PRO/VENIX™

UNIX™
for

Professional™

User Reference Manual

digital
software

PRO/VENIX™
for the Professional

User Reference Manual

Developed by:

VenturCom, Inc.
215 First Street
Cambridge, MA 02142

Digital Equipment Corporation
Maynard, MA 01754

First Printing

The PRO/VENIX† Documentation Set

The PRO/VENIX documentation set consists of the following manuals:

PRO/VENIX Installation and System Manager's Guide

The set up and maintenance of PRO/VENIX are described in the installation sections. Other articles explain the UNIX-to-UNIX‡ communications systems. The "System Maintenance Reference Manual" contains reference pages for devices and system maintenance procedures (sections (7) and (8)).

PRO/VENIX User Guide

The *User Guide* contains tutorials for newcomers to PRO/VENIX, covering basic use of the system, the editor **vi** and use of the command language interpreters.

PRO/VENIX Document Processing Guide

The line and screen editors and **nroff**-related text formatting utilities are described in the Document Processing Guide. Topics include: line editor **ed**, and stream editor **sed**; the text formatter **nroff**; the **nroff**-preprocessors **tbl** and **neqn**.

PRO/VENIX Programming Guide

The chapters in the *Programming Guide* explicate the different programming languages for VENIX.

† VENIX is a trademark of VenturCom, Inc.

‡ UNIX is a trademark of Bell Laboratories.

PRO/VENIX Support Tools Guide

This guide includes tools for programming, such as the compiler-writing languages Yacc and Lex, the M4 Macro processor, the program development utility Make, and the desk calculator programs DC and BC.

PRO/VENIX User Reference Manual

This is a complete and concise reference for the PRO/VENIX system. This volume contains write-ups on all PRO/VENIX commands.

PRO/VENIX Progammer Reference Manual

The reference pages in this volume include system calls, library functions, file formats, miscellaneous functions and games.

Introduction to the User Reference Manual

This volume describes features of the VENIX† system, a licensed UNIX‡ time-sharing operating system. The *User Reference Manual* does not provide an overview or tutorial information on the VENIX operating system, its facilities, or implementation. Various documents on these topics are contained in other VENIX manuals. For a general tutorial see "VENIX for Beginners" in the *User Guide*. Those with previous UNIX experience will find that VENIX is quite similar to other versions of UNIX.

The *User Reference Manual* attempts to be timely, complete and concise. A lot of information is packed into these pages in a concentrated form. Examples have been included to assist the reader to understand some of the more terse descriptions. We hope that the user will be able to flip open the manual, refer to the appropriate section and find the needed information.

The *User Reference Manual* contains section one, VENIX commands. Sections two through six are contained in the *Programmer Reference Manual*. The user should be familiar with the basic layout:

User Reference Manual

1. Commands

Programmer Reference Manual

2. System calls
3. Subroutines
4. File Formats and Conventions
5. Miscellaneous Facilities
6. Games

When referring to a particular item in the manual, it is common practice to append the section number in parentheses: for instance, **ls**(1) refers to the **ls** command in section 1 of the *User Reference Manual*.

†VENIX is a trademark of VenturCom, Inc.
‡UNIX is a trademark of Bell Laboratories

Introduction to the User Reference Manual

At the beginning of this volume is a table of contents, organized alphabetically.

Following is a short overview of the sections in both the *User Reference Manual* and the *Programmer Reference Manual*:

All the VENIX **commands** are contained in the *User Reference Manual*. **Commands** are programs intended to be invoked directly by the user. Commands generally reside in directory */bin* (for *bin* ary programs). Some programs also reside in */usr/bin*, to save space in */bin*. These directories are searched automatically by the command interpreter.

System calls are entries into the VENIX supervisor. Every system call has one or more C language interfaces described in section 2.

An assortment of **subroutines** is available; they are described in section 3. The primary libraries in which they are kept are described in **intro**(3).

Section 4 **file formats and conventions** documents the structure of particular kids of files, for example, the form of the output of the loader and assembler.

Miscellaneous facilities in sections 5 include user environment, graphics interface and terminal capability database.

If you find that the *User Reference Manual* is rather prosaic reading, see section 6 for **games**.

In the *User Reference Manual* the name of the entry is in the upper corner of the page, together with the section number, and sometimes a letter characteristic of a subcategory, e.g. graphics is 1G. Entries are alphabetized and the page numbers of each entry start at 1.

All entries are based on a common format; though not all the subsections will always appear:

> The *name* subsection lists the exact names of the commands and subroutines covered under the entry and gives a very short description of their purpose.
>
> The *synopsis* summarizes the use of the program being described. A few conventions are used:
>
> > **Boldface** is reserved for literals, typed just as they appear, e.g. **nroff**(1). Italics are used for generic arguments which are to be replaced by real parameters.

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

Ellipses '. . .' are used to show that the previous argument-prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus sig '−' is often taken to mean some sort of option-specifying argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with '−'.

The *description* subsection discusses in detail the subject at hand.

The *files* subsection gives the names of files which are built into the program.

A *see also* subsection gives pointers to related information, such as other commands or tutorials.

A *diagnostics* subsection discusses the diagnostic indications which may be produced. Messages which are intended to be self-explanatory are not listed.

The *bugs* subsection gives known bugs and sometime deficiencies. Occasionally the suggested fix is also described.

Table of Contents

Table of Contents

(COMMANDS continued)

Table of Contents

(COMMANDS continued)

Table of Contents

(COMMANDS continued)

NAME

intro — introduction to commands

DESCRIPTION

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

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

Most of the documents mentioned in the "SEE ALSO" section can be found in other VENIX Manuals.

DIAGNOSTICS

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

When programs are executed from the shell, the exit code is stored in a shell variable. It may be seen, for example, by typing:

```
echo $?
```

in the Bourne shell (**sh**), or by typing:

```
echo $status
```

in the C shell (**csh**). See shell programming documentation for details on how shell scripts may make use of this exit code.

ac — login accounting

SYNOPSIS

ac [**−w** wtmp] [**−p**] [**−d**] [people] ...

DESCRIPTION

ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced. **−w** is used to specify an alternate *wtmp* file. **−p** prints individual totals; without this option, only totals are printed. **−d** causes a printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. Times are given in hours.

If no *wtmp* file is given, **/usr/adm/wtmp** is used. The accounting file **/usr/adm/wtmp** is maintained by **init**(8) and **login**(1). Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. *wtmp* is not provided automatically with VENIX because not all users want accounting and if the file is left undisturbed, it grows endlessly. To start accounting, it should be created with length 0. This can be done by the super-user, by typing:

```
cp /dev/null /usr/adm/wtmp
```

Remember that the file should be cleaned out periodically.

FILES

/usr/adm/wtmp

SEE ALSO

login(1), utmp(4), ‘‘VENIX Maintenance’’

NAME

access — determine accessibility of files and directories

SYNOPSIS

access name ...

DESCRIPTION

access displays the user's permissions for the named files or directories. Permissions are given as "read," "write," and/or "execute." This information could otherwise be determined by listing the file with the **ls**(1) command and checking the appropriate mode.

If the named argument is a directory, "read" permission can be interpreted as permission to list the contents of the directory; "write" permission, to create or remove files within the directory; and "execute" permission, to enter the directory and access a file within it. Directory permissions are preceded with the word "(dir)."

If a file or directory can not be found, **access** determines if the directories leading to it are unreadable by the user, or if it simply does not exist. In the latter case, **access** prints out the pathname as far as it could follow it.

If a file/directory exists but is inaccessible, its owner or the system administrator must change the relevant modes before the user will be able to access it.

EXAMPLES

Determine accessibility of VENIX Berkeley enhancements, **curses** and **termcap**:

```
access /usr/lib/libcurses.a /etc/termcap
```

SEE ALSO

ls(1), chmod(1)

NAME

adb — debugger

SYNOPSIS

adb [− **w**] [objfil [corfil]]

DESCRIPTION

adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of VENIX programs. The following synopsis is supplemented by "A Tutorial Introduction to ADB" in the *Programming Guide*. This tutorial is strongly recommended for first-time users.

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

Requests to **adb** are read from the standard input and responses are sent to the standard output. If the −**w** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using **adb**. **adb** ignores QUIT ('^Z'); INTERRUPT ('^C') causes return to the next **adb** command.

In general requests to **adb** are of the form

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

If *address* is present then *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see ADDRESSES.

EXPRESSIONS

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

" The last *address* typed.

integer

VENIX/11 and PRO/VENIX:
An octal number if *integer* begins with a 0; a hexadecimal number if preceded by **#**; otherwise a decimal number.

RAINBOW/VENIX: A number. The prefixes 0o and 0O (''zero oh'') force interpretation in octal radix; the prefixes 0t and 0T force interpretation in decimal radix; the prefixes 0x and 0X force interpretation in hexadecimal radix. Thus 0o20 = 0t16 = 0x10 = sixteen. If no prefix appears, then the default radix is used; see the **$d** command. The default radix is initially hexadecimal. The hexadecimal digits are 0123456789abcdefABCDEF with the obvious values. Note that a hexadecimal number whose most significant digit is an alphabetic character must have a 0x (or 0X) prefix.

integer.fraction
A 32 bit floating point number.

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

< *name*
The value of *name*, which is either a variable name or a register name. **adb** maintains a number of variables (see VARIABLES) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil*. The register names are **r0**, **r5**, **sp**, **pc**, **ps**. (VENIX/11 and PRO/VENIX) and **IP**, **AX**, **BX**, **CX**, **DX**, **SI**, **DI**, **SP**, **BP**. (RAINBOW/VENIX). The **$r** command prints all the registers.

symbol A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. **** may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ or ~ will be prepended to *symbol* if needed.

_symbol
In C, the 'true name' of an external symbol begins with _. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

routine.name
: The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*.

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

Monadic operators

**exp* The contents of the location addressed by *exp* in *corfil*.

@exp The contents of the location addressed by *exp* in *objfil*.

−exp Integer negation.

~exp Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1+e2 Integer addition.

e1−e2 Integer subtraction.

*e1*e2* Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

e1#e2 *e1* rounded up to the next multiple of *e2*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '*'; see ADDRESSES for further details.)

?*f* Locations starting at *address* in *objfil* are printed according to the format *f*.

/*f* Locations starting at *address* in *corfil* are printed according to the format *f*.

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

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

o	2	Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O	4	Print 4 bytes in octal.
q	2	Print in signed octal.
Q	4	Print in long signed octal.
d	2	Print in decimal.
D	4	Print in long decimal.
x	2	Print 2 bytes in hexadecimal.
X	4	Print 4 bytes in hexadecimal.
u	2	Print as an unsigned decimal number.
U	4	Print in long unsigned decimal.
f	4	Print the 32 bit value as a floating point number.
F	8	Print in double floating point.
b	1	Print the addressed byte in octal.
c	1	Print the addressed character.
C	1	Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
s	*n*	Print the addressed characters until a zero character is reached.
S	*n*	Print a string using the @ escape convention. *n* is the length of the string including its zero terminator.
Y	4	Print 4 bytes in date format (see **ctime**(3)).
i	n	Print in the form of PDP-11 instructions (VENIX/11 and PRO/VENIX) or 8086 instructions (RAINBOW/VENIX). *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a	0	Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

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

p 2	Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t 0	When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.
r 0	Print a space.
n 0	Print a newline.
"..."0	Print the enclosed string.
^	*Dot* is decremented by the current increment. Nothing is printed.
+	*Dot* is incremented by 1. Nothing is printed.
−	*Dot* is decremented by 1. Nothing is printed.

newline If the previous command temporarily incremented *dot*, make the increment permanent. Repeat the previous command with a *count* of 1.

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

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

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

Write the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. On VENIX/11 and PRO/VENIX, odd addresses are not allowed when writing to the subprocess address space.

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

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

>*name*

Dot is assigned to the variable or register named.

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

$*modifier*

Miscellaneous commands. The available *modifiers* are:

<*f*	Read commands from the file *f* and return.
>*f*	Send output to the file *f*, which is created if it does not exist.
r	Print the general registers and the instruction addressed by **pc** (VENIX/11 and PRO/VENIX) or **IP** (RAINBOW/VENIX). *Dot* is set to **pc** (VENIX/11 and PRO/VENIX) or **IP** (RAINBOW/VENIX).
f	Print the floating registers in single or double length. On VENIX/11 and PRO/VENIX, if the floating point status of **ps** is set to double (0200 bit) then double length is used anyway.
b	Print all breakpoints and their associated counts and commands.
c	C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **r5** (VENIX/11 and PRO/VENIX) or **BP** (RAINBOW/VENIX)). If *count* is given then only the first *count* frames are printed. On VENIX/11 and PRO/VENIX, if **C** is used then the names and (16 bit) values of all automatic and static variables are printed for each active function.
e	The names and values of external variables are printed.
w	Set the page width for output to *address* (default 80).
s	Set the limit for symbol matches to *address* (default 255).
o	All integers input are regarded as octal.
d	VENIX/11 and PRO/VENIX : Reset integer input as described in EXPRESSIONS. RAINBOW/VENIX : Set the default radix to *address* and report the value. Note that *address* is interpreted in the (old) current radix. Thus "10$d" never changes the default radix. To make the default radix decimal, use "0t10$d". (RAINBOW/VENIX only.)
q	Exit from *adb*.
v	Print all non zero variables in octal.
m	Print the address map.
a	ALGOL 68 stack backtrace. (VENIX/11 and PRO/VENIX). If *address* is given then it is taken to be

the address of the current frame (instead of **r4**). If *count* is given then only the first *count* frames are printed.

