UniPress Software, Inc.
Suite 312
2025 Lincoln Highway
Edison, NJ 08817
201-985-8000
Telex: 709418

Brief Unix Manual

UniPress Software
Your Leading Source for UNIX Software

2025 Lincoln Hwy.
Edison, NJ 08817
201-985-8000
Telex: 709418

Installation Instructions and Brief Introduction to Commands and Keys

The UNIX V Operating System on the Apple Lisa (Model 2) for Software Shipped on Microdiskettes

Thank you for purchasing the UniPress UniPlus+ UNIX V operating system for the Apple Lisa Model 2. We hope you find it useful and enjoyable.

You have received manuals, a set of Unix 3 1/2-inch diskettes, and optionally other floppies containing application software. Power up the Lisa and your hard disk. Always wait for all the equipment to be ready before proceeding with booting!

Here is some general information about our implementation:

When the Lisa is powered-up, or restarted by hitting the round reset button on the back, the computer goes through a self-test procedure. If there is a built-in hard disk or a ProFile in the Parallel Device outlet, the Lisa will automatically attempt to boot from there. Thus, if you have the Apple Lisa Office system loaded onto the built-in drive, and Unix on a Sunol, for example, the Lisa will always try to boot the Office system! To avoid this, you MUST hit the space bar repeatedly as soon as the "CPU" self-test commences. Then by pointing the mouse you can boot from floppy disk.

On any Unix system it is a VERY good idea to perform periodic "fsck"s. fsck determines if the filesystem is in good order, and will fix many problems.

When powering-down we suggest that you issue the "sync" command twice, and hit the lit power button on the front right of the panel. (Hitting this button automatically performs the "sync" commands, but we advise you to do them manually anyway.) The computer will power-off.

If you want to re-boot without powering-down, issue two "sync" commands, and then hit the round "reset" button which is near the card cage on the back of the machine. The Lisa will automatically begin its startup procedure and self-test.

The Lisa "apple" key with the picture of the Apple is the Unix Control Key, the CLEAR key on the keypad is the Unix DEL (delete) key, and the OPTION key is the UNIX ESC (escape) key.

If you are a Unix devotee, you might like to put paste-labels on the keys to indicate the ESC, etc. keys.

By the way, the Lisa terminal is mostly like a VT100. A "vtl" (for VTLisa) entry has been placed into the termcap.

Throughout this document there are references to "Hit return" and <CR>. This means the Return key should be pressed.
The included devices in this Unix are:

```
/dev/s0a                    800 block full Sony 3-1/2" diskett
/dev/s0b                    599 block Sony diskette starting
                             at block 201
```

NOTE: To format diskettes enter:
diskformat /dev/rs0a
To make a filesystem on a diskette:
mkfslb /dev/s0a 800

/dev/p0a	Full 10-meg internal disk (less some swap area.)
/dev/p0b	Swap area on ProFile or internal 10-meg.
/dev/p0c	5-meg ProFile (less swap)
/dev/c4a	Full Corvus/Sunol 20-meg.
/dev/c4b	Swap area for above.
/dev/c4c	Balance of Corvus/Sunol if larger than 25-meg.

NOTE: The second digit indicates where the device is located. "0" (as in p0c) indicates the built-in port -- either the in-board Widget drive, or a disk plugged into the "Parallel" slot on the lower back of some Lisas.

Ports 1 and 2 are the bottom and top of the left (as seen from the back) parallel card; ports 4 and 5 are the middle card; ports 7 and 8 are the slots for a righthand card.

IF YOU EVER NEED TO RESTORE FILES FROM THE MASTER FLOPPY DISKETTES, it is easy to do. The "C" diskette contains a few programs. These are all in filesystem format, which means that you must mount them, and then use the cp program to get your file. Example:

```
mount /dev/s0a /t
ls -l /t/bin
cp /t/bin/eject /tmp
umount /dev/s0a
```

All other diskettes are in tar format.

```
tar xvf /dev/rs0a /bin/eject
```

This document is organized into sections. They are:

Section	Purpose
Serialize your Lisa	You must do this as the first step when you receive your software.
INSTALLING INTERNAL DISK OR PROFILE SYSTEMS	For users who have either the internal 10-meg or ProFile hard disks.
INSTALLING CORVUS/SUNOL DISK SYSTEMS	For users who have Corvus or Sunol disks.
WHAT TO DO AFTER INSTALLING	Explains how to use the system on an on-going basis.

Serialize your Lisa

Prior to use of the Unix software, you must "serialize" the system.

A. Position the boot/serialization ("A") diskette so that the write-protect hole is on the left side near you, and the notched corner of the diskette is on the far righthand side. Push the floppy into the slot GENTLY.

B. Press the reset button on the back of the Lisa. This is the black round button on the bottom near the card cage. The Lisa will begin its self-test when you hit the reset button.

C. Indicate that you want to boot from the floppy.

D. You will be asked whether you want to Serialize. Indicate that you want to serialize by typing "y". You will see several "Serialization Numbers". Call UniPress Software at 201-985-8000 or 800-222-0550 (outside NJ). Read the numbers to UniPress. We will give you a "Authorization Number", which is a 7 or 8-character sequence. (NOTE: any letters in the sequence must be keyed as lower case!) Save the Authorization Number, since you may need it later in the installation process, or possibly in the future. We suggest you write it here:

Authorization Number: _____

Input the Authorization number. The serialization diskette will be transformed into a boot diskette, which only functions on this specific Lisa. For the balance of this document, we refer to this floppy as the boot diskette.

Go on to the "INSTALLING ..." sections.

INSTALLING INTERNAL DISK OR PROFILE SYSTEMS

A. If you are using a ProFile, cable it into the "Parallel Device" slot on the lower rear of the chassis if your machine has one. (This slot is horizontal.) Otherwise, you must insert a Lisa Parallel card into the middle of the three slots in the card cage. Then plug the ProFile into the lower of the two outlets. Power up the ProFile and wait for it to be ready (solid red light) before using it!

