemingly arbitrary for various historical reasons.

CONSOLE(4)

CONSOLE(4)

NAME

console - super-user terminal

HP-UX COMPATIBILITY

Level:

HP-UX/?

Origin:

HP

DESCRIPTION

Console is the terminal (on multi-user systems) to which all super-user messages, warnings, and information are sent.

FILES

/dev/console

SEE ALSO

tty(4).

QUESTIONS

Need more info!

NAME

aurora, magic, odyssey - cathode ray tubes for HP 9000 Model 20

HP-UX COMPATIBILITY

Level:

Origin:
HP

DESCRIPTION

The CRT devices for the HP 9000 Model 20 are:

HP 98780B Monochrome CRT (aurora)

HP 98760A Monochrome CRT (magic)

HP 98770B Color CRT (odyssey)

They have the following characteristics:

	98770B	98780B
Horizontal Dimension	560 Pixels 184.79mm 7.28 Inches 123.1 GDUs	560 Pixels 184.79mm 7.28 Inches 123.1 GDUs
Vertical Dimension	455 Pixels 150.14mm 5.91 Inches 100 GDUs	455 Pixels 150.14mm 5.91 Inches 100 GDUs
Diagonal	238.10mm 9.38 Inches 158.6 GDUs	238.10mm 9.38 Inches 158.6 GDUs
Aspect Ratio	1.23	1.23
Alpha Display Lines	30	30
Characters per Line	80	80
Alpha Display Colors	White Red Yellow Green Cyan Blue Magenta Black	White
Alpha Memory Planes	1	1
Graphics Memory Planes	3	1
Color Mapper	No	No
Light Pen	Optional	Optional
Softkeys	Yes	Yes

CRTS(4)

(Series 500 Only)

CRTS(4)

FILES

/dev/aurora, dev/magic, /dev/odyssey

NAME

aurora, magic, odyssey - cathode ray tubes for HP 9000 Model 20

HP-UX COMPATIBILITY

Level:

Origin:

HP

DESCRIPTION

The CRT devices for the HP 9000 Model 20 are:

HP 98780B Monochrome CRT (aurora)

HP 98760A Monochrome CRT (magic)

HP 98770B Color CRT (odyssey)

They have the following characteristics:

	98770B	98780B
Horizontal Dimension	560 Pixels 184.79mm 7.28 Inches 123.1 GDUs	560 Pixels 184.79mm 7.28 Inches 123.1 GDUs
Vertical Dimension	455 Pixels 150.14mm 5.91 Inches 100 GDUs	455 Pixels 150.14mm 5.91 Inches 100 GDUs
Diagonal	238.10mm 9.38 Inches 158.6 GDUs	238.10mm 9.38 Inches 158.6 GDUs
Aspect Ratio	1.23	1.23
Alpha Display Lines	30	30
Characters per Line	80	80
Alpha Display Colors	White Red Yellow Green Cyan Blue Magenta Black	White
Alpha Memory Planes	1	1
Graphics Memory Planes	3	1
Color Mapper	No	No
Light Pen	Optional	Optional
Softkeys	Yes	Yes

NAME

disc - direct disc access

HP-UX COMPATABILITY

Level:

HP-UX/RUN ONLY

Origin:

HP

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 words 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 IO (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.

PORTABILITY CONSIDERATIONS

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

hp7908, hp7911, hp7912, linus - Command Set '80 devices

HP-UX COMPATIBILITY

Level:

Origin:

HP

DESCRIPTION

The Command Set '80 device drivers support the following mass storage devices:

HP 7908P 16.5 Mbyte Winchester disc/tape drive

HP 7911P 28.1 Mbyte Winchester disc/tape drive

HP 7912P 65.6 Mbyte Winchester disc/tape drive

HP 88140L/SDC-600 Data Cartridges

The CS80 devices are organized as follows:

Disc Drive Model	Surfaces Per Volume	Tracks Per Surface	Sectors Per Track	Bytes Per Sector	Total Capacity
7908P	5	370	35	256	16 576 000
7911P	3	572	64	256	28 114 944
7912P	7	572	64	256	65 601 536
Tape Cartridge Model	Feet of Tape per Cartridge	Tracks Per Surface	Blocks per Tape	Bytes per Block	Total Capacity
88140S	150	1	16352	1024	16 744 448
88140L	600	1	65408	1024	66 977 792

FILES

/dev/hp7908, /dev/hp7911, /dev/hp7912, /dev/linus

By judiciously choosing `mt` files, it is possible to read and write multi-file tapes.