:*modifier*

Manage a subprocess. Available modifiers are:

b*c*	Set breakpoint at *address*. The breakpoint is executed *count* − 1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero then the breakpoint causes a stop.
d	Delete breakpoint at *address*.
r	Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.
c*s*	The subprocess is continued with signal *s*; see **signal**(2). If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.
s*s*	As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
k	The current subprocess, if any, is terminated.

VARIABLES

adb provides a number of variables. Named variables are set initially by **adb** but are not used subsequently. Numbered variables are reserved for communication as follows.

0	The last value printed.
1	The last offset part of an instruction source.
2	The previous value of variable 1.

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

b	The base address of the data segment.
d	The data segment size.
e	The entry point.
m	The 'magic' number (0405, 0407, 0410, or 0411).
s	The stack segment size.
t	The text segment size.

ADDRESSES

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

$$b1 \le \text{address} < e1 => \text{file address} = \text{address} + f1 - b1,$$

otherwise,

$$b2 \le \text{address} < e2 => \text{file address} = \text{address} + f2 - b2,$$

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a **?** or **/** is followed by an ***** then only the second triple is used.

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

So that **adb** may be used on large files all appropriate values are kept as signed 32 bit integers.

FILES

/dev/mem
a.out
core

SEE ALSO

ptrace(2), a.out(4), core(4)

DIAGNOSTICS

adb returns 'adb' when there is no current command or format. **adb** comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

A breakpoint set at the entry point is not effective on initial entry to the program.
When single stepping, system calls do not count as an executed instruction.
Local variables whose names are the same as an external variable may foul up the accessing of the external.

NAME

ar — archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

ar maintains groups of files combined into a single archive file. Its main use is to create and update library files (object modules) as used by the loader. It can be used, though, for any similar purpose. Since it assumes no particular format for its files, it can be used to archive files of object modules, source files, or anything else.

key is one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcl**. *afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end. If no names are given, an attempt is made to replace all files in the archive with any copies found in the current directory.

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. When used with **v**, it gives a verbose directory listing analogous to that of **ls**(1). User/group ID numbers are given instead of names.

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 to change files in the archive, it lists each file, preceding it with a letter indicating its activity, such as '**a**' (appended), '**r**' (replaced), or '**d**' (deleted). 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.

EXAMPLES

Create a new archive called *modlib* with modules *mod1.o* and *mod2.o*:

```
ar r modlib mod1.o mod2.o
```

List verbosely all modules in the archive:

```
ar tv modlib
```

Insert new modules *mod1.o* and *mod3.o*:

```
ar r modlib mod1.o mod3.o
```

(*mod1.o* was replaced since it previously existed in *modlib*; *mod3.o* was added to the end as a new module.)

Add new module *mod2.o* before *mod3.o* in the archive:

```
ar rb mod3.o modlib mod2.o
```

FILES

/tmp/v*	temporaries

SEE ALSO

ld(1), ar(4), lorder(1), ranlib(1)

BUGS

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

NAME

as — assembler

SYNOPSIS

VENIX/11 and PRO/VENIX
as [−] [**−m**] [**−o** objfile] file ...

RAINBOW/VENIX
as file[.s]

DESCRIPTION

as is the VENIX assembler. The primary purpose of this assembler is to be used by the C or Fortran compilers; it is not normally executed directly by the user. Assembler listing of C or Fortran may be generated by using the **−S** flag with **cc** or **f77**.

− All undefined symbols are treated as global. (VENIX/11 and PRO/VENIX only).

−m "Code-mapping" (memory overlaying) is to be used. This alters the assembly output format slightly. This flag may in fact be used at all times under VENIX, whether or not code-mapping is actually invoked at load-time. However, it will cause incompatibilities with other PDP-11 UNIX assemblers. (VENIX/11 and PRO/VENIX only).

−o The object code output will be placed in the following file, instead of **a.out**. (VENIX/11 and PRO/VENIX only). In RAINBOW/VENIX, the output file name is taken from the source file (only one allowed), with '**.o**' replacing the optional '**.s**' in the name.

NOTES FOR RAINBOW/VENIX

Compiler additional assembler documentation is not provided for RAINBOW/VENIX. The following brief notes are provided for users who insist in writing assembly code; further details must be gleaned from assembler listings produced by the C compiler.

The basic RAINBOW/VENIX assembler syntax is:

```
label:   op[b] [dst/src][,src] | comments
```

where *dst/src* take the form:

reg:	ax,bx,cx,dx,bp,si,di,sp and also h,l for x
constant:	* for byte or # for word with normal C constants
memory:	label
index:	* number (reg) or # number (reg)
indirect:	(reg) except no (bp)

Several relevant pseudo ops:

```
.text                               |text or code space
.data                               |data space
.comm          name, size           |BSS space
.globl         name
.word          constant [, ...]
.byte          constant [, ...]
```

The input and output instructions are:

IN	reads a byte from port address specified in DX register into AL register.
IN port address	reads a byte from specified port address into AL register.
OUT	writes a byte from the AL register to the port address specified into the DX register.
OUT port address	writes a byte from the specified port address into the DX register.

FILES

file.s	assembler source file
file.o	object (RAINBOW/VENIX)
a.out	object (VENIX/11 and PRO/VENIX)

SEE ALSO

cc(1), ld(1), nm(1), adb(1), a.out(4)
"VENIX Assembler Reference Manual" in the *Programming Guide*

EXAMPLES (RAINBOW/VENIX only)

An example of a RAINBOW/VENIX assembler subroutine callable from C.

```
|
| add (a,b)              /* return a + b * /
|
.text
.globl _add
_add:   push bp          |save old frame pointer
        mov bp,sp        |get new pointer
        mov ax,*4(bp)    |get first argument
        add ax,*6(bp)    |add second to it
        pop bp           |restore old frame pointer
        ret              |return value in ax
```

DIAGNOSTICS (VENIX/11 and PRO/VENIX only)

When an input file cannot be read, its name followed by a question mark is typed and assembly ceases. When syntactic or semantic errors occur, a single-character diagnostic is typed out together with the line number and the file name in which it occurred. Errors in pass 1 cause cancellation of pass 2. The possible errors are:

)	Parentheses error
]	Parentheses error
<	String not terminated properly
*	Indirection used illegally
.	Illegal assignment to '.'
a	Error in address
b	Branch instruction is odd or too remote
e	Error in expression
f	Error in local ('f' or 'b') type symbol
g	Garbage (unknown) character
i	End of file inside an if
m	Multiply defined symbol as label
o	Word quantity assembled at odd address
p	'.' different in pass 1 and 2
r	Relocation error
u	Undefined symbol
x	Syntax error

BUGS

The RAINBOW/VENIX assembler does not conform to the Intel assembler specification.

NAME

at — execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

at squirrels away a copy of the named *file* (standard input default) to be executed at a specified later time.

The *time* is 1 to 4 digits, with an optional following 'A', 'P', 'N' or 'M' for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional *day* is either (1) a month name followed by a day number, or (2) a day of the week; if the word 'week' follows invocation is moved seven days further off. Names of months and days may be recognizably truncated.

at stores its copy of the user's commands in a file which is in fact a shell script. This script is created in such a way that, when run, the user commands are executed under the same conditions that the user was in when **at** was invoked. Towards this goal, a **cd**(1) command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the user and group ID of the invoker.

at programs are executed by periodic execution of the command **/usr/lib/atrun** which itself is scheduled and executed by **/etc/cron**; the **cron** command must be set up to execute for **at** to work. (For more details about **cron**, see ''VENIX Maintenance'' in the *Installation and System Manager's Guide.*) The granularity of **at** depends upon how often **atrun** is executed by **cron** as given by the table in **/usr/lib/crontab**; at distribution, execution is set to once every half hour.

Standard output or error output produced by the user's commands is lost unless redirected (to an error log file for example).

EXAMPLES

Execute the programs *blob* and *glob* at 8am on January 24:

```
at 8am jan 24   (at now reads from standard input)
blob
glob
^D              (end standard input)
```

Execute the shell script *xxx* at 3:30 in the afternoon on Friday of next week:

```
at 1530 fr week xxx
```

FILES

/usr/spool/at/yy.ddd.hhhh.uu	activity to be performed at hour *hhhh* of day *ddd* of year *yy*. *uu* is a unique number.
/usr/spool/at/lasttimedone	contains *hhhh* for last hour of activity.
/usr/spool/at/past	directory of activities now in progress
/usr/lib/atrun	program that executes activities that are due
pwd(1)	

SEE ALSO

calendar(1)
cron – *Installation and System Manager's Guide* Section (8)

DIAGNOSTICS

Complains about various syntax errors and times out of range.

BUGS

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

NAME

awk — pattern scanning and processing language

SYNOPSIS

awk [**−F***c*] [prog] [**−f** pfile] [file] ...

DESCRIPTION

awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally on the command line as *prog*, or in a file specified as **−f** *pfile*. See the document "The awk Programming Language" in the *Support Tools Guide* for information supplementing that given below.

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

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

A pattern-action statement has the form

```
pattern { action }
```

A missing { action } means print the line; a missing pattern always matches.

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

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
```

```
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

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

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

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

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in **egrep** (see **grep**(1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

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

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

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

```
BEGIN { FS = "c" }
```

or by using the −**F***c* option.

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

This could be executed by typing

```
awk 'length > 72' < infile
```

(note use of single quotes) or by placing the program line in a file, called for example *xprog*, and typing

```
awk −f xprog < infile
```

The other examples on this page can likewise by typed on the command line or placed in a program file; the latter approach is usually most convenient for programs longer than a single line.

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

If **awk** seems confusing, a practical example may help. For example, you can use **awk** to transform matrix data from a spreadsheet into input required by a graphics program to generate a bar chart.

First, the matrix data, in the file *profits*:

```
        1978    1979    1980    1981
Sales   2       5       9       15
Costs   1       2.5     4.5     9
Net     1       2.5     4.5     6
```

Now, the **awk** script, which uses the special NF (number of fields) variable:

```
BEGIN       { ncols = 0 }         # initialize ncols

    {
      if (ncols == 0){
              ncols = NF;
              for (c = 1; c <= NF; ++c)
                      col[c] = $c;# read in column headers
      }
```

```
            else {
                      for (rows = 2; rows < ncols +2; + +rows)
                                 print  col[rows-1], $1, $rows
            }
      }
```

The output from this program can then be passed through **sort**(1):

```
      awk  – f awkscript profit | sort > graph.d

      cat graph.d

      1978        Costs      1
      1978        Net        1
      1978        Sales      2
      1979        Costs      2.5
      1979        Net        2.5
      1979        Sales      5
      1980        Costs      4.5
      1980        Net        4.5
      1980        Sales      9
      1981        Costs      9
      1981        Net        6
      1981        Sales  15
```

Voila! Now you are ready to graph the bar chart.

SEE ALSO

lex(1), sed(1)
"The awk Programming Language" in the *Support Tools Guide*.

BUGS

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

NAME

bar — draw a bar chart
hist — draw a histogram

SYNOPSIS

bar [option] ...
hist [option] ...

DESCRIPTION

bar with no options takes input data of two separate types from the standard input as the groups, labels and magnitudes of bars for a bar chart. The delimiter between fields is a tab. The chart is encoded on the standard output for display by the **plot**(1) filters.

Bar input data has three elements: a) a group label; b) a sorting label; and c) the actual value. In a standard bar chart, the x-axis will have data clustered by sorting labels, with the individual bars representing the data value for each unique group. Also, each unique group will have a key appearing in the legend of the chart; the key will either be a type of pattern or color by which the viewer can identify the group within a cluster. Here is a typical input data file:

```
FORD	March	2.5
FORD	April	3.0
GM	March	4.5
GM	April	5.0
```

In this example, FORD and GM are the group labels; March and April, the sorting labels, and the third field carries the data values. In the bar chart, there will be two clusters, one for March and the other for April, each cluster having a bar for FORD and GM.

Up to five bars can be clustered in one group (therefore, up to five(5) legends may be shown at one time). Each legend is given a unique shading when displayed.

hist is almost identical to **bar** in that the effect of all options remains the same. However, as a histogram is essentially a single-group bar chart, the legend distinction is unnecessary and is thus eliminated.

A label may contain blanks and numbers as long as a tab follows it; labels never contain newlines. If labels are too long and/or deep, they may not be printed properly.

The following options are recognized, each as a separate argument.

−c **color**. Specify colors by entering the first letter(s) as separate argument(s). Colors available are:

blue	**g**reen	**c**yan
red	**m**agenta	**y**ellow
white		

The argument(s) entered set color in corresponding, consecutive regions. There are 10 regions total, divided as follows:

1 − 5 bar colors	6 graph & ticks
7 labels except title	8 title
9 gridlines	10 frame

You may enter less than the total number of regions.

−e **no erase**. Save screen, don't erase before plotting. This option is useful for creating composite graphic images.

−f **no frame**. Suppress frame around the window area. All titles and text are normally held inside the frame bounds.

−g **grid**. Next argument is grid style:

2 full grid including frame, ticks and gridlines (default).
1 same as above but without the gridlines.
0 no grid — only a base line will be displayed.

−l **limits**. Next 1 (or 2) arguments are the lower (and upper) vertical limits. Third argument, if present, is grid spacing on the vertical axis. Normally these quantities are determined automatically.