B. Boot the Lisa from floppy. (The "A" diskette from the previous step.)

C. The Lisa will go into its Unix initialization process and will put a new display on the screen, which says, "Standalone boot". A colon (:) will appear on the next line.

enter "copy" after the prompt. This copies the boot program from the diskette onto your hard disk.

The Lisa will prompt "Enter device to copy from", and you should hit return.

The diskette will be ejected soon, and you will be prompted: "Enter device to copy to:"

You respond with "w(x,0)" where "x" has the value as seen below:

ProFile in 0 port (the "Parallel Port")	0	
Profile in 4 port	4	
Builtin 10-meg disk	0	ex: w(0,0)

You will see activity on the hard disk (lights blinking) and then the Lisa will then reply: "Enter file to boot from"

D. Re-insert the boot ("A") diskette and enter "sunix". After a while you will be asked to insert the "root filesystem", which is the "C" diskette. Do so, and then hit return.

You will be asked for the location of the "swap" area. Indicate "p" . Then provide the same number as you did for the previous step. (eg. 0, or 4) when asked for the number.

Unix will now start operation! You will see the "#" Unix prompt. However, it is a very limited system at this point.

E. Type "./install", and answer the questions. This will take approximately 7 minutes. Then enter "eject" to eject the diskette.

Re-boot as requested, by issuing "sync" twice and hitting the round reset button on the back of the machine.

F. Boot up from floppy "A", as before. Then input the indicated entry from this table when you see the colon (:) prompt:

ProFile in 0 port (the "Parallel port")	"unix"
Internal 10-meg	"unix"
ProFile in 4 port	"eject", then insert "B" diskette, and enter "unix.w4"

G. Once again, when you get the "#" Unix prompt, enter "./install".

Enter "eject" to eject the diskette.

There is one last step to perform, which is the loading of the Unix files from diskette. Please issue the "./install_it" command to load in the rest of the Unix System.

H. Proceed directly to WHAT TO DO AFTER INSTALLATION.

INSTALLING CORVUS/SUNOL DISKS

A. Connect the disk to the Lisa with the special flat cable:

Put the flat ribbon end of the cable into the slot labelled "PROCESSOR" on the Corvus. On the Sunol there is only one receptacle. On the Corvus THE RED SIDE OF THE CABLE MUST FACE TOWARDS THE POWER CORD. FAILURE TO CONNECT THE CABLE PROPERLY MAY RESULT IN DAMAGE. On the Sunol, the cable can only go in one way, with the red side down.

WHEN USING THE CORVUS DISK YOU MUST ALWAYS BE SURE THAT THE FOUR RED SWITCHES WHICH ARE BENEATH THE LIGHTS ON THE FRONT PANEL ALL POINT TO THE LEFT.

WHEN USING SUNOL DISKS IT IS ABSOLUTELY ESSENTIAL THAT THEY BE "PRE-FORMATTED" FOR UNIX. STANDARD FACTORY FORMATTING IS NOT SATISFACTORY. UNIPRESS PERFORMS THE UNIX FORMATTING ON DISKS WHICH IT PROVIDES, BUT OTHER VENDORS MAY HAVE NOT DONE SO. IF YOU HAVE A SUNOL DISK FROM ANOTHER SOURCE YOU MUST CONTACT UNIPRESS FOR THE FORMATTING INSTRUCTIONS, WHICH YOUR VENDOR CAN APPLY.

Put the RS-232 side of the cable into the Lisa. It goes into a Lisa parallel card in the housing on the back of the Lisa. The card must be in the middle of the three slots. If there is no card in the middle slot, remove the back panel of the Lisa and insert one into the middle slot. Then replace the back panel. Connect the cable to the lower of the two RS-232 outlets on the board.

To recap: On the Lisa side the cable goes into the lower outlet of a card which must be located in the middle of the three slots in the housing on the back; on the hard disk the flat cable goes into the PROCESSOR outlet of the Corvus with the red side towards the power cord, while on the Sunol it goes into the only outlet, with the red side pointed down.

B. Insert the boot "B" diskette into the drive.

C. Re-boot the Lisa from floppy, by using the mouse to indicate floppy. Indicate that you want to serialize again, by entering "y". Then supply the same numbers given to you earlier by UniPress. Enter "eject" when you get the ":" prompt.

D. Re-insert the "A" diskette. At "Standalone boot:", enter "sunix" and hit return. As soon as the diskette is ejected, insert the "C" diskette. Then hit return to start, as indicated.

E. Answer "c" for Corvus/Sunol and "4" for port 4 when asked.

F. After you see the "#" Unix prompt, issue "./install" to copy files onto the harddisk. This takes approximately 7 minutes. Then enter "eject" and remove the diskette.

G. Reboot by hitting the reset button on the back of the Lisa. Then re-insert the "B" diskette. Type "unix.c4" at the ":" prompt. The system will boot itself. In the future you will boot using the "B" diskette, and will have no further use for the "A" diskette.

H. Type "./install" and when that step has finished, type "./install_it" to complete the installation.

I. Proceed directly to WHAT TO DO AFTER INSTALLATION

WHAT TO DO AFTER INSTALLATION

A. When you reboot in the future, the steps to take depend upon whether you have a ProFile, built-in 10-meg, or a Corvus/Sunol.

ProFile in 0 port (Parallel Device) or built-in 10-meg

Simply use the mouse to indicate your hard disk, and Unix will boot. No diskette is needed. Enter "w(0,2501)unix" when you get the ":" prompt.

ProFile in 4 port

Use the mouse to indicate slot 7. This is unfortunate, but Apple considers the lower middle outlet as 7, while Unix considers it as 4. No diskette is needed. Enter "w(4,2501)unix" when you get the ":" prompt.

Corvus/Sunol in 4 port

Use the mouse to indicate the floppy, and insert the "B" diskette. Enter "unix.c4" at the ":" prompt.

B. Lisa Unix is single user at this point. To come up multi-user, type the command "init 2" and hit return. You will get a confirming message from Unix that you have changed run state.