A tape treated as a blocked 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.

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 */
```

NAME

magtape - magnetic tape interface and controls

HP-UX COMPATABILITY

Level:

Magnetic Tape Support -- HP-UX/RUN ONLY

Origin:

UCB and HP

DESCRIPTION

The files `mt*` refer to magnetic tapes, usually the 7970E, 7971, or 7976. The files `/dev/mt*` refer to these 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.

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.

NULL(4)

NULL(4)

NAME

null - the null file

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

```

/* 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 {
    short    mt_type;        /* type of magtape dev. */

    /* The following two registers are device dependent */

    short    mt_dsreg;       /* drive status register */
    short    mt_erreg;       /* error register */

    /* end device-dependent registers */

    short    mt_resid;       /* residual count */

    /* 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 byte
 */

#define MT_ISTS      01
#define MT_ISHT      02
#define MT_ISTM      03
#define MT_IS7970E   04

```

FILES

```

/dev/mt*
/dev/rmt*

```

PRINTERS(4)

PRINTERS(4)

NAME

hp2631 - HP 2631 line printer

HP-UX COMPATIBILITY

Level:

Origin:

HP

DESCRIPTION

The external printing devices are:

HP 2631 Printer

FILES

/dev/hp2631

SEE ALSO

thermal(4)

OMNI(4)

OMNI(4)

NAME

omni - Internal 5 1/4 Inch Flexible Disc Drive

HP-UX COMPATIBILITY

Level:

HP-UX/?

Origin:

HP

DESCRIPTION

Omni is the special file associated with the internal 5 1/4 inch flexible disc drive.

FILES

/dev/omni

NAME

tty - general terminal interface

HP-UX COMPATABILITY

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 `setpgid(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.

THERMAL(4)

(Series 500 Only)

THERMAL(4)

NAME

thermal - internal thermal printer for HP 9000 Model 20

HP-UX COMPATIBILITY

Level:

Origin:

HP

DESCRIPTION

Thermal is the special file associated with the internal printer for the HP 9000 Model 20.

FILES

/dev/thermal

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 can not 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>`:



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-I 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

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:

```

#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

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	
UTDLY	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 <code>termio</code> structure referenced by <code>arg</code> .
TCSETA	Set the parameters associated with the terminal from the structure referenced by <code>arg</code> . 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 <code>arg</code> is 0, then send a break (zero bits for at least 0.25 seconds).	
TCXONC	Start/stop control. If <code>arg</code> is 0, suspend output; if 1, restart suspended output.	
TCFLSH	If <code>arg</code> is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.	

FILES

```
/dev/tty
/dev/tty*
/dev/console
```

SEE ALSO

`stty(1)`, `ioctl(2)`.

BUGS

European modems are not properly supported.

HP-UX STANDARDIZATION ISSUES

Because various HP-UX implementations use non-serial interfaces which look like terminals (e.g. internal CRTs) or 'smart cards' which cannot implement the capabilities described above exactly, not all the systems can meet the standard stated above exactly. Each implementation is required to state any deviations from the standard as part of its system specific documentation.

INTRO(5)

INTRO(5)

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**.

A.OUT(5)

A.OUT(5)

NAME

a.out - assembler and link editor output

HP-UX COMPATABILITY

Level:

Use: HP-UX/RUN ONLY

Header: HP-UX/DEVELOPMENT

Origin:

HP

Remarks:

There will be separate manual pages for each
implementation that wishes to describe its a.out format
to its customers.

DESCRIPTION

The a.out (i.e., object file) format is completely machine-
dependent except for the first word, which contains a magic
number as defined in magic(5).

SEE ALSO

magic(5)

NAME

ar - archive file format

SYNOPSIS

```
#include <ar.h>
```

HP-UX COMPATABILITY

Level:

HP-UX/STANDARD

Origin:

System III

DESCRIPTION

The archive command `ar` is used to combine several files into one. 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(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.h(5)`.

CHECKLIST(5)

CHECKLIST(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(1M) command.

SEE ALSO

fsck(1M).