−p **pattern**. Next argument(s) are pattern(s) to fill the bars with. Each argument represents one bar out of up to 5 per cluster. Try out various patterns to see what they are.
If the **−S** option is present, this option will set symbol types. Symbol types available are: 0 box; 1 circle; 2 triangle; 3 star; 4 asterisk; 5 point.
Default is 1 0 2 3 4.

−s[pl] **stack**. Change to additive column (stacked bar) format. (default is side-by-side bars). The **−sp** option displays stacked bars in percentage format, where the total height of each bar = 100%. The **−sl** option provides connect lines between corresponding bar sections (although no grid lines are displayed). The **−spl**, or **−slp**, option will do both.

−S **symbols**. Line-connected symbols are drawn in place of the conventional bars. Next argument(s) are patterns for the lines connecting each set of points. Patterns are: 0, 1 solid; 2 dotted; 3 dashed.

−t **transpose**. Transpose horizontal and vertical axes (option **−y** now applies to the horizontal axis).

−T **title**. When given alone, this option signifies that a title and axis labels are included in the standard input. The program will read in a title and both x and y axis labels as the first three lines of standard input. If a string follows the flag on the command-line, it is taken to be the title. In this case there are no axis labels. Long titles may be broken into two lines by inserting the character '!' (exclamation mark) where line break is desired.

−u **user input**. Input will be read user-input style, as opposed to VENIX database style(default). Both types of input are described below. NOTE: titles must be included in user-style input data.

−w **window**. Specify a window area on the screen. The next four arguments are left, bottom, right, and top of window area in screen coordinates. All chart data will be held inside these boundaries. Coordinates must be entered in global screen units (0.0 to 1.0). For example, **−w .5 0 1 .5** on the command line will put a bar chart in the lower right corner.

−x **no x-axis**. Suppress x-axis labels. (These are the labels for each bar cluster)

−y **no y-axis**. Suppress y-axis labels and tick marks. The y-axis title will remain displayed.

EXAMPLE

To execute the bar chart program, a simple shell command, indicating an entry file, will do:

```
bar [−options] < bardata.dat | plot
```

Database-style input option is shown by example below. (Notice that a title is present and must be indicated with a **−T** flag on the command line: bar −T < bardata.dat | plot .)

```
US Manufacturer Auto Sales in 1982
UNITS SOLD (x1000)
MANUFACTURER

FORD MOTOR COMPANY        First Quarter 6
FORD MOTOR COMPANY        Second Quarter 8
FORD MOTOR COMPANY        Third Quarter 12

GENERAL MOTORS            First Quarter 12
GENERAL MOTORS            Second Quarter 14
GENERAL MOTORS            Third Quarter 9
```

If the user-input option is given, the following format will be read:

```
US Manufacturer Auto Sales in 1982
UNITS SOLD (x1000)
MANUFACTURER

FORD MOTOR COMPANY
        First Quarter     6
        Second Quarter    8
        Third Quarter     12
GENERAL MOTORS
        First Quarter     12
        Second Quarter    14
        Third Quarter     9
```

NOTES

In order to run **bar** or **hist**, the graphics shell parameter $GTERM must be set in the environment. You may place this in your **.profile** or **.login** for convenience.
All data is stored internally, and excess data is dropped.
bar accepts five bars maximum per sorting cluster.
hist accepts only one.

SEE ALSO

plot(1), chart(1g), scat(1g), pie(1g)

NAME

basename — strip filename affixes

SYNOPSIS

basename string [suffix]

DESCRIPTION

basename deletes any prefix ending in '/' and the *suffix*, if present in *string*, from *string*, and prints the result on the standard output. It is normally used inside command substitution marks ` ` in shell scripts.

EXAMPLES

This shell script, when invoked with the argument **/u0/src/cmd/cat.c**, compiles the named file and moves the output to *cat* in the current directory:

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

This shell script, when run with arguments of one or more filenames ending with '.o,' will rename each file to give it a '.save' in place of the '.o.'

```
for N
do
  mv $N `basename $N .o`.save
done
```

SEE ALSO

sh(1)

NAME

basic — UNIX BASIC Interpreter from University of British Columbia

SYNOPSIS

basic [**−anrsxt**] [+n] [file] [plotfile] [args ...]

DESCRIPTION

basic is an ANSI compatible BASIC interpreter that runs under VENIX.

The following options are available:

−a Abort (via IOT trap) on any error. Not for general use.

−n Requires that variable names follow the ANSI standard of a letter followed by an optional digit.

−r Runs the file specified. If +*n* is specified also, then execution will start at line *n*. Basic will exit upon completion of the program.

−s Causes single precision to be used. Normally all floating point numbers are stored in double precision. With the **−s** option, the numbers are stored as single precision, but all expressions are still evaluated in double precision. This option is most useful when a large floating point array is being used and there is not enough storage available for all of it in double precision.

−t Is a debugging option that traces the operation of the interpreter. More **−t** options produce more debugging output.

−x Is a debugging option that suppresses the normal buffering of output.

file An optional file name that specifies a program that is to be read (or run if **−r** was specified) before control is passed to the user.

args Is an option list of arguments that are passed to the program via the ARG$ function. The program may then use the argument strings as it sees fit.

BASIC Commands

The following commands are available to control the running of a BASIC program or interact with the system:

!*command*

causes the VENIX command *command* to be executed, and control to be returned to **basic**.

AUTO [*LINE1*], [*LINE2*]
causes the program to enter into automatic line numbering mode. Line numbering starts at *LINE1* and increments by *LINE2* for each line entered. Automatic numbering mode is terminated by a null line, an attention interrupt, or by any error. A star '*' will be printed if an automatically numbered line already exists in the program. To delete such a line, enter a line with at least one space.

BYE causes **basic** to exit.

CATALOG
prints out the user's current file catalog (equivalent to **ls**(1)).

CLEAR
equivalent to **CLEAR VARS** plus **CLEAR STACK**. This is also done automatically as part of the **RUN** command.

CLEAR VARS
clears the variable symbol table.

CLEAR STACK
clears the execution stack. This is necessary after a "stack overflow" error, if BASIC statements (such as "print i") are to be used.

DEL [*LINE1*], [*LINE2*]
deletes *LINE1* through *LINE2* inclusive. It is also possible to delete a line by just specifying its line number.

DUMP displays the contents of the symbol table. This is mainly for debugging **basic** itself.

EDIT causes the program in memory to be written out into a scratch file and the system file editor (**ed**(1)) invoked upon it. Any changes made to the program will be reflected in the memory version upon leaving the editor.

LIST [*LINE1*], [*LINE2*]
causes the program to be listed from lines *LINE1* through *LINE2*. The default range is the entire program.

LOAD is the same as the **OLD** command.

MERGE [*FILE*]
causes the program in *FILE* to be merged with the current program in memory. Existing lines will be replaced when duplicate line numbers are found.

OLD [*FILE*]
causes the current program to be discarded, and a new program read in from *FILE*. Execution may then be begun via the **RUN** command. Note that all previous variable names are cleared.

RENUMBER [*LINE1*], [*LINE2*]
causes the program in memory to be renumbered, starting at *LINE1*, with an increment of *LINE2*. *LINE1* and *LINE2* both default to 10.

RUN [*LINE*]
causes the program to start execution. If *LINE* is specified, then execution starts at that line number.

SAVE [*FILE*]
causes the program in memory to be saved in *FILE*. *FILE* defaults to the last name mentioned in an **OLD** or **SAVE** command.

SCR erases the program and then clears the symbol table.

UNSAVE [*FILE*]
deletes (unlinks, removes) a file saved via the **SAVE** command.

VI is the same as **ED**, except that the visual editor **vi**(1) is used instead of **ed**.

BASIC STATEMENTS
The following BASIC statements may be issued from a stored program or as a statement typed in immediate execution mode:

CLOSE #*n*
closes unit *n*. It is quite legal to close a unit that has not been opened. When a program begins execution, unit 0 is opened for input on standard input, and unit 1 is opened for output on standard output.

CHAIN *"file"* [,*line*]
replace the current program with the program in the file *"file"*, and start execution at the first line of the program, or at line *line*, if specified. Note that the *"file"* must be either a string expression or a string constant.

DATA *constant-list*
consists of one or more constants separated by commas. Blanks or tabs are not modified in DATA statements.

DEF FN *var*(*args*) = *expr*
: causes a defined function (**FN** *var*) to be defined with the given dummy arguments. When '**FN** *var*(...)' appears in an expression later, the dummy arguments will have real values assigned to them and the value of the expression will be returned.

DIM *var*(*bounds*), ...
: The dimensions of each of the named variables are defined by the bounds provided. Basic allows up to two dimensions for each vector variable. Note that a dimensioned variable is distinct from the scalar variable with the same name. (For example, *a* and *a(i)* are different, for all values of *i*.)

END causes the program to terminate. Some BASIC versions require that **END** only appear on the last line of the program, but this restriction does not apply to VENIX **basic**.

FOR *var* = *expr1* **TO** *expr2* [**STEP** *expr3*]
: The variable specified is assigned the value of *expr1*. The values *expr2* and *expr3* (if specified) are evaluated and saved. The following statements up to the matching **NEXT** are executed until the value of *var* is greater than *expr2*. When the **NEXT** statement is executed, the value of *expr3* is added to *var*. *expr3* defaults to 1, and may be negative. If *expr3* is negative, the **FOR** terminates when *var* is less than *expr2*.

GOSUB *line*
: causes execution to transfer to the line number specified. When a **RETURN** is executed, control will continue from the next statement after **GOSUB**.

GOTO *line*
: causes execution to transfer to the line number specified.

IF *expr* **THEN** *statement*
: If the expression *expr* is TRUE (non-zero), then *statement* will be executed. Some versions of BASIC only allow a statement number or a **GOTO** for *statement*, but here any legal statement is allowed, including another **IF** statement. Other variations of the **IF** statement are supported, where **THEN** is optional, and *statement* is a line number.

INPUT [*#n,*] [*"string",*] *var, ...*
: The variable (or variables) specified are read from the specified unit. If no unit is indicated, unit 0 is used. If the string is

specified, it will be used as a prompt, as if printed via a **PRINT** statement.

LET *var* = *expr*
: assigns the value of *expr* to the variable *var*. Note that **LET** is optional, but should be specified for compatibility with other BASIC's.

NEXT [*var*]
: terminates the preceding **FOR** statement. The value of the **FOR** variable is incremented and if not greater than the limit, the execution continues at the statement following the **FOR** statement.

NOTRACE
: disables the tracing started by the **TRACE** statement.

ON *expr* **GOSUB** *line*, ...
: The value of *expr* is taken as an index into the list of line numbers and transfer is made to the n-th line number specified.

OPEN *file* **FOR** [*INPUT/OUTPUT/APPEND*] #*n*
: opens the file *file* for either input or output on unit *n*. *file* may be any string expression. The *INPUT* statement reads (by default) from unit 0, the **PRINT** statement writes by default to unit 1. *APPEND* is the same as *OUTPUT*, but writes at the end of the file.

PRINT [#*n*,] *expr* [,] [;] ...
: The value of the expression is printed on the specified unit. If no unit is specified, then unit 1 is used. It may be either a number or a character string. If the comma is specified as a delimiter, the values will be aligned in fields 16 columns wide. If a semi-colon is used, there will be no spacing between fields. If there is no delimiter, the last value will be followed by a new line. A **PRINT** statement by itself will issue a *newline*.

PRINT [#*n*,] **USING** *fmt; expr* [,] [;] ...
: The value of the expression is printed in the same fashion as the **PRINT** statement, except that the arithmetic expression is written according to the format specification *fmt*. See the subsection ‘‘PRINT USING Format Specifications.’’

READ *var* ...
: will read values (from subsequent **DATA** statements) into the variables given. An error will occur if there are insufficient values in **DATA** statements.

REM causes the following characters to be taken as a remark or comment. Note that the same effect can be obtained by starting a line with either a single or double quote.

RESTORE
causes the next **READ** statement to start at the first **DATA** statement in the program.

RETURN
causes control to be transferred to the statement following the last **GOSUB** or **ON** *expr* **GOSUB** statement. If there is no enclosing **GOSUB**, an error will occur.

STOP causes the program to terminate with an appropriate comment.

TRACE
causes the line number of each statement to be printed as it is executed. This is sometimes helpful in following the execution of a program that is not behaving properly.

Variables

In ANSI BASIC, a variable must start with a letter, and may have an optional digit following. In VENIX **basic**, only the first two characters of a variable name are considered significant; an arbitrarily long name may be used. The second character does not have to be a digit.

Following the name, either a **%** (percent) or **$** (dollar) sign may be present. The percent sign indicates that the variable is to hold only integer values. The dollar sign indicates that the variable is to hold string values. If neither is present, then the variable will hold floating point values. Real variables are normally double precision unless the **−s** option is specified.

Functions

The following standard functions are provided:

Those marked with an asterisk (*) are not ANSI BASIC and are a local implementation to facilitate communications with the VENIX operating system.

ABS(*value***)**
returns the absolute value of *value*.

ARG$(*n*) *
: returns the *n*-th argument specified when **basic** was opened. Argument zero is the name of the program file (if any). A null string will be returned if an argument is requested that was not specified. This is useful when the **−r** option is used to run a previously stored program.

ATN(*value*)
: returns the arc-tangent (in radians) function of *value*.

CHR$(*n*)
: returns the ASCII character that corresponds to *n*. *n* should normally be an integer in the range $0 \le n \le 127$. Values in the range $128 \le n \le 255$ are possible. All other values are taken modulo 256.