C. Login as root, or rootcsh if you want the Cshell as your command prompter.

D. The port labelled "Serial B" on the back is /dev/ttyl, and if you plug a terminal there with a reverser (null modem) cable, you can use Unix multi-user at 9600 baud. Use stty command to change speed, after you are logged in on that port.

E. To change the speed of a port other than the one you are using, edit the /etc/inittab file. To handle 300/1200 baud communication, as when dialing in, change the "d" at the end of the relevant line to an "H". The "d" means 9600 baud. See the inittab pages in Sections 4 and 5 of the full Unix manual for further information. To make the change effective, either reboot or do the following:

> Enter "kill -1 1", and enter "init 2" when you get a prompt.

F. The port labelled "Serial A" on the back is /dev/tty0, and is typically used to communicate with other computers via the cu or uucp commands.

As shipped, this port is not enabled for login (since we have it set up for cu). To enable it for login, edit the file /etc/inittab, and duplicate the ttyl entry for tty0 to enable login. You must then either reboot, or follow the steps indicated above in "D". The port will come up at 9600 baud. Use the stty command to change the baud rate.

G. If you have additional hard disks you can connect them to the Lisa for more storage. These disks must be ProFile, Corvus or Sunol drives. (Of course, if you use them for Unix you will lose any other data you have put there.) To add more disks to the Lisa Unix:

a Hook up the additional disk to the Lisa by plugging it into a parallel card.

b The outlets on vertical parallel cards are:

lower left	/dev/c1a
upper left	/dev/c2a
lower middle	/dev/c4a
upper middle	/dev/c5a
lower right	/dev/c7a
upper right	/dev/c8a

(NOTE: Left and right are as seen from the BACK of the Lisa.)

c You can access these devices by:

```
mount /dev/c7a /t
cd /t
.
.
.
```

H. IF YOU HAVE A DISK LARGER THAN 20-MEG

You must "inform" Unix that you have more storage area. The way you do this is by issuing the following commands:

a. mknod /dev/c4c b 2 66

b. mkfslb /dev/c4c <SIZE> -- where <SIZE> is the value shown here:

disk size	value
25-meg	10000
40-meg	40000
65-meg	80000
92-meg	130000

c. mkdir /u

When you want to use the larger area, you do so by treating it as a "mountable filesystem". This is done by issuing the following commands:

a. mount /dev/c4c /u
b. cd /u
.
.
.

NOTE: You can put the mount command as shown above into the /etc/rc file, and the rest of the disk will be available automatically to you when you run multi-user.

I. REMEMBER, PLEASE RUN THE FSCK COMMAND OFTEN TO HELP KEEP YOUR FILESYSTEM IN GOOD ORDER.

This is a brief summary of some things you must understand in order to use the Unix UniPlus+ operating system on the Apple Lisa. This list is not complete or comprehensive; these commands are described in more detail on the manual pages which follow, and in the full Unix manuals.

Things to know first:

A number of Unix commands use the CONTROL and DEL keys. On the Lisa keyboard the CONTROL key the one with the picture of an apple, and the DEL key is labelled CLEAR. Throughout the Unix documentation you will see references to these keys. The Control key must be pressed simultaneously with the other key involved. Thus for Control-D, press Control and "d" together.

The 'DEL' key will stop a command which is in progress.

The Control-D combination acts as end-of-file when you are redirecting the "standard input" from the keyboard. (Control-D is also used to log you out if you are using the regular "Bourne" shell, so be careful).

The Control-S combination will stop the output to the screen, and Control-Q combination will re-start that output.

A Few Commands

cat - is used to type a file or files to the screen. This is equivalent to the 'type' command found in most systems.

cc - is used to run the c compiler; e.g. "cc file.c" will produce an executable program called 'a.out'. Just type "a.out" and the program will execute.

cd - is used to change your current working directory. Try the 'cd /bin' command to 'go to' the directory where most of these commands live. Then do an 'ls' command to see what is listed there.

cmp - is used to compare two files, bit for bit.

cp - is used to copy one file to another. You can also copy a group of files to another directory, since this command, and most of the others, accept "wild cards." For example, "ab*" means all files staring with "ab", and "a?c" means all files staring with "a", ending with "c", and having any second character, such as "abc", "aac", etc.

cu - is used to call up another Unix system and (possibly) transfer files between them. This is VERY useful for the Lisa, and very easy to do. To sign on to another Unix through Serial B of the Lisa (which has the Unix address /dev/tty1), type:

```
% cu -s 1200 -l /dev/tty1 dir
```

then login to the other machine. To disconnect type: ~. To transfer files, use the ~%take and ~%put options described in the cu manual page. If you are using an autodialer, after you see the "Connected" message, enter the dialing command. (On the Hayes modem at this point you enter ATDT9761212, for example.)

There is a peculiarity in cu: Even though you specify the baud rate you want (with the -s XXXX flag), you must also change the terminal entry in /usr/lib/uucp/L-devices. (The change to make is self-explanatory).

> NOTE: cu only functions on a port which has no login running. To see if you have a login running, do a ps -ale command and look under the "TTY" column for an entry under the port number involved. (This will be 0 or 1.) If there is an entry there with a "getty" in the right column, login is enabled. If you have previously enabled login on the port you now intend for cu, you must disable login by modifying the /etc/inittab file. Read the relevant section in the Unix manual and/or look at /etc/inittab. You then must either shutdown and re-boot, or follow the steps indicated earler in the WHAT TO DO AFTER INSTALLATION section of this guide.

date - is used to print or set the date. The format is MMDDHHMMYY, as in 0204144584, for February 4, 1984; 2:45PM.

df - is used to display the amount of free space on your filesystem. For instance, to print the free space on the Lisa hard disk, type: df /dev/p0a (which is the Unix name for the Profile). df with no arguments shows the free space on all the "mounted" disks.

du - is used to list the space occupied by each file in the current directory and its subdirectories.

diff - is used to compare two files and get a list of the differences.