NAME

cpio - format of cpio archive

HP-UX COMPATABILITY

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 in an element of the array of varying length structures, archive, 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 on the tape 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

dir - format of directories

SYNOPSIS

```
#include <sys/dir.h>
```

**HP-UX COMPATIBILITY**

Level:

HP-UX/RUN ONLY

Origin:

System III

Requires:

HP-UX permits several file system formats. This is for the "Bell file system". Appropriate pages are available for the other file systems.

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry (see [fs\(5\)](#)). The structure of a directory entry as given in the include file is:

```
#ifndef DIRSIZ
#define DIRSIZ      14
#endif
struct direct
{
    ino_t d_ino;
    char d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are for `.` and `..`. The first is an entry for the directory itself. The second is for the parent directory. The meaning of `..` is modified for the root directory of the master file system; there is no parent, so `..` has the same meaning as `..`.

SEE ALSO

[fs\(5\)](#).

NAME

file system - format of system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/filsys.h>
#include <sys/fblk.h>
```

HP-UX COMPATABILITY

Level:

Bell File System -- HP-UX/RUN ONLY

Origin:

System III

DESCRIPTION

HP-UX defines a small number of file system media formats for compatibility with other HP operating systems. This page describes the

Every BFS file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of blocks the size of which is either 512 or 1024 bytes, depending on the implementation. Block 0 is unused and is available to contain a bootstrap program or other information.

Block 1 is the super-block. The format of a super-block is given as 'struct filsys' in `/usr/include/sys/filsys.h` which is copied below:

/* fundamental constants of the filesystem */

```
#define BSIZE 1024 /* or 512; block size (bytes) */
#define BMASK (BSIZE-1) /* BSIZE-1 */
```

```
#define NINDIR (BSIZE/sizeof(daddr_t))
#define NMASK (NINDIR-1) /* NINDIR-1 */
```

```
#define INOPB (BSIZE/sizeof(ino_t)) /* inodes/block */
#define INOMOD (INOPB-1) /* offset val */
```

```
#define ROOTINO ((ino_t)2) /* i-no. of all roots */
#define SUPERB ((daddr_t)1) /* block no. of super blk */
```

```
#define DIRSIZ 14 /* max chars/directory */
```

```
#if BSIZE==1024
#define INOSHFT 4 /* LOG2(INOPB) */
#define BSHIFT 10 /* LOG2(BSIZE) */
#define NSHIFT 8 /* LOG2(NINDIR) */
#define NICINOD 250 /* no. of superblock inodes */
```

```

#define NICFREE 100    /* no. of superblock free blks */
#endif

#if BSIZE==512
#define INOSHFT 3      /* LOG2(INOPB) */
#define BSHIFT 9      /* LOG2(BSIZE) */
#define NSHIFT 7      /* LOG2(NINDIR) */
#define NICINOD 100   /* no. of superblock inodes */
#define NICFREE 50    /* no. of superblock free blks */
#endif

/* Some macros for units conversion */

/* Core clicks to disk blocks */

#define ctod(x) (x*(BSIZE/PAGESIZE))

/* inumber to disk address */

#define itod(x) (daddr_t)((((unsigned)x+INOPB*2-1)>>INOSHFT))

/* inumber to disk offset */

#define itoo(x) (int)((((unsigned)x+INOPB*2-1)&INOMOD))

/* Structure of the super-block */

struct filsys
{
    long    s_isize;      /* i-list size (blocks) */
    daddr_t s_fsize;      /* volume size (blocks) */
    long    s_nfree;      /* no. of addresses in s_free */
    daddr_t s_free[NICFREE]; /* free blk list */
    long    s_ninode;     /* no. i-nodes in s_inode */
    ino_t    s_inode[NICINOD]; /* free i-node list */
    char    s_flock;      /* free list lock */
    char    s_ilock;      /* i-list lock */
    char    s_fmod;       /* super blk modified flag */
    char    s_ronly;      /* mounted read-only flag */
    time_t   s_time;      /* last super blk update */
    short    s_dinfo[4];  /* device info */
    daddr_t s_tfree;      /* total free blocks */
    ino_t    s_tinode;     /* total free inodes */
    char    s_fname[6];   /* file sys name */
    char    s_fpack[6];   /* file sys pack name */
};

```

S_isize is the address of the first data block after the i-list; the i-list starts just after the super-block, namely in block 2; thus the i-list is s_isize-2 blocks long.