COS(*angle*)
: returns the cosine function of *angle* in radians.

DATE$() *
: returns the current date in the format yy/mm/dd.

EXP(*value*)
: returns the natural anti-logarithm of *value*.

INT(*value*)
: returns the integer value of *value*.

LEFT$(*string, length*)
: the first *length* characters of *string* are returned. It is an error if there are not *length* characters in *string*.

LOG(*value*)
: returns the natural logarithm of *value*.

MID$(*string,start, length*)
: The middle *length* characters of *string* starting at offset *start* (1 = first character) are returned. It is an error if there are not *length* characters in *string* after *start*.

RIGHT$(*string, length*)
: The last *length* characters of *string* are returned. It is an error if there are not *length* characters in *string*.

SIN(*angle*)
: returns the sine function of *angle* in radians.

SQR(*value*)
returns the square root of *value*.

SYS(*n*) *
executes system function *n*, as follows:

1	turn off input echoing
2	turn on input echoing
3	get one character from keyboard
4	print symbol table
5	not implemented
6	not implemented
7	exit from **basic** interpreter

SYSTEM(*"string"*) *
executes *"string"* as a system (VENIX) command, and returns its exit status. Zero means the command terminated properly.

TAN(*angle*)
returns the tangent function of *angle* in radians.

TIME$()
returns the current time in hh:mm:ss format.

OPERATORS

The following operators are available:

+	adds two numeric operands together. Concatenates two strings.
−	subtraction.
*	multiplication.
/	division.
^	exponentiation.
<>	not equal comparison.
=	equal comparison, also assignment operator.
<	less than comparison.
>	greater than comparison.
<=	less than or equal comparison.
>=	greater than or equal comparison.

AND	logical **AND** of two comparisons.
OR	logical **OR** of two comparisons.

PRINT USING Format Specifications

The **PRINT USING** statement format has the following general format ([and] enclose optional features):

[+][**][$$]###[,]###[,]###.##[+]

The characters have the following meanings:

#	indicates that a digit may be placed there.
.	indicates a decimal point.
+	indicates a sign may be placed before (or after) the number.
**	causes leading zeros to be replaced with asterisk fill. (Normally blanks are used).
$$	causes a dollar sign to be placed at the start of the number.
,	A comma is inserted into the number if any digits have been printed. Unlike some versions of BASIC, the position of the comma is significant.

FILES

/usr/bin/basic	**basic** interpreter
/bin/ed	**ed** command editor
/bin/vi	**vi** command editor
/tmp/Ba*	**ed** or **vi** temporary files

BUGS

The **MAT** operation is not implemented.
The **PRINT USING** implements only a minimal subset of the normal facilities.

NAME

bc — arbitrary-precision arithmetic language

SYNOPSIS

bc [−c] [−l] [file ...]

DESCRIPTION

bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. (For a tutorial, see "Arbitrary Precision Desk Calculator Language (BC)" in the *Support Tools Guide*.) It takes input from any files given, then reads the standard input. The **−l** argument stands for the name of an arbitrary precision math library. The syntax for **bc** programs is as follows; *L* means letter a−z, *E* means expression, *S* means statement.

Comments
  enclosed in /* and */

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

Other operands
  arbitrarily long numbers with optional sign and decimal point.
  (*E*)
  sqrt(*E*)
  length(*E*) number of significant decimal digits
  scale(*E*) number of digits right of decimal point
  L(*E* , ... , *E*)

Operators
```
+   −   *   /   %   ^  (% is remainder; ^ is power)
++   −−              (prefix and postfix; apply to names)
==   <=   >=   !=   <   >
=   =+   =−   =*   =/   =%   =^
```

Statements
  E
  { *S* ; ... ; *S* }
  if (*E*) *S*
  while (*E*) *S*
  for (*E* ; *E* ; *E*) *S*

null statement
break
quit

Function definitions

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

Functions in **−l** math library

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

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of **dc**(1). Assignments to **ibase** or **obase** set the input and output number radix respectively.

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

For example

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

```
                    b = b*i
                    c = a/b
                    if(c == 0) return(s)
                    s = s+c
          }
}
```

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

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

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

bc is actually a preprocessor for **dc**(1), which it invokes automatically, unless the **−c** (compile only) option is present. In this case the **dc** input is sent to the standard output instead.

FILES

/usr/lib/lib.b	mathematical library
dc(1)	desk calculator proper

SEE ALSO

dc(1)
"An Arbitrary Precision Desk Calculator Language (BC)" in the *Support Tools Guide.*

BUGS

No &&, ||, or ! operators.
for statement must have all three E's.
quit is interpreted when read, not when executed.

NAME

cal — print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *year* can be between 1 and 9999. *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.
Beware that 'cal 78' refers to the early Christian era, not the 20th century.

NAME

calendar — reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

calendar consults the file 'calendar' in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as 'Dec. 7,' 'december 7,' '12/7,' etc., are recognized, but not '7 December' or '7/12'. On weekends 'tomorrow' extends through Monday. (See ''VENIX Maintenance'' for more details about **cron**.)

When any argument is present **calendar** does its job for every user who has a file **calendar** in his login directory and sends him any positive results by **mail**(1). Normally this is done daily in the wee hours under control of **cron**. See ''VENIX Maintenance'' for more details on **cron**; the control file **/usr/lib/crontab** must be edited to make **calendar** run automatically.

EXAMPLES

A typical calendar file might look like:

```
Apr. 10      do taxes
Apr. 18      phone aunt lia
Apr. 19      board meeting
```

FILES

calendar
/usr/lib/calendar to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
egrep, sed, mail subprocesses

SEE ALSO

at(1), mail(1)
cron – Section (8) in the *Installation and System Manager's Guide*

BUGS

Your calendar must be public information for you to get reminder service.
calendar's extended idea of 'tomorrow' doesn't account for holidays.

NAME

cat — concatenate and print

SYNOPSIS

cat [**−u**] file ...

DESCRIPTION

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

```
cat file
```

prints the file and

```
cat file1 file2 >file3
```

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

If no *file* is given, or if the argument '−' is encountered, **cat** reads from the standard input. Output is buffered in 512-byte blocks unless the standard output is a terminal or the **−u** option is present.

EXAMPLES

Print the contents of *prog.c*:

```
cat prog.c
```

Append file *memo1* to *memo2*:

```
cat memo1 >> memo2
```

(note use of >> to make shell append output).

SEE ALSO

pr(1), cp(1)

BUGS

Beware of 'cat a b >a' and 'cat a b >b', which destroy input files before reading them.

NAME

cb — C program beautifier

SYNOPSIS

cb

DESCRIPTION

cb places a copy of the C program from the standard input on the standard output with spacing and indentation that displays the structure of the program.

EXAMPLES

Beautify program *prog1.c*:

```
cb < prog1.c > beaut
mv beaut prog1.c
```

NAME

cc — C compiler

SYNOPSIS

cc [option] ... file ...

DESCRIPTION

cc is the VENIX C compiler. It normally takes one or more C language source files and compiles and loads them to produce an executable program. Optionally, it can stop compilation at the assembler or object code level, or take assembler, object files, or object libraries, and produce executable programs.

cc accepts several types of file names as arguments:

File names ending with '.c' are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with '.o' substituted for '.c'. The '.o' file is normally deleted, however, if a single C program is compiled and loaded all at one go.

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

Other files are taken as object-code modules or libraries of such modules produced by **ar**(1). Standard VENIX libraries in /**lib** and /**usr/lib** can be referred to using the shorthand provided by the **−l** flag (see below).

The following options are interpreted by **cc**. It is also possible to include loader flags on the same line. See **ld**(1) for load-time options. (Note also that for normal use the loader **−l** flag, used to load an additional library, must come at the very *end* of the command line given **cc**.)

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

−p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls **monitor**(3) 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 Invoke an object-code optimizer.

−f For systems with 8087 floating point chips, generate code which uses 8087 floating point instruction. If this option is not specified, a floating-point simulator is used and 8087 instructions are not generated. Do not use this option if an 8087 chip is not present. (RAINBOW/VENIX only.)

−f For systems without hardware floating-point, generate object code which handles floating-point constants and loads the object program with a floating-point simulator. Do not use if the hardware is present. (VENIX/11 and PRO/VENIX only.)

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

−P Run only the macro preprocessor and place the result for each '.c' file in a corresponding '.i' file that has no '#' lines in it.

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

−z Generate code that checks for stack overflow upon subroutine entry. This option is also passed on to the loader, unless the **−c** flag (suppress loader) option is given and object modules only are produced. In the latter case, the user must be sure to specify a **−z** when the loader is run on the object modules; a **−z** specified for compilation only is ineffective. See **ld**(1) for the **−z** flag effects on the loader. (RAINBOW/VENIX only.)

−m Program is to be "code-mapped." This is only done to run unusually large programs; see documentation in the *Programming Guide* This flag must be used both when compiling and linking programs (similar to the **−z** flag); it is passed to the assembler and the loader. (VENIX/11 and PRO/VENIX only.)

−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 **/usr/include**.

−B*string*
: Find substitute compiler passes in the files named *string* with the suffixes cpp, c0, c1 and c2. If *string* is empty, use a standard backup version. This is only used for debugging or testing new passes. (VENIX/11 and PRO/VENIX only.)

−t[p012]
: Find only the designated compiler passes in the files whose names are constructed by a **−B** option. In the absence of a **−B** option, the *string* is taken to be **/usr/c/**. (VENIX/11 and PRO/VENIX only.)

Any and all object modules and libraries given, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**.

The loader is called in the following way:

ld −X *startoff args* −lc

where *startoff* is one of the possible startoff object files shown below; *args* are any loader arguments and files specified by the user (.c files are given .o extensions) in the order the user gave them; and **−lc** specifies the standard C library.

FILES

file.c	C source file
file.i	preprocessed file
file.s	assembly source file
file.o	object file
a.out	executable output from **cc**
/lib/cpp	preprocessor
/lib/c0	compiler for **cc**
/lib/c1	second compiler pass (VENIX/11 and PRO/VENIX only)
/lib/fc1	floating-point compiler (VENIX/11 and PRO/VENIX only)
/lib/copt	optional optimizer (RAINBOW/VENIX only)
/lib/c2	optional optimizer (VENIX/11 and PRO/VENIX only)
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/fcrt0.o	startoff for floating-point interpretation (VENIX/11 and PRO/VENIX only)

/lib/fmcrt0.o	startoff for profiling w/floating-point (VENIX/11 and PRO/VENIX only)
/lib/libc.a	standard library, see **intro**(3)
/usr/include	standard directory for '#include' files

EXAMPLES

Compile *prog.c* to produce an executable **a.out**:

```
cc prog.c
```

Compile *prog.c* to produce an object file *prog.o*

```
cc −c prog.c
```

Compile *prog.c* and link with math library:

```
cc prog.c −lm
```

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, 1978
"C Language" in the *Programming Guide.*
monitor(3), prof(1), adb(1), ld(1)

DIAGNOSTICS

The diagnostics produced by C itself are intended to be largely self-explanatory. Occasional messages may be produced by the assembler or loader. See **as**(1) and **ld**(1) for these errors.

The following errors may be produced by various stages of the C compiler (explanations are given as deemed necessary):

XXX? Can not create output file *XXX*. The file may not exist, or may exist and not be writable, or the directory in which it lies may not be writable

0-length row: XXX

XXX: actuals too long
Arguments passed when invoking macro function are too long

Arg count Compiler pass called with wrong number of command line arguments

−B overwrites earlier option

bad formal: XXX Illegal argument used when declaring macro function

Bad func. storage class

bad include syntax	Bad '#include' usage
Bad structure/union/enum name	
Bad type for field	Illegal type declaration in bitfield
Binary expression botch	Internal error
Botch in outcode	Internal error
Break/continue error	Use of 'break' or 'continue' not within bounds of 'while', 'for', etc.
C error: const	
Call of nonUfunction	Attempt to call something which is not a function
Can't create temp	Can't create temporary storage file in /tmp
Can't create XXX	Can't create file: perhaps no write permission in directory?
Can't find include file XXX	
Can't find XXX	
Case not in switch	Use of 'case' statement outside of 'switch'
Can't create temp	Can't create temporary storage file in /tmp
compiler botch: call	Internal error
Compiler error (length)	Internal error
Compiler error: pname	Internal error
Compound statement required	Opening braces do not follow beginning of function
Conflict in storage class	
Constant required	Frequently due to use of variable in 'case' (only constants allowed)
Declaration syntax	
Default not in switch	'default' case is not within switch construction
Disallowed conversion	
Divide check	Constant expression has divide by zero
Duplicate case (%d)	Same 'case' appears more than once within switch
excessive −I file (XXX) ignored	
Expression input botch	Internal error
expression overflow	Expression too complicated

Expression syntax	
External definition syntax	
extraneous name XXX	
−f overwrites earlier option	
Fatal error	Unrecoverable error in compiler execution. Will occur when evaluating floating-point expressions if no floating point hardware exists and **−f** flag is not used with **cc**. Otherwise due to system error.
Floating % not defined	% (modulo operator) illegal for floating point variables
If-less else	Use of preprocessor '#else' without previous '#if'
If-less endif	Use of preprocessor '#endif' without previous '#if'
Illegal #	Illegal preprocessor command
Illegal character X in preprocessor if	
Illegal conditional	
Illegal conversion	
Illegal enum constant for XXX	
Illegal enumeration XXX	
Illegal indirection	Often due to use of non-pointer variable as a pointer
Illegal initialization	
Illegal lvalue	See 'Lvalue required'
Illegal number XXX	
Illegal operation on structure	Structures may not be added together, assigned, etc.
Illegal storage class	?
Illegal structure operation	(See 'Illegal operation on structure')
Illegal structure ref	
Illegal type of operand	
Illegal use of register	For example, attempt to take address of register variable
Illegal use of type	
Illegal use of type name	
Inappropriate 'else'	Not matched with 'if'
Inappropriate parameters	
Incompatible structures	
Intermediate file error	Intermediate temporary file was corrupted