diskformat - is used to format a floppy.

```
diskformat /dev/rs0a
```

dump - is used to dump a file system to a floppy diskette. The filesystem which is dumped can be restored with the restor command.

echo - is used to echo the command line arguments to your terminal.

ed - is the Unix line editor. It is much more powerful than most microcomputer line editors available. See pages 3-8 of the ed manual pages for more information.

fsck - is used to check the consistency and correctness of the Unix filesystem. Just type /etc/fsck (fsck 'lives' in the /etc directory), and the 'root' filesystem will be checked for internal consistency; if necessary fsck will fix any problems. (No problems will exist unless the system has previously crashed). You can check other filesystems by giving their names, as in fsck /dev/c4a.

grep - is used to search a file or files for a text pattern.

login - is used to sign on to the system. If you sign on as root, you will have 'superuser' privileges... you can access almost any file, etc. Read the login entry in Section 1 and the passwd entry in Section 5 of the full manual for details on adding additional user ids, etc.

ls - is used to list the contents of a directory. This is known as dir on many systems. The command ls -l will list the contents of a directory in long form. ls7 gives a colmnar list of the files.

mkdir - is used to make a new directory. The command 'mkdir mark' will make a new directory named 'mark'; to get to this directory, type 'cd mark'

mkfslb - will make a new Unix filesystem on a hard disk or floppy disk. This is the Unix version of the usual format commands. NEVER issue mkfslb to your hard disk or you will kill it! mkfslb should be used to make a Unix filesystem on Lisa for new floppy disks after they are formatted.

```
mkfslb /dev/s0a 800
```

more - is used to look at a file or files a screenful at a time. Hit the space bar to get the next screen of data.

mount - is used to mount a Unix filesystem. To mount a diskette, type 'mount /dev/s0a /t', Then /t is the name of that floppy.

To use a floppy:
. If it is not yet formatted
  diskformat /dev/rs0a
  mkfslb /dev/s0a 800

. In any case
  mount /dev/s0a /t
  cd /t

As an example,
  cat /usr/me/myfile >/t

mv - is used to move (or rename) a file or files. 'mv file* dir' moves a series of files to the directory dir.

od - is used to dump files to the CRT in a variety of formats.

pr - is used to print files with headings, including the file name, date, etc. Files can be printed multi-column.

rm - is used to remove a file or series of files.

sh - is the Unix shell or command line interpreter. The shell is the program which listens to you when you log on. Shells can run under shells, and shells have their own programming language built-in. By the way, your Unix system also includes the more-powerful cshell (csh).

sort - is used to sort and merge data.

stty - is used to set or display the terminal options. All of the Lisa ports including the bit-map display can be run at 300, 1200, or 9600 baud, and with many other options.

sync - is used to ensure that all disk writes have been completed before the system is shutdown. Issue sync twice to be sure.

umount - is used to unmount a mounted floppy filesystem.

REMEMBER, READ THE MANUAL FOR FULL EXPLANATIONS

NAME

ar — archive and library maintainer

SYNOPSIS

ar [uvbail] [mrxtdpq] [posname] archivename filename(s) ...

DESCRIPTION

The archive command *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 loader. However, *ar* can be used for any similar archiving purpose. Archives often consist of unlinked program modules.

Key is *one* character from the set **mrxtdpq**, optionally concatenated with one or more of **uvnbail**. *Archivename* is the archive file. The *filename(s)* are constituent files in or destined for 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 **p**, 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.

EXAMPLE

```
ar rv libar.a text.o
```

places file "text.o" in archive "libar.a".

```
ar bm file1 archivename file2
```

changes the location of a file inside an archive. "File2" is the file to be moved. "File2" is moved to a new position before "file1".

FILES

/tmp temporaries

SEE ALSO

ld(1), ar(4).

BUGS

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

Sufficient disk space must be present to make an entire copy of the archive or the *ar* command will fail.

NAME

as — assembler

SYNOPSIS

as [**−o** objfile] [**−l**] [name ...]

DESCRIPTION

As assembles the named files, or the standard input if no file name is specified.

All undefined symbols in the assembly are treated as global.

The relocatable output of the assembly is left on the file *objfile*; if that is omitted, *a.out* is used.

The **−l** option produces an assembly listing on file *objfile.lst*. If the **−l** option is specified and no **−o** parameter is specified, the assembly listing is placed on *a.lst*.

EXAMPLE

```
as −o file.o filea fileb filec
```

would assemble the three named files and put the output of the assembly into "file.o".

FILES

/tmp/as*	default temporary file
a.out	default resultant object file
a.lst	default assembly listing file

SEE ALSO

adb(1), ld(1), nm(1), a.out(4)
AS Assembler Reference Guide, James L. Gula and Thomas J. Teixeira. Revised by UniSoft Systems.

NAME

cat – concatenate and print files

SYNOPSIS

cat [**-u**] [**-s**] file ...

DESCRIPTION

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

If no input file is given, or if the argument – is encountered, *cat* reads from the standard input file. Output is buffered unless the **-u** option is specified. The **-s** option makes *cat* silent about non-existent files. No input file may be the same as the output file unless it is a special file.

EXAMPLE

```
cat file
```

prints the file, and:

```
cat file1 file2 > file3
```

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

WARNING

Command formats such as

```
cat file1 file2 >file1
```

will cause the original data in *file1* to be lost, therefore, take care when using shell special characters.

SEE ALSO

cp(1), pr(1).

NAME

cc − C compiler

SYNOPSIS

cc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler.

Cc 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 if a single C program is compiled and loaded.

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.

−n Passed on to *ld* to make the text of the resulting program shared.

−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 startup 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).

−O(KPS) Invoke an object-code improver (optimizer). If **K** is specified, certain UNIX kernel optimizer functions are not performed. If **P** is specified, stack probe instructions are removed. (Note: **P** should only be used for the operating system source.) If **S** is specified, stack frame optimization is performed and the debugger, *adb*(1), might indicate too few subroutine parameters on stack trace back.