S_fsize is the first block not potentially available for allocation to a file. These numbers are used by the system

to check for bad block numbers; if an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The `s_free` array contains, in `s_free[1]`, ..., `s_free[s_nfree-1]`, up to `NICFREE-1` block numbers of free blocks. `s_free[0]` is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to `NICFREE`) of free-block numbers listed in the next `NICFREE` longs of this chain member. The structure which defines this is `fbk` in file `/usr/include/sys/fblk.h`. The first of these `NICFREE` blocks is the link to the next member of the chain.

```
struct fblk
{
    long    df_nfree;          /* was int on VAX! */
    daddr_t df_free[NICFREE];
};
```

To allocate a block: decrement `s_nfree`, and the new block is `s_free[s_nfree]`. If the new block number is 0, there are no blocks left, so give an error. If `s_nfree` became 0, read in the block named by the new block number, replace `s_nfree` by its first word, and copy the block numbers in the next `NICFREE` longs into the `s_free` array. To free a block, check if `s_nfree` is `NICFREE`; if so, copy `s_nfree` and the `s_free` array into it, write it out, and set `s_nfree` to 0. In any event set `s_free[s_nfree]` to the freed block's number and increment `s_nfree`.

`s_ifree` is the total free blocks available in the file system.

`s_ninode` is the number of free i-numbers in the `s_inode` array. To allocate an i-node: if `s_ninode` is greater than 0, decrement it and return `s_inode[s_ninode]`. If it was 0, read the i-list and place the numbers of all free inodes (up to `NICINOD`) into the `s_inode` array, then try again. To free an i-node, provided `s_ninode` is less than `NICINOD`, place its number into `s_inode[s_ninode]` and increment `s_ninode`. If `s_ninode` is already `NICINOD`, do not bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

`s_tinode` is the total free inodes available in the file system.

S_flock and s_ilock are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of s_fmod on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

S_ronly is a read-only flag to indicate write-protection.

S_time is the last time the super-block of the file system was changed, and is a 32 bit representation of the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the s_time of the super-block for the root file system is used to set the system's idea of the time unless a true real time clock is present.

S_fname is the name of the file system and s_fpack is the name of the pack.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-node i is therefore located in the block given by the macro itod(i), and begins itoo(i) bytes from its start. I-node 1 is reserved for future use. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. For the format of an inode and its flags, see inode(5).

FILES

/usr/include/sys/filsys.h
/usr/include/sys/stat.h

SEE ALSO

fsck(1M), fsdb(1M), mkfs(1M), inode(5).

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:

- ttabs** 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.

ssize 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.

mmargin 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

Doesn't work with vi(1) and ex(1).

GROUP(5)

GROUP(5)

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.

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)`.

INITTAB(5)

INITTAB(5)

NAME

inittab - control information for init

HP-UX COMPATIBILITY

Level:

HP-UX/RUN ONLY

Origin:

System III

DESCRIPTION

When a state is entered, init reads the file /etc/inittab. Lines in this file have the format:

state:id:flags:command

All lines in which the state field match init's current state are recognized. If a process is active under the same two character id as a recognized line, it may be terminated (signal 15), killed (signal 9), or both by including the flags t and k in the order desired. The signal is sent to all processes in the process group associated with the id. The command field is saved for later execution. The flag c requires the command to be continuously reinvoked whenever the process with that id dies. Otherwise the command is invoked a maximum of one time in the current state.

FILES

/etc/inittab

NAME

magic - magic numbers for HP-UX implementations

SYNOPSIS

```
#include <magic.h>
```

HP-UX COMPATABILITY

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 system_id;
    unsigned short 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), ld(1), a.out(5), ar(5)

NAME

mknod - create a special file entry

SYNOPSIS

```
#include <mknod.h>
```

HP-UX COMPATABILITY

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 minimum possible number of digits of the resulting string. (Splitting a field across unnecessary digits for the sake of packing is prohibited.)