Long character constant	char's can't be long
Lvalue required	An 'lvalue' is the expression on the left side of an equation, i.e. one which is assigned a value. Expressions which evaluate as constants can not be assigned to (e.g. '3 = x').
XXX: macro recursion	Preprocessor macro refers to itself
Misplaced 'long'	
Misplaced 'unsigned'	
Missing '}'	Unbalanced braces
Missing temp file	Temporary file disappeared from /tmp
XXX multiply defined	Symbol XXX set more than once
More than 1 'default'	Only one 'default' case allowed in 'switch'
No auto. aggregate initialization	Automatic arrays (those declared local to a function) may not be initialized; make them static or external
No code table for op: XXX	Operation not valid, e.g. '++' or '−−' operation on floats or doubles
No field initialization	Bit fields can not be initialized
No match for op	Illegal operation
No source file XXX	
no space	Out of memory
no space for file names	
No strings in automatic	Automatic (local) strings can not be initialized; make them static or extern.
Nonterminated comment	Open '/*' without closing '*/'
Nonterminated string	No closing quote
Not an argument: XXX	Function declaration implies XXX was passed as an argument
Null dimension	Zero-dimensioned array
Out of space	Not enough memory for internal tables
pname called illegally	Internal error

pow2 botch	Internal error
Rank too large	
XXX redeclared	
Register overflow: simplify expression	Expression too complicated
Stack overflow botch	Internal error
Statement syntax	
Switch table overflow	Too many cases in 'switch'
Symbol table overflow	Too many symbols
−t overwrites earlier option	
token too long	
too many −D options, ignoring XXX	
too many −U options, ignoring XXX	
too many defines	
Too many DIUC options	
too many formals: XXX	Too many arguments given in declaration of macro function
Too many initializers: XXX	
Too many object/library files	
Too many source files	
Too many structure initializers	
Too many structure members	
Too many }'s	
too much defining	
XXX: too much pushback	
Type clash	Types of variables don't mix in expression
Type is too complicated	
XXX undefined	Variable XXX was not declared prior to use
undefined control	Bad use of preprocessor control statements
Undefined structure	
Undefined structure initialization	
Unexpected EOF	Source file ended without legally terminating construction; e.g., no closing brace at end.
Unimplemented field operator	Operation not possible with bitfields
unimplemented structure assignment	
Unknown character	

unknown flag XXX
Unknown keyword
Unreasonable include nesting
XXX: unterminated macro call
Warning: X= operator assumed
Better to use + =, *=, − = etc. rather than old syntax of = +, =* ...

(Warning only)	Indicates a message was a warning, not a fatal error

Warning: possibly too much data
Warning: structure redeclaration
Warning: very large data structure
Would overwrite XXX
Attempt to use your input file as output file, as in 'cc −o file.c file.c'

Write error on temp
Error when writing on temporary file in /**tmp**.

NAME

cd — change working directory

SYNOPSIS

cd directory

DESCRIPTION

directory becomes the new working directory. The user 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. It is therefore recognized and executed by the Shell.

cd without an argument brings the user back to his or her home directory. There's no place like $HOME..

EXAMPLES

Change into directory *data*:

```
cd data
```

Change into home directory:

```
cd
```

SEE ALSO

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

NAME

chart — draw a flow/organizational chart

SYNOPSIS

chart [option] ...

DESCRIPTION

chart with no options takes a series of nodes and linkages from the standard input and creates any one of many types of organizational charts. Each line of input contains information for one node. Included in the input are: 1) the node number, 2) informative text, 3) box type, and 4) a number identifying a following linked node.
Here is an example of chart input data:

```
1    J&J Company    5    2
1    J&J Company    5    3
1    J&J Company    5    4
2    R&D Dept.      2    0
3    Personnel      2    0
4    Marketing      2    0
```

This is an example of a simple organization chart for the 'J&J Company'. All text in the second field will be displayed inside text-boxes.
Box types for the third field are identified as follows:

0 simple box frame	1 octagonal frame
2 hexagonal	3 double box
4 double octagon	5 double hexagon

The example shows that node #1 links with three other boxes denoting the various departments of the company. Also note that the data line for node #1 must be repeated for each linkage it contains. The database can be used to create data such as above from much simpler user input.

The normal delimiter between input fields is a tab. The chart is encoded on the standard output for display by the **plot**(1) filters.

The following options are recognized, each as a separate argument.

−c **color**. Specify colors by entering the first letter(s) as separate argument(s). Colors available are:

blue	**g**reen	**c**yan
red	**m**agenta	**y**ellow
white		

The argument(s) entered set color in corresponding, consecutive regions. There are 10 regions total, divided as follows:

1 – 6 boxes by level	7 linkage
8 box text	9 title
10 frame	

You may always enter less than the maximum number of arguments.

–e **no erase**. Save screen, don't erase before plotting. This is useful for producing multiple images on the same screen.

–f **no frame**. Suppress frame around the window area. All titles and text are normally held inside the frame bounds.

–l[a] **linkage**. Next argument is the type of linkage desired between the nodal boxes. Connecting linkage modes are: 0 – consecutive, 1 – pick. Consecutive linkages are those normally found in flow charts, where each pair of boxes are connected with a unique line. The pick linkage is that found in a company organizational chart. If the **a** option is given, arrows will be drawn on each linkage, pointing from parent node to follower node.

–t[lcr] **text**. Text placement inside nodal boxes. The **l** option will cause all text to be left-flushed with respect to each box. Likewise, the **c** and **r** options will respectively cause centering and right-flushing of text. (Centering is the default.)

–T **title**. When this option is given alone, the program will read a title as the first line from the standard input. If a string follows the flag on the command-line, then this is taken to be the title and no title line will be read from input. Long titles may be broken into two lines by inserting the character '!' (exclamation mark) where a line break is desired.

–w **window**. Specify window area on screen. Next four(4) arguments are left, bottom, right, and top of window area in screen coordinates. (screen coordnates are 0.0 to 1.0). For example, the command-line option **–w .5 .5 1.0 1.0** will put a graph in the upper-right hand corner of the screen.

EXAMPLES

Draw a chart from coordinates in a file called *company*, and pipe it to the plot filter:

```
chart < company | plot
```

NOTES

In order to run **chart**, the graphics shell parameter $GTERM must be set in your environment. You may do this in your **.profile** or **.login** for convenience.

SEE ALSO

spline(1), plot(1), bar(1g), scat(1g), pie(1g)

NAME

chmod — change mode

SYNOPSIS

chmod [− **v**] mode file ...

DESCRIPTION

The mode of each named file is changed according to *mode,* which may be absolute or symbolic. If the − **v** (verify) flag is given, the new mode of the file will be listed, in the style of **ls**(1).

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** ('all') stands for **ugo**. If *who* is omitted, the default is **a** but the setting of the file creation mask (see **umask**(2)) is taken into account.

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

permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group id) and **t** (save text − sticky). Letters **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**.

Only the owner of a file (or the super-user) may change its mode. See **chmod**(2) for information on sticky bit usage.

EXAMPLES

Deny write permission to others on file *file1*:

```
chmod o-w file1
```

Make *file1* executable:

```
chmod +x file1
```

Change all files in current directory to read/write permission for user only:

```
chmod 600 *
```

SEE ALSO

ls(1), chmod(2), chown (1), stat(2), umask(2)

BUGS

Attempts to set an illegal mode (for instance, a non-super-user trying to set the sticky bit) are silently ignored.

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 UID 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 GID or a group name found in the group-ID file.

Only the super-user can change owner or group, in order to simplify as yet unimplemented accounting procedures.

EXAMPLES

Change user-ID of all files in current directory to "fred", and change group-ID to "panda":

```
chown fred *
chgrp panda *
```

FILES

/etc/passwd
/etc/group

SEE ALSO

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

NAME

cmp — compare two files

SYNOPSIS

cmp [**−l**] [**−s**] file1 file2

DESCRIPTION

cmp compares two files. (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.

EXAMPLES

Compare files *a.out* and *prog*:

```
        cmp a.out prog
```

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

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

NAME

col — filter reverse line feeds

SYNOPSIS

col [**−bfx**]

DESCRIPTION

col reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ESC − 7 in ASCII) and by forward and reverse half line feeds (ESC − 9 and ESC − 8). **col** is particularly useful for filtering multicolumn output made with the **.rt** command of **nroff**(1) and output resulting from use of the **tbl**(1) preprocessor. Simple line backspacing can also be achieved in **nroff**, such as **.sp −2**, with the **col** filter.

Although **col** accepts half line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full line boundary. This treatment can be suppressed by the **−f** (fine) option; in this case the output from **col** may contain forward half line feeds (ESC − 9), but will still never contain either kind of reverse line motion.

If the **−b** option is given, **col** assumes that the output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

The control characters SO (ASCII code 017), and SI (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

col normally converts white space to tabs to shorten printing time. If the **−x** option is given, this conversion is suppressed.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by one of 7, 8, 9, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

EXAMPLES

Filter reverse line feeds from 2-column output of **nroff** and send to terminal at */dev/tty01*:

```
nroff paper | col > /dev/tty01
```

SEE ALSO

nroff(1), tbl(1)

BUGS

Can't back up more than 128 lines.
No more than 800 characters, including backspaces, on a line.

NAME

comm — select or reject lines common to two sorted files

SYNOPSIS

comm [– [**123**]] file1 file2

DESCRIPTION

comm reads *file1* and *file2,* which should be ordered in ASCII collating sequence, and produces a three column output: lines only in *file1;* lines only in *file2;* and lines in both files. The filename '–' means the standard input. **comm** is only useful for list type files.

Flags **1**, **2**, or **3** suppress printing of the corresponding column. Thus **comm –12** prints only the lines common to the two files; **comm –23** prints only lines in the first file but not in the second; **comm –123** is a no-op.

EXAMPLES

Print lines common to files *sort1* and *sort2*:

```
comm –12 sort1 sort2
```

SEE ALSO

cmp(1), diff(1), uniq(1)

NAME

cp — copy

SYNOPSIS

cp [**−i**] file1 file2

cp [**−i**] file ... directory

cp [**−i**] **−r** directory1 directory2

DESCRIPTION

cp copies files and directories. In the first usage shown above, *file1* is copied onto *file2*. *file2* will be overwritten. The mode and owner of *file2* are preserved if it already existed; the mode of the source file is used otherwise.

In the second form, one or more *files* are copied into the *directory* with their original file-names maintained.

In the third form (with the **−r** flag), **cp** recursively copies all files and sub-directories under *directory1* to *directory2*.

With the **−i** (interactive) flag, **cp** asks for user permission before copying each file; respond with a 'y' or 'n'.

cp refuses to copy a file onto itself.

EXAMPLES

Copy *prog1* to */u0/fred/bin/newprog*:

```
cp prog1 /u0/fred/bin/newprog
```

Copy all files in current directory to directory */u0/fred/bin*, keeping the same names; ask permission before copying each file:

```
cp −i * /usr/fred/bin
```

Copy all files (including directories and sub-directories) under directory *src* to directory *newsrc*:

```
cp −r src newsrc
```

SEE ALSO

cat(1), pr(1), mv(1)

NAME

crypt — encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

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

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

will print the "clear", that is, the decrypted version.

Files encrypted by **crypt** are compatible with those treated by the editor **ed**(1) in encryption mode.

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

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

Since the key is an argument to the **crypt** command, it is potentially visible to users executing **ps**(1). To minimize this possibility, **crypt** takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of **crypt**.

EXAMPLES

Encrypt file *message1* with key '*pluto*,' calling encrypted file *message1.cpt*; remove *message1*.

```
crypt pluto < message1 > message1.cpt
rm message1
```

Uncrypt *message1.cpt* into *message1*; remove *message.cpt*:

```
crypt pluto < message.cpt > message1
rm message1.cpt
```

FILES

/dev/tty	— for typed key
/usr/lib/makekey	

SEE ALSO

ed(1), crypt(3)
makekey – section (8) in the *Installation and System Manager's Guide*

BUGS

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, VenturCom assumes no responsibility for their use by the recipient. Further, VenturCom assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

NAME

csh – a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**–cefinstvVxX**] [arg ...]

DESCRIPTION

csh is a command language interpreter. Beginners should read "An Introduction to the C Shell" in the *User Guide* for a tutorial.

csh begins by executing commands from the file **.cshrc** in the home directory of the invoker. If this is a login shell then it also executes commands from the file **.login** there. In the normal case, the shell will then begin reading commands from the terminal, prompting with "%". Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into words. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file **.logout** in the user's home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters '&' '|' ';' '<' '>' '(' ')' form separate words. If doubled in '&&', '| |', '< <' or '> >' these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with '\'. A newline preceded by a '\' is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, '′', '`' or '″', form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of ` ′ or '″' characters a newline preceded by a '\' gives a true newline character.