−R (addr) Passed on to *ld*, making the resulting object module *origin*ed at *addr(hex).*

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

−E Run only *cpp*(1) on the named C programs, and send the result to standard output.

−P Run only the macro preprocessor on the named C programs, and send the result to the corresponding files suffixed. '.i'

−C Prevent the macro preprocessor from eliding (leaving out) comments.

−o *output* Name the final executable output file *output.* If this option is used the file "a.out" will be left undisturbed.

−D*name*=*def*

−D*name* Define the *name* to the preprocessor, as if by **#define**. If no definition is given, the name is defined as "1".

−U*name* Remove any initial definition of *name*.

−I*dir* **#include** files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in −I options, then in the directory **/usr/include**.

−v print the name of each subprocess as it is executing.

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 via *ld*(1) (in the order given) to produce an executable program with name *a.out*.

EXAMPLE

```
cc -o output prog1.c prog2.c prog3.c
```

would compile code in the three named C programs and put the compiled code into the file "output".

FILES

file.c	input file
file.o	object file
a.out	linked output
/tmp/ctm?	temporary
/lib/cpp	preprocessor
/lib/c	combined compiler pass1 and pass2
/lib/c0	compiler pass1
/lib/c1	compiler pass2
/lib/c2	optional optimizer invoked with "-O"
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	runtime startoff for profiling
/lib/libc.a	standard library, see section 3
/usr/include	standard directory for '#include' files
/lib/libm.a	math library

SEE ALSO

adb(1), ld(1), lint(1), prof(1), monitor(3C)
The C Programming Language, Prentice-Hall, 1978, by B. W. Kernighan and D. M. Ritchie
Programming in C—a tutorial, by B. W. Kernighan
C Reference Manual, by D. M. Ritchie

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. Confusing syntax may cause the C compiler to indicate an error on the line following the actual error.

NAME

cd — change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **$HOME** is used as the new working directory. If *directory* specifies a complete path starting with /, ., .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable. *Cd* must have execute (search) permission in *directory*.

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 internal to the shell.

EXAMPLE

```
cd /unisoft/usr/games
```

would relocate you to the directory **/unisoft/usr/games** if this directory is executable (searchable) by you.

SEE ALSO

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

CHMOD (1)

NAME

chmod – change mode

SYNOPSIS

chmod mode files

DESCRIPTION

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

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

A symbolic *mode* has the form:

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

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

Op can be + to add *permission* to the file's mode, – to take away *permission*, or = to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save text, or 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.

EXAMPLE

chmod 755 filename

changes the mode of "filename" to: read, write, execute (400+200+100) by owner; read, execute (40+10) for group; read, execute (4+1) for others. An *ls -l* of filename shows [-rwxr-xr-x filename] that the requested mode is in effect.

chmod = filename

will take away all permissions from *filename*, including yours.

chmod o-w file

denies write permission to others.

chmod +x file

makes a file executable.

SEE ALSO

ls(1), chmod(2).

NAME

chown, chgrp — change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

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.

EXAMPLE

```
chown unisoft filea fileb filec
```

would make "unisoft" the owner of the three files.

FILES

/etc/passwd
/etc/group

SEE ALSO

chown(2), group(4), passwd(4).

NAME

clear — clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type and then in **/etc/termcap** to figure out how to clear the screen.

EXAMPLE

```
clear
```

clears the screen.

FILES

/etc/termcap terminal capability data base

CP(1)

NAME

cp, ln, mv — copy, link or move files

SYNOPSIS

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

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same (take care when using *sh*(1) metacharacters). If *target* is a directory, then one or more files are copied (linked, moved) to that directory.

If *mv* determines that the mode of *target* forbids writing, it will print the mode (see *chmod*(2)) and read the standard input for one line (if the standard input is a terminal); if the line begins with y, the move takes place; if not, *mv* exits.

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.

EXAMPLE

cp alpha beta gamma /unisoft/roxanne

places copies of the three files in the directory **/unisoft/roxanne**.

SEE ALSO

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

BUGS

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

Ln will not link across file systems.

NAME

cmp − compare two files

SYNOPSIS

cmp [−l] [−s] file1 file2

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.

EXAMPLE

```
cmp alpha beta
```

will report if the files are different and at what point they differ, such as:

```
alpha beta differ: char 33, line 2
```

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

cu − call another UNIX System

SYNOPSIS

cu [−sspeed] [−lline] [−**h**] [−**t**] [−**d**] [−**m**] [−**o**|−**e**] telno | **dir**

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, 600, 1200, 4800, 9600); 300 is the default value. Most of our modems are either 300 or 1200 baud. For dial out lines, *cu* will choose a modem speed (300 or 1200) as the slowest available which will handle the specified transmission speed. Directly connected lines may be set to speeds higher than 1200 baud.

The −**l** value may be used to specify a device name for the communications line device to be used. This can be used to override searching for the first available line having the right speed. The speed of a line is taken from the file **/usr/lib/uucp/L-devices**, overriding any speed specified by the −s option. The −**h** option emulates local echo, supporting calls to other computer systems which expect terminals to be in half-duplex mode. The −**t** option is used when dialing an ASCII terminal which has been set to auto-answer. Appropriate mapping of carriage-returns to carriage-return-line-feed pairs is set. The −**d** option cause diagnostic traces to be printed. The −**m** option specifies a direct line which has modem control. The −**e** (−**o**) option designates that even (odd) parity is to be generated for data sent to the remote. The −**d** option causes diagnostic traces to be printed. *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* may be used for directly connected lines, and implies a null ACU. Using **dir** insures that a line has been specified by the −**l** option.

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.

EXAMPLE

cu -s 1200 777-8888

attempts to connect to the telephone line numbered "777-8888" at 1200 baud rate.

FILES

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

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uucp(1C).

DIAGNOSTICS

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

BUGS

Cu buffers input internally.
There is an artificial slowing of transmission by *cu* during the **~%put** operation so that loss of data is unlikely.