MNTTAB(5)

MNTTAB(5)

NAME

mnttab - mounted file system table

SYNOPSIS

```
struct mnttab {
    char    mt_dev[10];
    char    mt_filsys[10];
    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(1M) command.

Each entry is 26 bytes in length; the first 10 bytes are the null-padded name of the place where the special file is mounted; the next 10 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** located in /usr/src/uts/cf/conf.c, which defines the number of allowable mounted special files.

SEE ALSO

mount(1M).

NAME

nlist - nlist structure format

SYNOPSIS

```
#include <nlist.h>
```

HP-UX COMPATIBILITY

Level:
HP-UX/DEVELOPMENT



Origin:
System III

Remarks:
This command is dependent on the **a.out** format, which is
in turn architecture dependent.

DESCRIPTION

An object file's symbol table is machine dependent as defined in each implementation's copy of **<a.out.h>**. The nlist structure can be used to extract this information. The data associated with each symbol is machine specific, thus only the name and position of the **n_name** field is standardized by HP-UX. The rest of the structure includes at least the value and type of the symbol. The names and meanings of all fields not standardized will change no more than necessary.

HARDWARE DEPENDENCIES

Series 200:
The nlist structure looks as follows:

```
struct nlist {  
    char      *n_name;  
    long      n_value;  
    unsigned char n_type;  
    unsigned char n_length;  
    short      n_unit;  
    short      n_sdindex;  
};
```

SEE ALSO

nlist (3), a.out (5)

HP-UX STANDARD NOTE

There will be a separate nlist(5) manual page (with appropriate changes) for each system.

NAME

passwd, pwd.h - password file

HP-UX COMPATIBILITY

Level:

Multi-user - HP-UX/STANDARD

Origin:

System III

DESCRIPTION

Passwd contains for each user the following information:

- login name
- encrypted password
- numerical user ID
- numerical group ID
- GCOS job number, box number, optional GCOS user ID
- initial working directory
- program (or shell script) to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The GCOS field is used only when communicating with that system, and in other installations can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself 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. Password aging is effected 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.) The first character of the age, `M` say, 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, `m` say, 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.) `M` and `m` have numerical values in the range 0-63. If `m` = `M` = 0 (derived from the string `.` or `..`) 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 $m > M$ (signified, e.g., by the string `./`) 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;
    int     pw_uid;
    int     pw_gid;
    char    *pw_age;
    char    *pw_comment;
    char    *pw_gecos;
    char    *pw_dir;
    char    *pw_shell;
};
```

FILES

`/etc/passwd`

SEE ALSO

`login(1)`, `passwd(1)`, `a64l(3C)`, `crypt(3C)`, `getpwent(3C)`, `group(5)`.

PROFILE(5)

PROFILE(5)

NAME

profile - setting up an environment at login time

HP-UX COMPATIBILITY

Level:

HP-UX/STANDARD

Origin:

System III

DESCRIPTION

If your login directory contains a file named **.profile**, that file will be executed (via the shell's **exec .profile**) before your session begins; **.profiles** are handy for setting exported environment variables and terminal modes. If the file **/etc/profile** exists, it will be executed for every user before the **.profile**. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
# Tell me when new mail comes in
MAIL=/usr/mail/myname
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
echo "terminal: \c"
read TERM
case $TERM in
  300)      stty cr2 nl0 tabs; tabs;;
  300s)     stty cr2 nl0 tabs; tabs;;
  450)      stty cr2 nl0 tabs; tabs;;
  hp)       stty cr0 nl0 tabs; tabs;;
  7451735)  stty cr1 nl1 -tabs; TERM=745;;
  43)       stty cr1 nl0 -tabs;;
  40141tek) stty cr0 nl0 -tabs ff1; TERM=4014; echo "\33;";
  *)        echo "$TERM unknown";;
esac
```

FILES

\$HOME/.profile
/etc/profile

SEE ALSO

env(1), login(1), mail(1), sh(1), stty(1), su(1),
environ(7), term(7).

NAME

sysconf - system configuration table file format

HP-UX COMPATIBILITY

Level:

HP-UX/NUCLEUS

Origin:

HP

DESCRIPTION

The file /etc/configure contains a default configuration table which the super-user can edit and modify for his particular system. These are the default values:

```
root_device_driver = no default value
root_device = no default value
vm_device_driver = no default value
vm_device = no default value
ios_cache_buf_size = 1024
ios_cache_buf_num = 25
ios_read_ahead_level = 4
ios_running_secured = 1
vm_swap_time = 20
vm_pagesize = 2048
vm_max_pool_size = 163840
uke_num_scroll_pages = 2
uke_local_time_zone = 420
ios_keep_stats = 0
ios_cluster_size = 4
vm_working_set_ratio = 0.012
```

Running the program /etc/reconfig then checks the consistency and integrity of /etc/configure and installs the data in the boot area. Its syntax is:

```
/etc/reconfig [-c file] [-i boot_device]
```

File specifies the name of the configuration table file. Default is /etc/configure.

Boot_device specifies that the new configuration data be installed in the boot area of that device. Otherwise, it is not installed.

If there is an error, /etc/configure returns -1. If a modified value is outside the acceptable range, the default value will be used instead. Errors are written to STDERR.

FILES

/etc/configure

UTMP(5)

UTMP(5)

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)**, **acctcon(1)** (see **acctcon(1M)**), 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 */
};
```

FILES

/etc/utmp
/usr/adm/wtmp
/usr/include/utmp.h

SEE ALSO

acctcon(1M), login(1), who(1), write(1).

INTRO(7)

INTRO(7)

NAME

intro - introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as
macro packages, character set tables, etc.

|
|

ENVIRON(7)

ENVIRON(7)

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 `xxxnzzz` 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 fildes flags */
#define F_SETFD 2 /* set fildes flags */
#define F_GETFL 3 /* Get file flags */
#define F_SETFL 4 /* Set file flags */
```

SEE ALSO`fcntl(2)`, `open(2)`.

NAME

regex - 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 <regex.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;

int step(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/regex.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 unpleasantly complex. Programs that include this file must have the following five macros declared before the `"#include <regex.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 be placed. If the compiled expression cannot fit in (endbuf-exbuf) 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 register variable to point the beginning of the regular expression so that this register 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, exbuf)
```

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 exbuf is the compiled regular expression which was obtained by a call of the function `compile`.

The function `step` returns 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 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 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 `loc` is equal to the point in the string 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 `gcmp` 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 `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 kludgy.

The routine ecmp is equivalent to the Standard I/O routine strcmp and should be replaced by that routine.

The actual code is probably easier to understand than this manual page.

NAME

stat - data returned by stat 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
 */
```

```
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_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_IRREAD    00400 /* read perm., owner */
#define S_IWRITE    00200 /* write perm., owner */
#define S_IXEXEC    00100 /* ex/search perm., owner */
```

STAT(7)

STAT(7)

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., `proff(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`:

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(Reg.) 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

TERM(7)**TERM(7)**

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 `-Tterm` 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)`.

BUGS

This is a small candle trying to illuminate a large, dark problem. Programs that ought to adhere to this nomenclature do so somewhat fitfully.

TYPES(7)

TYPES(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 UNIX system code; some data of these types are accessible to user code:

```
#define NREGS_S 13 /* no. of regs saved */
```

```
typedef struct { int r[11]; }      *physadr;
typedef long                      daddr_t;
typedef char                      *caddr_t;
typedef unsigned short            ushort;
typedef ushort                    ino_t;
typedef short                     cnt_t;
typedef long                      time_t;
typedef int                       label_t[NREGS_S];
typedef short                     dev_t;
typedef long                      off_t;
typedef long                      paddr_t;
```

The form `daddr_t` is used for disk addresses except in an i-node on disk, see `fs(5)`. Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. 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.

SEE ALSO

`fs(5)`.

INTRO(8)

INTRO(8)

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

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, or 6 (may vary locally) which selects a speed table in getty, or !, which tells getty to update **/etc/utmp** and exit; 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.

First, getty types the **login:** message. The **login:** message depends on the speed table being used, and may include the characters that put the GE TermiNet 300 terminal into full-duplex, take the DASI terminals out of the plot mode, or put a TELETYPE(Reg.) Model 37 into full-duplex. 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 (which must be all lower case) is terminated by a new-line or carriage-return character. 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.

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 a change in 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`. Lines in this file have the format:

```
state:id:flags:command
```

All lines in which the state field matches `init`'s current state are recognized. If a process is active under the same two character `id` as a recognized line, it may be terminated (signal 15), killed (signal 9), or both by including the flags `t` and `k` in the order desired. The signal is sent to all processes in the process group associated with the `id`. The `command` field is saved for later execution.

After reading `/etc/inittab` and signaling running processes as required, but before invoking any processes under the new state, `/etc/rc` is invoked with three arguments. This command file performs housekeeping such as removing temporary files, mounting file systems, and starting daemons. The three arguments are the current state, the number of times this state has been entered previously, and the prior state. `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, `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 and writing on file descriptors 0, 1, and 2, resetting all signals to system default, setting up a new process group (`setpgid(2)`), and `execing`:

```
/bin/sh -c exec command
```

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.

FILES

`/etc/inittab`
`/etc/rc`
`/bin/sh`
`/dev/console`



SEE ALSO

`login(1)`, `sh(1)`, `exec(2)`, `setpgid(2)`, `inittab(5)`, `getty(8)`.

MAKEKEY(8)

MAKEKEY(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 COMPATABILITY

Level:

HP-UX/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 possible 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.

You must change directories to the place you want the device files to go (typically, /dev) before running **mkdev**.

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.

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 is expected to modify this script before using it the first time, an error is given if it has not been modified.

SEE ALSO

mknod(1m), mkdir(1), chmod(1).

RC(8)

RC(8)

NAME

rc - system initialization shell script

SYNOPSIS

/etc/rc

HP-UX COMPATIBILITY

Level:

HP-UX/RUN ONLY

Origin:

System III

DESCRIPTION

The /etc/rc file is executed by init(8) whenever the init state is changed.

SEE ALSO

init(8).

Glossary

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.

archive - a file which is made up of other files (such as a group of library files to be used by the linker (ld)).

block special file - a special file associated with discs and other volumes (such as CS '80 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 a mass storage medium which contains the operating system.

character special file - a special file associated with devices which transfer data character-by-character. Examples are printers, card readers, terminals, and discs accessed in raw mode.

child - a process spawned via the fork 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 command language).

command interpreter - a program which reads lines typed at the keyboard, and interprets them as requests to execute other programs. The command interpreter for HP-UX is called the shell.

control character - a member of a character set which may produce some action in a device other than a printed or displayed character. In ASCII, control characters are those in the code range 0 thru 31, and 127. Control characters are generated by simultaneously pressing a displayable character key and [CTRL], [CONTROL] or [CNTRL].

crash - the unexpected shutdown of a system, usually requiring the system to be re-booted.

data encryption - a method for scrambling information in order to protect sensitive or proprietary data.

default search path - the sequence of directory prefixes that sh, time, and other HP-UX commands apply in searching for a file known by an incomplete path name. It is defined by PATH in environ. login sets PATH = :/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.

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 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. The effective user ID is equal to the process's real user ID, unless the process or one of its ancestors evolved from 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 type of terminal you are using) that exist for any given process.

end-of-file character - the character returned when trying to read past the logical end of a file.

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 0 (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).

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 currently the only means by which new processes are created in HP-UX.

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 ASCII DEL ("rubout") character. This signal generally causes whatever program you are running to terminate. The key which sends this signal 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.

library - a file containing a set of subroutines and variables which may be accessed by a user program.

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) in a.out format. The ld command performs this function.

loader - takes a file (in a.out format) from the linker (ld), loads it into memory, then executes it. The exec system call performs this function.

login directory - the directory which becomes the default directory when you first log in. It is specified in /etc/passwd, on the line beginning with your user name.

login procedure - the procedure required before a user can begin using a working HP-UX system. He must enter his user name and password (if one exists).

metacharacter - a character which has special meaning to the HP-UX shell. The set of metacharacters includes:
* ? [] < > ; | ' " &

mode - the 12 bits which determine access and execute permission for a file, 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. It is usually a separate disc or disc pack.

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 shell user interface normally recognizes the carriage-return sent by a terminal as the new-line indicator and translates it into the new-line character.

ordinary file - a type of HP-UX file, created by the user, and containing a program or text.

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.

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.

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).

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 system 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 define it to be any string.

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 user ID and real group ID that are set to the real user ID and real group ID, respectively, 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 and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

regular expression - specifies 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.

relative path name - path name not beginning with a slash. It indicates that the file's location is given relative to the 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 "/" refers to the root directory.

shell - the shell is both a command language and a programming language that provides the user-interface to the HP-UX operating system.

shell script - a sequence of shell commands and shell programming language constructs, usually stored in a text 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.

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 special file normally reside in the directory /dev.

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 CRT, 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 terminal keyboard, 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 terminal CRT, 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.

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.

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 "intrinsic".

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 process 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 "/".