When the shell's input is not a terminal, the character '#' introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by '\' and in quotations using '`', '′', and '″'.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by '|' characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ';', and are then executed sequentially. A sequence of pipelines may be executed without waiting for it to terminate by following it with an '&'. Such a sequence is automatically prevented from being terminated by a hangup signal; the **nohup** command need not be used.

Any of the above may be placed in '(' ')' to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with '| |' or '&&' indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See "Expressions.")

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions can be used to reintroduce sequences of words from previous commands, possibly performing modifications on these words. Thus history substitutions provide a generalization of a *redo* function.

History substitutions begin with the character '!' and may begin anywhere in the input stream if a history substitution is not already in progress. This '!' may be preceded by a '\' to prevent its special meaning; a '!' is passed unchanged when it is followed by a blank, tab, newline, '=' or '('. History substitutions also occur when an input line begins with '^'. This special abbreviation will be described later.

Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list, the size of which is controlled by the **history** variable. The previous command is always retained. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the history command:

```
 9  write michael
10  ex write.c
11  cat oldwrite.c
12  diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the **prompt** by placing an '!' in the prompt string.

With the current event 13 we can refer to previous events by event number '!11', relatively as in '! − 2' (referring to the same event), by a prefix of a command word as in '!d' for event 12 or '!w' for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially a *redo*. The form '!#' references the current command (the one being typed in). It allows a word to be selected from further left in the line, to avoid retyping a long name, as in '!#:1'.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of a input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

0	first (command) word
n	*n*'th argument
^	first argument, i.e. '1'
$	last argument
%	word matched by (immediately preceding) ?*s*? search
x − *y*	range of words
− *y*	abbreviates '0 − *y*'
*	abbreviates '^ − $', or nothing if only 1 word in event
*x**	abbreviates '*x* − $'
x −	like '*x**' but omitting word '$'

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '^', '$', '*' '−' or '%'. After the optional word designator can be placed a sequence of modifiers, each preceded by a ':'. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.xxx' component, leaving the root name.
s/l/r	Substitute *l* for *r*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. 'g&'.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a 'g' the modification is applied only to the first modifiable word. In any case it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of '/'; a '\' quotes the delimiter into the *l* and *r* strings. The character '&' in the right hand side is replaced by the text from the left. A '\' quotes '&' also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in '!?*s*?'. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing '?' in a contextual scan.

A history reference may be given without an event specification, e.g. '!$'. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus '!?foo?^ !$' gives the first and last arguments from the command matching '?foo?'.

A special abbreviation of a history reference occurs when the first nonblank character of an input line is a '^'. This is equivalent to '!:s^' providing a convenient shorthand for substitutions on the text of the previous line. Thus '^lb^lib' fixes the spelling of 'lib' in the previous command. Finally, a history substitution may be surrounded with '{' and '}' if necessary to insulate it from the characters which follow. Thus, after 'ls −ld ~paul' we might do '!{l}a' to do 'ls −ld ~paula', while '!la' would look for a command starting 'la'.

Quotations with ' and "

The quotation of strings by '' and '"' can be used to prevent all or some of the remaining substitutions. Strings enclosed in '' are prevented any further interpretation. Strings enclosed in '"' are yet variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see "Command Substitution" below) does a '"' quoted string yield parts of more than one word; '' quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the **alias** and **unalias** commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for **ls** is **ls −l** the command **ls /usr** would map to **ls −l /usr**, the argument list here being undisturbed. Similarly if the alias for **lookup** was **grep !^ /etc/passwd** then **lookup bill** would map to **grep bill /etc/passwd**.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can "alias print 'pr \!* | lpr −n'" to make a command which **pr**'s its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the **set** and **unset** commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the **verbose** variable is a toggle which causes command input to be echoed. The setting of this variable results from the **−v** command line option.

Other operations treat variables numerically. The **@** command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '$' characters. This expansion can be prevented by preceding the '$' with a '\' except within '"'s where it always occurs, and within ''s where it never occurs. Strings quoted by '`' are interpreted later (see "Command substitution" below) so '$' substitution does not occur there until later, if at all. A '$' is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in '"' or given the ':q' modifier the results of variable substitution may eventually be command and filename substituted. Within '"' a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the ':q' modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters, digits, and underscores.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}

May be used to select only some of the words from the value of *name*. The selector is subjected to '$' substitution and may consist of a single number or two numbers separated by a '−'. The first word of a variables value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '$#name'. The selector '*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}

Gives the number of words in the variable. This is useful for later use in a '[selector]'.

$0

Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}

Equivalent to '$argv[number]'.

$*

Equivalent to '$argv[*]'.

The modifiers ':h', ':t', ':r', ':q' and ':x' may be applied to the substitutions above as may ':gh', ':gt' and ':gr'. If braces '{' '}' appear in the command form then the modifiers must appear within the braces. The current implementation allows only one ':' modifier on each '$' expansion.

The following substitutions may not be modified with ':' modifiers.

$?name
${?name}

Substitutes the string '1' if name is set, '0' if it is not.

$?0

Substitutes '1' if the current input filename is known, '0' if it is not.

$$

Substitute the (decimal) process number of the (parent) shell.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in '``'. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within '"'s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters '*', '?', '[' or '{' or begins with the character '~', then that word is a candidate for filename substitution, also known as 'globbing'. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters '*', '?' and '[' imply pattern matching, the characters '~' and '{' being more akin to abbreviations.

In matching filenames, the character '.' at the beginning of a filename or immediately following a '/', as well as the character '/' must be matched explicitly. The character '*' matches any string of characters, including the null string. The character '?' matches any single character. The sequence '[...]' matches any one of the characters enclosed. Within '[...]', a pair of characters separated by '–' matches any character lexically between the two.

The character '~' at the beginning of a filename is used to refer to home directories. Standing alone, i.e. '~' it expands to the invoker's home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and '–' characters the shell searches for a user with that name and substitutes their home directory; thus *~ken* might expand to */usr/ken* and *~ken/chmach* to */usr/ken/chmach*. If the character '~' is followed by a character other than a letter or '/' or appears not at the beginning of a word, it is left undisturbed.

The metanotation 'a{b,c,d}e' is a shorthand for 'abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus *~source/s1/{oldls,ls}.c* expands to */usr/source/s1/oldls.c /usr/source/s1/ls.c* whether or not these files exist without any chance of error if the home directory for 'source' is */usr/source*. Similarly *../{memo,*box}* might expand to *../memo ../box ../mbox*. (Note that 'memo' was not sorted with the results of matching '*box'.) As a special case '{', '}' and '{ }' are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

< name
: Open file *name* (which is first variable, command and filename expanded) as the standard input.

< < word
: Read the shell input up to a line which is identical to *word*. *word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting ‘\’, ‘"’, ‘'’ or ‘`’ appears in *word* variable and command substitution is performed on the intervening lines, allowing ‘\’ to quote ‘$’, ‘\’ and ‘`’. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name
: The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, it is truncated, its previous contents being lost.

  If the variable **noclobber** is set, then the file must not exist or be a character special file (e.g. a terminal or ‘/dev/null’) or an error results. This helps prevent accidental destruction of files. In this case the ‘!’ forms can be used and suppress this check.

  The forms involving ‘&’ route the diagnostic output into the specified file as well as the standard output. *name* is expanded in the same way as ‘<’ input filenames are.

> > name
> >& name
> >! name
> >&! name
: Uses file *name* as standard output like ‘>’ but places output at the end of the file. If the variable **noclobber** is set, then it is an error

for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

If a command is run detached (followed by '&') then the default standard input for the command is the empty file '/dev/null'. Otherwise the command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the **@**, **exit**, **if**, and **while** commands. The following operators are available:

```
||  &&  |  ^  &  ==  !=  <=  >=  <  >  <<  >>  +  -  *  /  %
!  ~  (  )
```

Here the precedence increases to the right, '==' and '!=', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '*' '/' and '%' being, in groups, at the same level. The '==' and '!=' operators compare their arguments as strings, all others operate on numbers. Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '-*l* name' where *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable **status** examined.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The **foreach**, **switch**, and **while** statements, as well as the **if – then – else** form of the **if** statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias
alias *name*
alias *name wordlist*

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name; wordlist* is command and filename substituted. *name* is not allowed to be **alias** or **unalias.**

break

Causes execution to resume after the end of the nearest enclosing **forall** or **while**. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw

Causes a break from a **switch**, resuming after the **endsw**.

case *label:*

A label in a **switch** statement as discussed below.

cd
cd *name*
chdir
chdir *name*

Change the shell's working directory to directory *name*. If no argument is given then change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory (and does not begin with '/', './', or '../'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

continue

Continue execution of the nearest enclosing **while** or **foreach**. The rest of the commands on the current line are executed.

default:

Labels the default case in a **switch** statement. The default should come after all **case** labels.

echo *wordlist*

The specified words are written to the shell's standard output. A '\c' causes the echo to complete without printing a newline, akin to the '\c' in **nroff**(1). A '\n' in wordlist causes a newline to be printed. Otherwise the words are echoed, separated by spaces.

else
end
endif
endsw

See the description of the **foreach**, **if**, **switch**, and **while** statements below.

exec *command*

The specified command is executed in place of the current shell.

exit
exit(*expr*)

The shell exits either with the value of the **status** variable (first form) or with the value of the specified *expr* (second form).

foreach *name* (*wordlist*)
...
end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching **end** are executed. (Both **foreach** and **end** must appear alone on separate lines.)

The builtin command **continue** may be used to continue the loop prematurely and the builtin command **break** to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob *wordlist*

Like **echo** but no '\' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto *word*

The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

history

Displays the history event list.

if (*expr*)*command*

If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the **if** command. *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is not executed (this is a bug).

if (*expr*)*then*
...
else if (*expr2*) *then*
...
else
...
endif

If the specified *expr* is true then the commands to the first **else** are executed; else if *expr2* is true then the commands to the second else are executed, etc. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** part is likewise optional. (The words **else** and **endif** must appear at the beginning of input lines; the **if** must appear alone on its input line or after an **else**.)

login

Terminate a login shell, replacing it with an instance of /**bin**/**login**. This is one way to log off, included for compatibility with /**bin**/**sh**.

logout

Terminate a login shell. Especially useful if **ignoreeof** is set.

nice
nice +*number*
nice *command*
nice +*number command*

The first form sets the **nice** for this shell to 4. The second form sets the **nice** to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using 'nice −number ...'. Command is always executed in a sub-shell, and the restrictions place on commands in simple **if** statements apply.

nohup
nohup *command*

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. Unless the shell is running detached, **nohup** has no effect. All processes detached with '&' are automatically **nohup**'ed. (Thus, **nohup** is not really needed.)

onintr
onintr −
onintr *label*

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form **onintr** − causes all interrupts to be ignored. The final form causes the shell to execute a **goto** label when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of **onintr** have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

rehash

Causes the internal hash table of the contents of the directories in the **path** variable to be recomputed. This is needed if new commands are added to directories in the **path** while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat *count command*

The specified *command* which is subject to the same restrictions as the *command* in the one line **if** statement above, is executed *count* times. I/O redirections occurs exactly once, even if *count* is 0.

set
set *name*
set *name=word*
set *name[index]=word*
set *name=(wordlist)*

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index*'th component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv *name value*

Sets the value of environment variable *name* to be *value*, a single string. Useful environment variables are 'TERM' the type of your terminal and 'SHELL' the shell you are using.

shift
shift *variable*

The members of *argv* are shifted to the left, discarding *argv*[1]. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source *name*

The shell reads commands from *name*. **source** commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a **source** at any level terminates all nested **source** commands. Input during **source** commands is never placed on the history list.

switch (*string*)
case *str1*:
   ...
   breaksw
...
default:
   ...
  breaksw
endsw

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '[...]' may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command **breaksw** causes execution to continue after the **endsw**. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the **endsw**.

time
time *command*

With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the **time** variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask *value*

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias *pattern*

All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be **unaliased**.

unhash

Use of the internal hash table to speed location of executed programs is disabled.

unset *pattern*

All variables whose names match the specified pattern are removed. Thus all variables are removed by **unset ***; this has noticeably distasteful side-effects. It is not an error for nothing to be **unset**.

wait

All child processes are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and process numbers of all children known to be outstanding.

while (*expr*)
...
end

While the specified expression evaluates non-zero, the commands between the **while** and the matching end are evaluated. **break** and **continue** may be used to terminate or continue the loop prematurely. (The **while** and **end** must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the **foreach** statement if the input is a terminal.

@
@ *name = expr*
@ *name[index] = expr*

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains '<', '>', '&' or '|' then at least this part of the expression must be placed within '(' ')'. The third form assigns the value of *expr* to the *index*'th argument of *name*. Both *name* and its *index*'th component must already exist.

The operators '*=', '+=', etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix '++' and '−−' operators increment and decrement *name* respectively, i.e. '@ i++'.

Pre-defined variables

The following variables have special meaning to the shell. Of these, **argv, child, home, path, prompt, shell** and **status** are always set by the shell. Except for **child** and **status** this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

The shell copies the environment variable PATH into the variable *path*, and copies the value back into the environment whenever *path* is set. Thus it is not necessary to worry about its setting other than in the file **.cshrc** as inferior **csh** processes will import the definition of **path** from the environment.

argv	Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. '$1' is replaced by '$argv[1]', etc.
cdpath	Gives a list of alternate directories searched to find subdirectories in **chdir** commands.
child	The process number printed when the last command was forked with '&'. This variable is **unset** when this process terminates.
echo	Set when the −**x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.
histchars	Can be assigned a two character string. The first character is used as a history character in place of '!', the second character is used in place of the '^' substitution mechanism. For example, 'set histchars = ",;"' will cause the history characters to be comma and semicolon.
history	Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of **history** may run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the invoker, initialized from the environment. The filename expansion of '~' refers to this variable.

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.

If the first word of the value of **mail** is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says 'New mail in *name*' when there is mail in the file *name*.

noclobber As described in the section on "Input/output," restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that '>>' redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. 'echo [' still gives an error.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no **path** variable then only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the super-user the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the **−c** nor the **−t** option will normally hash the contents of the directories in the **path** variable after reading **.cshrc**, and each time the **path** variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the **rehash** or the commands may not be found.

prompt	The string which is printed before each command is read from an interactive terminal input. If a '!' appears in the string it will be replaced by the current event number unless a preceding '\' is given. Default is '% ', or '# ' for the super-user.
shell	The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of "Non-builtin Command Execution" below.) Initialized to the (system-dependent) home of the shell.
status	The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status '1', all other builtin commands set status '0'.
time	Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.
verbose	Set by the **−v** command line option, causes the words of each command to be printed after history substitution.

Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via **exec**(2). Each word in the variable **path** names a directory from which the shell will attempt to execute the command. If it is given neither a **−c** nor a **−t** option, the shell will hash the names in these directories into an internal table so that it will only try an **exec** in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via **unhash**), or if the shell was given a **−c** or **−t** argument, and in any case for each directory component of **path** which does not begin with a '/', the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus '(cd ; pwd) ; pwd' prints the home directory; leaving you where you were (printing this after the home directory), while 'cd ; pwd' leaves you in the home directory. Parenthesized commands are most often used to prevent **chdir** from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an **alias** for **shell** then the words of the alias will be prepended to the argument list to form the shell command. The first word of the **alias** should be the full path name of the shell (e.g. '$shell'). Note that this is a special, late occurring, case of **alias** substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell is '–' then this is a login shell. The flag arguments are interpreted as follows:

–c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in **argv**.

–e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

–f The shell will start faster, because it will neither search for nor execute commands from the file '.cshrc' in the invoker's home directory.

–i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

–n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

–s Command input is taken from the standard input.

–t A single line of input is read and executed. A '\' may be used to escape the newline at the end of this line and continue onto another line.

−v Causes the **verbose** variable to be set, with the effect that command input is echoed after history substitution.

−x Causes the **echo** variable to be set, so that commands are echoed immediately before execution.

−V Causes the **verbose** variable to be set even before '.cshrc' is executed.

−X Is to **−x** as **−V** is to **−v**.

After processing of flag arguments if arguments remain but none of the **−c**, **−i**, **−s**, or **−t** options was given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by '$0'. (Since many systems use the standard Bourne shell (see **sh**(1)) whose scripts are not compatible with the C shell, the C shell will execute the Bourne shell on scripts whose first character is not a '#', i.e. if the script does not start with a comment. Since Bourne shell scripts never begin with a '#', compatibility between the two shells is maintained.) Remaining arguments initialize the variable **argv**.

Signal handling

The shell normally ignores *quit* signals. The *interrupt* and *quit* signals are ignored for an invoked command if the command is followed by '&'; otherwise the signals have the values which the shell inherited from its parent. The shell's handling of interrupts can be controlled by **onintr**. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file '.logout'.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after '.cshrc' at login.
~/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a '#'.
/tmp/sh*	Temporary file for '<<'.
/dev/null	Source of empty file.
/etc/passwd	Source of home directories for '~name'.

LIMITATIONS

Words can be no longer than 512 characters. The number of characters in an argument varies from system to system. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Also command substitutions may substitute no more characters than are allowed in an argument list.

To detect looping, the shell restricts the number of **alias** substitutions on a single line to 20.

SEE ALSO

access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(4), environ(5), ''An Introduction to the C Shell'' in the *User Guide*

BUGS

Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with '|', and to be used with '&' and ';' metasyntax.

Commands within loops, prompted for by '?', are not placed in the **history** list.

It should be possible to use the ':' modifiers on the output of command substitutions. All and more than one ':' modifier should be allowed on '$' substitutions.

Some commands should not touch **status** or it may be so transient as to be almost useless. Oring in 0200 to **status** on abnormal termination is a kludge.

There are a number of bugs associated with the importing/exporting of the PATH. For example, directories in the path using the ~ syntax are not expanded in the PATH. Unusual paths, such as (), can cause a core dump.

NAME

cu — call UNIX

SYNOPSIS

cu telno [−**t**] [−s speed] [−**a** acu] [−**l** line]

DESCRIPTION

cu calls up another UNIX or VENIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of text files (transfer of binary files is not possible). *telno* is the telephone number, with minus signs at appropriate places for delays. (If you do not have an automatic dialer, *telno* should be a dummy number.) The −**t** flag is used to dial out to a terminal. *speed* gives the transmission speed (110, 150, 300, 1200, 2400, 4800, 9600); 300 is the default value.

The −**a** and −**l** values may be used to specify pathnames for the ACU and communications line devices. They can be used to override the following built-in defaults:

−**a** /dev/null
−**l** /dev/com1.dout

If no automatic call-up port is used, then *telno* can be any dummy number, and the default −**a /dev/null** should be used. For ease of use, **mknod**(1) can be used to make up a /dev/com1.dout which points to whatever terminal port is actually used. You must have both read and write permission on this line.

After making the connection, **cu** runs as two processes: the *send* process reads the standard input and passes most of it to the remote system; the *receive* process reads from the remote system and passes most data to the standard output. In general use, the *send* and *receive* processes simply allow you to converse with the remote system: the *send* process sends over everything you type at the keyboard (without echoing); the *receive* process writes to your terminal everything the other system types back (including any echoing of what you just typed). It is possible to specify diversions to allow the redirection of I/O, however, by beginning a line with a '~' (this means that files being sent should not have lines beginning with '~'). Be aware that the *send* and *receive* processes handle these diversions separately: *send* diversions can only be generated directly from your keyboard, and *receive* diversions can only be generated from the other system.

The *send* process interprets the following:

~.	terminate the conversation.
~EOT	terminate the conversation.
~<file	send the contents of *file* to the remote system, as though typed at the terminal.
~!	invoke an interactive shell on the local system.
~!cmd ...	run the command on the local system (via **sh −c**). See **sh**(1).
~$cmd ...	run the command 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' name is used both places.
~%put from [to]	copy file 'from' (on local system) to file 'to' on remote system. If 'to' is omitted, the 'from' name is used both places.
~~...	send the line '~...'.

The *receive* process handles output diversions of the following form:

~>[>][:]file

zero or more lines to be written to file

~>

In any case, output is diverted (or appended, if '>>' used) to the file. If ':' is used, the diversion is *silent,* i.e., it is written only to the file. If ':' is omitted, output is written both to the file and to the standard output. The trailing '~>' terminates the diversion.

The use of ~%**put** requires **stty** 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** and **tee**(1) on the remote system. Also, **stty tabs** mode is required on the remote system if tabs are to be copied without expansion.

The following shell script can be run on the remote system to send a list of files to the local system:

```
for f
do
        echo "~>$f"
        cat $f
        echo "~>"
done
```

For every file specified, the script sends to your *receive* process the proper diversion code, the file, and then a diversion terminator.

EXAMPLES

The following might be done to log in to a remote system and exchange some files. The user logs in on the remote system as user "phil". After exchanging files, the user writes a letter to phil; presumably the real Phil will read it when he logs in.

```
cu 1 -s 300        (modem is 300 baud; dummy telno given)
login phil         (now talking to remote machine)
ls                 (list files in phil's home directory)
~%take prog1.c     (copy prog1.c from remote system to yours)
~%put junk.c       (copy junk.c from your system to remote)
mail phil          (send mail to phil)
Phil -
           I copied prog1.c, and left you
           junk.c
                             -Fred
^D                 (cntrl-D to end mail)
login              (log out of remote machine)
~.                 (exit from cu)
```

cu can be used to talk with non-UNIX systems, so long as these systems can be told to type the appropriate '~' escape sequences. The 'put' and 'take' commands will probably not work, since they do assume certain UNIX commands for creating these sequences.

cu handles XON/XOFF protocols for both transmitting and receiving.

FILES

/dev/com1.dout
/dev/null

/bin/sh	
/bin/echo, /bin/tee, /bin/stty	commands sent to remote system

SEE ALSO

ttys(4)

DIAGNOSTICS

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

BUGS

The syntax is unique.

Lines of text which begin with '~' will be interpreted as commands when sent over; if this is a problem, filter your files to add an additional '~' to the beginning of any such line.

When executing local commands, a new shell is run to handle each command. This means that doing a **~!cd** will have no value, since the current directory is changed only for the duration of the new shell.

To do several local commands in a row, type **~!sh** to start up a temporary shell which will stay in effect until a ^D is typed.

NAME

date — print and set the date

SYNOPSIS

date [**−l**] [yymmddhhmm [.ss]]

DESCRIPTION

If no argument is given, the current date and time are printed. If an argument is given, the current date is set. *yy* is the last two digits of the year; 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; *.ss* is optional and is the seconds. The year, month and day may be omitted, the current values being the defaults. The system operates in GMT. **date** takes care of the conversion to and from local standard and daylight time.

The **−l** flag causes the current date to be loaded from the DEC Professional battery backed-up clock (PRO/VENIX only). This clock maintains the correct date even when the hardware is turned off, so that the system administrator need not set the date himself. Normally, '**date −l**' is executed from the **/etc/rc** file on boot-up.

EXAMPLES

Set the date to Oct 8, 12:45 AM.

```
date 10080045
```

Print today's date:

```
date
```

FILES

/usr/adm/wtmp	to record time-setting

SEE ALSO

utmp(4)

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. The **−l** flag will produce an error if the battery backed-up clock is unreliable (usually due to power being off for a prolonged period), in which case the date must be set manually.

NAME

dc — desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

dc is an arbitrary precision arithmetic package. (For a tutorial, see "Interactive Desk Calculator Language (DC)" in the *Support Tools Guide*.) Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of **dc** is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number
: The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9. It may be preceded by an underscore _ to input a negative number. Numbers may contain decimal points.

+ – / * % ^
: The top two values on the stack are added (+), subtracted (–), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x* The top of the stack is popped and stored into a register named *x,* where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x* The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack and in registers are printed.

q exits the program. If executing a string, the recursion level is popped by two. **Q** pops the top value from the stack and the string execution level is popped by that value.

x treats the top element of the stack as a character string and executes it as a string of dc commands.

X replaces the number on the top of the stack with its scale factor.

[...] puts the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x*
The top two elements of the stack are popped and compared. Register *x* is executed if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! interprets the rest of the line as a VENIX command.

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input. **I** pushes the input base on the top of the stack.

o The top value on the stack is popped and used as the number radix for further output. **O** pushes the output base on the top of the stack.

k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

Z replaces the number on the top of the stack with its length.

? A line of input is taken from the input source (usually the terminal) and executed.

; : are used by **bc**(1) for array operations.

EXAMPLES

Print the first ten values of n! is

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1), which is a preprocessor for **dc** providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

''Interactive Desk Calculator Language DC'' in the *Support Tools Guide*.

DIAGNOSTICS

'x is unimplemented' where x is an octal number.
'stack empty' for not enough elements on the stack to do what was asked.
'Out of space' when the free list is exhausted (too many digits).
'Out of headers' for too many numbers being kept around.
'Out of pushdown' for too many items on the stack.
'Nesting Depth' for too many levels of nested execution.

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, that is, to specify that transfers be done in chunks greater than the default 512 bytes.

option	*values*
if=	input file name. Standard input is default so you can also use < redirection
of=	output file name *which must already exist.* Standard output is default; you can also use > redirection. If > redirection is used, the output file is created if it does not exist, and if it does exist, *it is cleared.*
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 copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
files=*n*	copy *n* files from (tape) input
seek=*n*	seek *n* records from beginning of output file before copying
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

The record size (used for the **skip**, **seek** and **count** options) is the same as the "block" size.

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. All numbers are specified in decimal.

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

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

EXAMPLES

Copy 640 blocks from floppy unit 0 to unit 1; use transfer size of 10 blocks:

```
dd if=/dev/f0 of=/dev/f1 bs=10b count=64
```

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

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

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968.

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

NAME

deroff — remove nroff, tbl and neqn constructs

SYNOPSIS

deroff [**−w**] file ...

DESCRIPTION

deroff reads each file in sequence and removes all **nroff** command lines, backslash constructions, macro definitions, **neqn** constructs (between '.EQ' and '.EN' lines or between delimiters), and table descriptions and writes the remainder on the standard output. **deroff** follows chains of included files ('.so' and '.nx' commands); if a file has already been included, a '.so' is ignored and a '.nx' terminates execution. If no input file is given, **deroff** reads from the standard input file.

If the **−w** flag is given, the output is a word list, one 'word' (string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed) per line, and all other characters ignored. Otherwise, the output follows the original, with the deletions mentioned above.

EXAMPLES

Strip nroff constructions from file *paper* and type on terminal:

```
deroff paper
```

Strip nroff constructions and save output in file *paper.strip:*

```
deroff paper > paper.strip
```

SEE ALSO

nroff(1), neqn(1), tbl(1)

BUGS

deroff is not a complete **nroff** interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

NAME

df — disk free

SYNOPSIS

df [filesystem] ...

DESCRIPTION

df prints out the number of free blocks available on the *filesystems.* If no file system is specified, the set of default file systems in **/etc/checklist** is taken. **df** only works on VENIX filesystems. It cannot be used with **tar,** **dump**, or DOS format diskettes.

EXAMPLES

Print number of free blocks on the winchester hard disk (whose partitions are given in **/etc/checklist**):