NAME

date − print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

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.

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.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year − 01 to 12
d	day of month − 01 to 31
y	last 2 digits of year − 00 to 99
D	date as mm/dd/yy
H	hour − 00 to 23
M	minute − 00 to 59
S	second − 00 to 59
T	time as HH:MM:SS
j	day of year − 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'
```

generates as output:

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

DIAGNOSTICS

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

FILES

/dev/kmem

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
seek=*n*	seek *n* records from beginning of output file before copying; *dd* creates the specified output file (see *creat*(2)), which insures the length of the file will be zero; seeking *n* records from the beginning of the output file will fill the skipped area with zeros (nulls).
count=*n*	copy only *n* input records
conv=**ascii**	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
..., ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii*, *ebcdic*, or *ibm* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter two cases ASCII characters are read into the conversion buffer, converted to EBCDIC (or the IBM version of EBCDIC), and blanks added to make up an output record of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

EXAMPLE

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file "x".

NAME

diff − differential file comparator

SYNOPSIS

diff [−**efbh**] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is −, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

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

The −**e** option produces a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The −**f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with −**e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

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

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

EXAMPLE

```
diff -e file1 file2
```

where "file1" and "file2" are two versions of the manual text for the *cp* command, produces:

```
35,41d
27c
In the second form, one or more

18,25c
existed; the mode of the source file
is used otherwise.
```

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

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

BUGS

The ASCII/ EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The *ibm* conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

```
15c
The mode and owner of

10c
file ... directory

7c
file1 file2

1,3c
.TH CP 1
.SH NAME
```

Following this *ed* script would transform "file1" into file2", line for line and character for character.

FILES

/tmp/d?????
/usr/lib/diffh for −h

SEE ALSO

cmp(1), comm(1), ed(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]

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.

EXAMPLE

```
du dir1 dir2
```

produces a count of the number of blocks in each of the directories. In order to see how many blocks are in each file, the **–a** option must be used.

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.

NAME

dump − dump selected parts of an object file

SYNOPSIS

dump [−a] [−f] [−o] [−h] [−s] [−r] [−l] [−t] [−z name] files

DESCRIPTION

The *dump* command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

−a	Dump the archive header of each member of each archive file argument.
−f	Dump each file header.
−o	Dump each optional header.
−h	Dump section headers.
−s	Dump section contents.
−r	Dump relocation information.
−l	Dump line number information.
−t	Dump symbol table entries.
−z name	Dump line number entries for the named function.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

−d number	Dump the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by +d.
+d number	Dump sections in the range either beginning with first section or beginning with section specified by −d.
−n name	Dump information pertaining only to the named entity. This *modifier* applies to −h, −s, −r, −l, and −t.
−t index	Dump only the indexed symbol table entry. The −t used in conjunction with +t, specifies a range of symbol table entries.
+t index	Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the −t option.
−v	Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of 0X02). This *modifier* can be used with all the above options except −s and −o options of *dump*.
−z name,number	Dump line number entry or range of line numbers starting at *number* for the named function.
+z number	Dump line numbers starting at either function *name* or *number* specified by −z, up to *number* specified by +z.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **-z** option may be replaced by a blank.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

EXAMPLE

```
dump 0bf 2310 /dev/rfdc0 /dev/rmsc0a
```

would perform a level '0' dump to the floppy disk device "rfdc0", which has 2310 blocks. The filesystem to be dumped is **/dev/rmsc0a**. Note that all the parameters in the key are grouped first in the command line, followed by the dump device (if other than tape), size, etc. The last argument should be the pathname of the file system being dumped.

SEE ALSO

a.out(4), ar(4).

NAME

echo — echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

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

- **\b** backspace
- **\c** print line without new-line
- **\f** form-feed
- **\n** new-line
- **\r** carriage return
- **\t** tab
- **** backslash
- *****n* the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero.

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

EXAMPLE

```
echo curmudgeon
```

simply responds

```
curmudgeon
```

on the standard output.

SEE ALSO

sh(1).

NAME

ed, red − text editor

SYNOPSIS

ed [−] [−x] [file]

red [−] [−x] [file]

DESCRIPTION

Ed is the standard 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.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via !*shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty −tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10 and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character 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., **[]a-f]** matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly* *m* occurrences; \{*m*,\} matches *at least* *m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between* *m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the

same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A currency symbol ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^ *entire RE*$ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. 'x addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or −, the addition or subtraction is taken with respect to the current line; e.g, −5 is understood to mean .−5.
9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the

editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so −− refers to the current line less 2.

10. For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n** or **p**, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file* The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell

command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file* The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file* If *file* is given, the *f*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE/command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted; the . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE/*

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given,

this command does nothing.

(.)k*x* The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address *'x* then addresses this line; . is unchanged.

(.,.)l The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a* The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

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

($)r *file* The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/g
The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator g appears after the command. If the global indicator does not appear, only the first

occurrence of the matched string is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a* This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

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

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

(1,$)V/*RE/*

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

(1,$)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

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

(.)= The line number of the addressed line is typed; . is unchanged by this command.

!*shell command*
The remainder of the line after the ! is sent to the UNIX System 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?
```

EXAMPLE

```
ed text
```

would invoke the editor with the file named "text". For further examples, see "*A Tutorial Introduction to the UNIX Text Editor*" and "*Advanced Editing on UNIX*"

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 *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy

ed's buffer via the *e* or *q* commands: it prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The − command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1), stty(1), fspec(4), regexp(5).
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 *v* command.
The *!* command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).
The sequence \n in a RE does not match a new-line character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.
Characters are masked to 7 bits on input.

NAME

grep, egrep, fgrep − search a file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

−v All lines but those matching are printed.

−x (Exact) only lines matched in their entirety are printed (*fgrep* only).

−c Only a count of matching lines is printed.

−l Only the names of files with matching lines are listed (once), separated by new-lines.

−n Each line is preceded by its relative line number in the file.

−b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

−s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).

−e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a − (does not work with *grep*).

−f *file*
The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters $, *, [, ^, |, (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

Fgrep searches for lines that contain one of the *strings* 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.

EXAMPLE

```
grep -v -c 'regular' grep.1
```

reports a count of the number of lines that do **not** contain the word *regular* in the file "grep.1".

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.
Lines are limited to 256 characters; longer lines are truncated.
Egrep does not recognize ranges, such as **[a−z]**, in character classes.

NAME

login – sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file".

If *login* is invoked as a command, it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful *login*.

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*, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and the file **.profile** in the working directory is executed, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is – followed by the last component of the interpreter's pathname (i.e., **–sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your *login* name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

L*n*=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people,

logging into restricted shell environments, from spawning secondary shells which aren't restricted. Both *login* and *getty* understand simple single character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

EXAMPLE

At the beginning of each terminal session, the following sort of message is displayed on the screen:

```
UniSoft 68000 UNIX
:login:
```

to which a user name is the appropriate response.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	systemwide personal profile (*sh*(1))
/etc/cshrc	systemwide personal csh startup (*csh*(1))
.profile	personal profile (*sh*(1))
.login	personal csh startup used at login time (*csh*(1))
.cshrc	personal csh startup (*csh*(1))
.logout	personal csh logout used at logout time (*csh*(1))

SEE ALSO

mail(1), newgrp(1), sh(1), su(1), passwd(4), profile(4), environ(5).

DIAGNOSTICS

Login incorrect
if the user name or the password cannot be matched.

No shell, cannot open password file, or *no directory*
consult a UNIX system programming counselor.

No utmp entry. You must exec "login" from the lowest level "sh".
if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

ls − list contents of directories

SYNOPSIS

ls [**−logtasdrucifp**] names

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 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 and minor device numbers, rather than a size.

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

−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.

−p Put a slash after each filename if that file is a directory. Especially useful for CRT terminals when combined with the *pr*(1) command as follows: **ls −p | pr −5 −t −w80**.

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 (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

EXAMPLE

```
ls -l /etc
```

will list all entries in **/etc** in long format.

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).

NAME

mkdir — make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask* (1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically. These and other directories beginning with . are not visible in listings unless you use the **-a** option to *ls*.

Mkdir requires write permission in the parent directory.

EXAMPLE

```
mkdir letters
```

creates a directory **letters** as a subdirectory of the directory you are in at the time you employ the command.

SEE ALSO

rm(1), sh(1), umask(1).

DIAGNOSTICS

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

NAME

more − file perusal filter for crt viewing

SYNOPSIS

more [**−dfl***n*] [+*linenumber* | +/*pattern*] [name ...]

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a CRT terminal. It normally pauses after each screenful, printing "--More--" at the bottom of the screen.

If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. If a space is preceded by an integer, that number of lines is printed. If the user hits **d** or control-D, 11 more lines are displayed (a "scroll").

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

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

The following options are available:

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

−d causes *more* to prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful.

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

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

+ *linenumber*
option causes *more* to start up at *linenumber*

+/*pattern*
causes *more* to start up two lines before the line containing the regular expression *pattern*.

Once inside *more*, other sequences may be typed when *more* pauses. The sequences and their effects are as follows (*i* is an optional integer argument, defaulting to 1) :

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

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

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

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

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

q or Q
Exit from more.

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

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

!command
invoke a shell with *command.*

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

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

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

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

EXAMPLE

nroff −ms +2 doc.n | more

would show the *nroff* output on the terminal screen.

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

AUTHOR

Eric Shienbrood

NAME

od – octal dump

SYNOPSIS

od [**–bcdosx**] [file] [[+]offset[.][**b**]]

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 words in unsigned decimal.

–o Interpret words in octal.

–s Interpret 16-bit words in signed decimal.

–x Interpret 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.

EXAMPLE

```
od -d filea +2
```

produces an octal dump of "filea" divided up into 32-bit words expressed in decimal equivalents with the dump starting point offset by 2 octal bytes.

SEE ALSO

dump(1).

−p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

−f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on failure to open files.

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

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

EXAMPLE

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

prints "file1" and "file2" as a double-spaced, three-column listing headed by "file list".

```
pr −e9 −t < file1 > file2
```

writes "file1" on "file2", expanding tabs to columns 10, 19, 28, 37, ...

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1).

NAME

rm, rmdir − remove files or directories

SYNOPSIS

rm [**−fri**] file ...

rmdir dir ...

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.

EXAMPLE

```
rm -r dirname
```

will remove the entire contents of the named directory and all subdirectories, and finally the directory itself, with no questions asked.

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

sh, rsh − shell, the standard/restricted command programming language

SYNOPSIS

sh [−**ceiknrstuvx**] [args]
rsh [−**ceiknrstuvx**] [args]

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* 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 | (or, for historical compatibility, 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*

SH (1)

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). The user may not change PATH if executing under *rsh*.
CDPATH	The search path for the *cd* command.
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. If the first character following the opening "[" is a "!", then any character not enclosed is matched.

Quoting.

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

; & ( ) | ^ < > new-line space tab

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

Prompting.

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

Input/Output.

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

< word

Use file *word* as standard input (file descriptor 0).

> word

Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.

>> word

Use file *word* as standard output. If the file exists, then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[−] word

The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\new-line** is ignored, and \ must be used to quote the characters \, $, `, and the first character of *word*. If − is appended to <<, then all leading tabs are stripped from *word* and from the document.

< &digit

The standard input is duplicated from file descriptor *digit* (see *dup*(2)). Similarly for the standard output using >.

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

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

... 2>&1

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

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

Redirection of output is not allowed in the restricted shell.

Environment.

The *environment* (see *environ*(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the 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; such commands will not be executed by the restricted shell. 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*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, then the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

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

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

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

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

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

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

SH(1)

end-of-file is encountered.

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

set [−−**ekntuvx** [*arg* ...]]
−**e** Exit immediately if a command exits with a non-zero exit status.
−**k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
−**n** Read commands but do not execute them.
−**t** Exit after reading and executing one command.
−**u** Treat unset variables as an error when substituting.
−**v** Print shell input lines as they are read.
−**x** Print commands and their arguments as they are executed.
−− Do not change any of the flags; useful in setting **$1** to −.
Using + rather than − causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$**−. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given, then the values of all names are printed.

shift [*n*]
The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

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

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.

ulimit [−**fp**] [*n*]
imposes a size limit of *n*
−**f** imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.
−**p** changes the pipe size to *n* (UNIX/RT only).
If no option is given, −**f** is assumed.

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

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given, then all currently active child processes are waited for and

the return code is 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.

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

Rsh Only.

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **$PATH**,
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

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

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

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

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).

EXAMPLE

```
sh -x script1
```

will execute each command in "script1", echoing the command just before executing it.

FILES

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

SEE ALSO

cd(1), env(1), login(1), newgrp(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), profile(4), environ(5).

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.

NAME

sort − sort and/or merge files

SYNOPSIS

sort [−**cmubdfinrtx**] [+pos1 [−pos2]] ... [−o output] [names]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name − means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d "Dictionary" order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b**.

r Reverse the sense of comparisons.

t*x* "Tab character" separating fields is *x*.

The notation +*pos1* −*pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m*.*n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first non-blank in the field; **b** is attached independently to *pos2*. A missing .*n* means .0; a missing −*pos2* means the end of the line. Under the −t*x* option, fields are strings separated by *x*; otherwise fields are non-empty non-blank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

EXAMPLE

```
sort −u +0f +0 list
```

prints in alphabetical order all the unique spellings in a list of words (capitalized words differ from uncapitalized).

sort −t: +2n /etc/passwd

prints the password file (*passwd*(4)) sorted by user ID (the third colon-separated field).

sort −um +0 −1 dates

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).

FILES

/usr/tmp/stm???

SEE ALSO

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

DIAGNOSTICS

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

BUGS

Very long lines are silently truncated.

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**–a**] [**–g**] [options]

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 *termio*(7) for asynchronous lines in the *UniPlus*$^{+}$ *Administrator's Manual*. 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 *termio*(7)).

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 possible. (All speeds are not supported by all hardware interfaces.)

hupcl (**–hupcl**) hang up (do not hang up) a DATA-PHONE® data set 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 *termio*(7)).

inpck (**–inpck**) enable (disable) input parity checking.

istrip (**–istrip**) strip (do not strip) input characters to seven bits.

inlcr (**–inlcr**) map (do not map) NL to CR on input.

igncr (**–igncr**) ignore (do not ignore) CR on input.

icrnl (**–icrnl**) map (do not map) CR to NL on input.

iuclc (**–iuclc**) map (do not map) upper-case alphabetics to lower case on input.

ixon (**–ixon**) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (**–ixany**) allow any character (only DC1) to restart output.

ixoff (**–ixoff**) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (−opost) post-process output (do not post-process output; ignore all other output modes).

olcuc (−olcuc) map (do not map) lower-case alphabetics to upper case on output.

onlcr (−onlcr) map (do not map) NL to CR-NL on output.

ocrnl (−ocrnl) map (do not map) CR to NL on output.

onocr (−onocr) do not (do) output CRs at column zero.

onlret (−onlret) on the terminal NL performs (does not perform) the CR function.

ofill (−ofill) use fill characters (use timing) for delays.

ofdel (−ofdel) fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 select style of delay for carriage returns (see *termio*(7)).

nl0 nl1 select style of delay for line-feeds (see *termio*(7)).

tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *termio*(7)).

bs0 bs1 select style of delay for backspaces (see *termio*(7)).

ff0 ff1 select style of delay for form-feeds (see *termio*(7)).

vt0 vt1 select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (−isig) enable (disable) the checking of characters against the special control characters INTR and QUIT.

icanon (−icanon) enable (disable) canonical input (ERASE and KILL processing).

xcase (−xcase) canonical (unprocessed) upper/lower-case presentation.

echo (−echo) echo back (do not echo back) every character typed.

echoe (−echoe) echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (−echok) echo (do not echo) NL after KILL character.

lfkc (−lfkc) the same as **echok (−echok)**; obsolete.

echonl (−echonl) echo (do not echo) NL.

noflsh (−noflsh) disable (enable) flush after INTR or QUIT.

stwrap (−stwrap) disable (enable) truncation of lines longer than 79 characters on a synchronous line.

stflush (−stflush) enable (disable) flush on a synchronous line after every *write*(2).

stappl (−stappl) use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **ctab**, **min**, or **time** (**ctab** is used with **−stappl**), (**min** and **time** are used with **−icanon**; see *termio*(7)). 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* ($0 < i < 127$).

Combination Modes

evenp or **parity** — enable **parenb** and **cs7**.

oddp — enable **parenb**, **cs7**, and **parodd**.

−parity, **−evenp**, or **−oddp** — disable **parenb**, and set **cs8**.

raw (**−raw** or **cooked**) — enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).

nl (**−nl**) — unset (set) **icrnl**, **onlcr**. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**−lcase**) — set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (**−LCASE**) — same as **lcase** (**−lcase**).

tabs (**−tabs** or **tab3**) — preserve (expand to spaces) tabs when printing.

ek — reset ERASE and KILL characters back to normal # and

sane — resets all modes to some reasonable values.

term — set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

EXAMPLE

```
stty
```

produces a list of the terminal settings currently in use. To change a setting, type in the command and the desired option. More than one option can be requested on one command line.

```
stty 300
```

sets your terminal to operate at 300 baud (hardware permitting).

```
stty < /dev/tty1
```

reports the terminal characteristics of **/dev/tty1**.

SEE ALSO

tabs(1), ioctl(2).
termio(7) in the *UniPlus+ Administrator's Manual.*

NAME

sync — update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

EXAMPLE

```
sync
```

should be typed to flush all internal disk buffers, before bringing down the system.

SEE ALSO

sync(2).