```
df
```

Print number of free blocks on a VENIX floppy diskette:

```
df /dev/f0
```

FILES

/etc/checklist	default file system list

SEE ALSO

du(1), fsck(1)

NAME

diff — differential file comparator

SYNOPSIS

diff [**−efbh**] file1 file2

DESCRIPTION

diff finds lines differing between two text files. It 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 whose file-name is the same as the file-name of *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**(1) 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**.

diff is designed to work on text files only; **cmp**(1) should be used to compare binaries.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

EXAMPLES

Print differences between *file1* and *file1.bak*:

```
diff file1 file1.bak
```

SEE ALSO

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

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

BUGS

Editing scripts produced under the **−e** or **−f** option are naive about creating lines consisting of a single '**.**'.

NAME

dtree — print the tree structure of a directory

SYNOPSIS

dtree [**−adfglnpsvx**] [**−c** length] [dir]

DESCRIPTION

dtree prints a directory as a tree structure, with the branches on the right of the page. If *dir* is specified, it is used as the root of the tree to print; the default is to use the current directory. The current date is output as the first line.

The following options are allowed, and may be given in any order:

−a	include files as well as directories in listing
−d	sort tree with directories at the top
−f	sort tree with files at the top
−g	same as **−l**, but use group name instead of user name
−l	Print stats with each listing. If both **−g** and **−l** flags are given, both owner and group will be printed.
−n	Do not sort the tree.
−p	include files starting with a '.' (except '.' and '..').
−s	Use shorter stats. Implies **−l** if **−g** isn't given.
−v	Variable length columns.
−x	Do not cross mounted file systems.
−c	Set max column length to be *length*.

FILES

/etc/passwd	to get user ID's for **dtree −l**.
/etc/group	to get group ID's for **dtree −g**.

SEE ALSO

date(1), pwd(1), stat(2), getgrent(3), dir(4), types(5)

AUTHOR

Courtesy of St. Olaf College.

NAME

du — summarize disk usage

SYNOPSIS

du [−s] [−**a**] [name ...]

DESCRIPTION

du gives the number of blocks contained in all files and (recursively) directories within each specified directory or file *name*. If *name* is missing, the current directory ('.') is used.

The optional argument −s causes only the grand total 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.

A file which has two links to it is only counted once.

EXAMPLES

Summarize disk usage for the user area under directory /usr, giving totals for all subdirectories too:

```
du /usr
```

Print total number of blocks in use under current directory, but don't print individual totals for subdirectories:

```
du −s
```

A block in VENIX is 512 bytes. Divide the number of blocks by two to get disk usage in K bytes.

SEE ALSO

df(1), quot(1)

BUGS

Non-directories given as arguments (not under −**a** option) are not listed. If there are too many distinct linked files, **du** counts the excess files multiply.

NAME

dump — incremental file system dump

SYNOPSIS

dump [keys [arguments]] filesystem

DESCRIPTION

dump makes an incremental file system dump, on tape or disk, of all files changed after a certain date. It is somewhat faster than **tar(1)**, but as it works on on entire file systems at once, **dump** does not match **tar**'s ease of use in selectively extracting or saving individual files or directories. Unlike **tar**, **dump** does not provide compatibility or portability to other versions of UNIX.

The default output device is floppy drive 0 (**/dev/f0**) on PRO/VENIX, RAINBOW/VENIX, and VENIX/11 on Micro PDP-11 systems; it is magtape drive 0 (**/dev/mt0**) on other VENIX/11 systems. This default may be overridden with the **f** modifier (see below).

The *key* argument specifies the date and other options about the dump. *key* consists of one or more characters from the following set:

a Normally files larger than 1000 blocks are not incrementally dumped; this flag forces them to be dumped.

b The next argument is taken to be the maximum size of the dump medium in blocks. If the **b** key is not used, a default size is assumed. On PRO/VENIX and RAINBOW/VENIX, the default is the size of a standard diskette (790 and 640 blocks, respectively); on VENIX/11, it is 24200 blocks (the approximate size of a 2200 foot, 800 bpi tape — see **s** key).

c If the medium overflows, increment the last character of its name and continue on that drive. (Normally it asks you to change media).

f Place the dump on the next argument file instead of the default device.

i The dump date is taken from the entry in the file **/etc/dtab** corresponding to the last time the file system was dumped with the **u** option.

0 The dump date is taken as the epoch (beginning of time). Thus this option causes an entire file system to be taken.

h The dump date is some number of hours before the current date. The number of hours is taken from the next argument in *arguments*.

d The dump date is some number of days before the current date. The number of days is taken from the next argument in *arguments*.

u The file **/etc/dtab** is updated to indicate when this dump was made, to allow subsequent dumps on the same file system to save only files changed since this date.

s The size of the dump magtape is specified in feet (useful for VENIX/11 only). The number of feet is taken from the next argument in *arguments*. It is assumed that there are 11 VENIX blocks (512 bytes/block) per foot.

When the specified size is reached on the output media, **dump** will prompt for it to be changed. The **restor**(1) command is used to restore dump media.

EXAMPLES

Full dumps should be taken on quiet file systems as follows:

```
dump 0u /dev/w0.usr
ncheck /dev/w0.usr > nch
```

The **ncheck**(1) will come in handy in case it is necessary to restore individual files from this dump. Incremental dumps should be taken when desired by:

```
dump
```

When the incremental dumps get cumbersome, a new complete dump should be taken. In this way, a restore requires loading of the complete dump medium and only the latest incremental medium.

To dump the entire file system on **/dev/w0.sys**:

```
dump 0 /dev/w0.sys
```

To dump the files on **/dev/w0.usr** modified in the last 7 days to floppies of length 320 blocks on unit */dev/f1*:

```
dump dfb 7 /dev/f1 320 /dev/w0.usr
```

Note that the arguments for the keys 'dfb' are given in order, following the keys themselves.

FILES

/dev/f0	default output device (PRO/VENIX and RAINBOW/VENIX)
/dev/mt0	default output device (VENIX/11)
/etc/dtab	dump record

SEE ALSO

restor(1), tar(1), "VENIX Maintenance" in the *Installation and System Manager's Guide*.

DIAGNOSTICS

If the dump requires more than one medium, it will ask you to change media. Reply with a newline when this has been done. If the first block on the new medium is not writable, e.g because you forgot the write tab, you get a chance to fix it. Generally, however, read or write failures are fatal.

NAME

echo — echo arguments

SYNOPSIS

echo [**−n**] [arg] ...

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a newline on the standard output. If the flag **−n** is used, no newline is added to the output.

echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, do 'echo ... 1>&2'.

echo exists as a built-in command when running the C shell (**csh**), and has slightly different options. See **csh**(1) for details.

EXAMPLES

Echo the word "*tree*" into the file *list*:

```
echo tree >list
```

Print the names of the files in the current directory:

```
echo *
```

Print the name of the home directory:

```
echo $HOME
```

NAME

ed — text editor

SYNOPSIS

ed [−] [−x] [name]

DESCRIPTION

ed is the standard line-oriented text editor. It is an alternative to **vi**(1) and used chiefly with hard-copy terminals.

ed is more fully described in *Introducing the UNIX System*, chapter 7, by Henry McGilton and Rachel Morgan. This tutorial is recommended reading for new users.

If a *name* 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. If **−x** is present, an **x** command is simulated first to handle an encrypted file. The optional '−' suppresses the printing of character counts by **e**, **r**, and **w** commands.

ed operates on a copy of any file it is editing; changes made in 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*.

Commands to **ed** have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to 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. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus **\ [.** and sometimes ^ * $.

2. A **.** matches any character.

3. A \ followed by any character except a digit or () matches that character.

4. A nonempty string *s* bracketed [*s*] (or [^*s*]) matches any character in (or not in) *s*. In *s*, \ has no special meaning, and] may only appear as the first letter. A substring *a*−*b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.

5. A regular expression of form 1−4 followed by * matches a sequence of 0 or more matches of the regular expression.

6. A regular expression, *x*, of form 1−8, bracketed \(*x* \) matches what *x* matches.

7. A \ followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *n*-th \(matched.

8. A regular expression of form 1−8, *x*, followed by a regular expression of form 1−7, *y* matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.

9. A regular expression of form 1−8 preceded by ^ (or followed by $), is constrained to matches that begin at the left (or end at the right) end of a line.

10. A regular expression of form 1−9 picks out the longest among the leftmost matches in a line.

11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

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; however, the exact effect on the current line is discussed under the description of the 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 name *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.

5. A regular expression enclosed in slashes '/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.

6. A regular expression enclosed in queries '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.

7. An address followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that address plus (resp. 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 (resp. subtracted). As a consequence of this rule and rule 8, the address '−' refers to the line before the current line. Moreover, trailing '+' and '−' characters have cumulative effect, so '− −' refers to the current line less 2.

10. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is equivalent to '−'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma ','. They may also be separated by a semicolon ';'. In this case the current line '.' is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches ('/', '?'). The second address of any two-address sequence must correspond to a line following 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, but are used to show that the given addresses are the default.

As mentioned, it is generally illegal for more than one command to appear on a line. However, most commands may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed respectively in the way discussed below.

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.

(., .)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(., .)d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit 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. The number of characters read is typed. 'filename' is remembered for possible use as a default file name in a subsequent **r** or **w** command. If 'filename' is missing, the remembered name is used.

E filename

This command is the same as *e*, except that no diagnostic results when no *w* has been given since the last buffer alteration.

f filename

The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1,$) g/regular expression/command list

In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '.' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with '\'. *A*, *i*, and *c* commands and associated input are permitted; the '.' terminating input mode may be omitted if it would be on the last line of the command list. The commands *g* and *v* are not permitted in the command list.

(.)i
<text>
.

This command inserts the given text before the addressed line. '.' is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

(., .+1)j

This command joins the addressed lines into a single line; intermediate newlines simply disappear. '.' is left at the resulting line.

(.)k*x*

The mark command marks the addressed line with name *x*, which must be a lower-case letter. The address form ′*x* then addresses this line.

(., .)l

The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The *l* command may be placed on the same line after any non-I/O command.

(., .)m*a*

The move command repositions the addressed lines after the line addressed by *a*. The last of the moved lines becomes the current line.

(.,.)p
: The print command prints the addressed lines. '.' is left at the last line printed. The *p* command may be placed on the same line after any non-I/O command.

(.,.)P
: This command is a synonym for *p*.

q
: The quit command causes **ed** to exit. No automatic write of a file is done.

Q
: This command is the same as *q*, except that no diagnostic results when no *w* has been given since the last buffer alteration.

($)r filename
: The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). The file name is remembered if there was no remembered file name already. 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 left at the last line read in from the file.

(., .) s/regular expression/replacement/ or,
(., .) s/regular expression/replacement/g
: The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, 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 regular expression and the replacement. '.' is left at the last line substituted.

  An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of '&' in this context may be suppressed by preceding it by '\'. The characters '*n*' where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

Lines may be split by substituting new-line characters into them. The new-line in the replacement string must be escaped by preceding it by '\'.

(. , .)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 on the last line of the copy.

(. , .)u

The undo command restores the preceding contents of the current line, which must be the last line in which a substitution was made.

(1, $) v/regular expression/command list

This command is the same as the global command *g* except that the command list is executed *g* with '.' initially set to every line *except* those matching the regular expression.

(1, $) w filename

The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writable by everyone). The file name is remembered if there was no remembered file name already. If no file name is given, the 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 printed.

(1,$)W filename

This command is the same as *w*, except that the addressed lines are appended to the file.

x A key string is demanded from the standard input. Later *r*, *e* 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 **sh**(1) to be interpreted as a command. '.' is unchanged.

(. + 1) <newline>

An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '. + 1p'; it is useful for stepping through text.

If an interrupt signal (^C) 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 temporary file. The limit on the number of lines depends on the amount of core: each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.

FILES

/tmp/e*	temporary files
ed.hup	work is saved here if terminal hangs up
/usr/lib/makekey	used for encryption

SEE ALSO

McGilton & Morgan, *Introducing the UNIX System*. 1983
''Advanced Editing'' in the *Document Processing Guide*
ex(1), vi(1), sed(1), crypt(1)

DIAGNOSTICS

'?name' for inaccessible file; '?' for errors in commands; '?TMP' for temporary file overflow.

To protect against throwing away valuable work, a *q* or *e* command is considered to be in error, unless a *w* has occurred since the last buffer change. A second *q* or *e* will be obeyed regardless.

BUGS

The *l* command mishandles DEL.
A *!* command cannot be subject to a *g* command.
Because 0 is an illegal address for a *w* command, it is not possible to create an empty file with **ed**.

NAME

edebug — debug interpreted Pascal programs

SYNOPSIS

edebug [−**l**xxx] [−**f**xxx] [−**c**xxx] [−**p**xxx] [source.p]

DESCRIPTION

The EM-1 interpreter,**em1**(1), has several debugging features built in. They can be activated by flag options to **em1**(1), or **pc**(1). The EM-1 interpreter collects the information while it runs the program. When the program is terminated, the interpreter dumps this information onto files with fixed names. **edebug** reads these files and makes a listing of the source program, augmented with the collected information. Four debugging features are available. If none are given, −**l** is the default.

−**l** em1_last	Print contents of the circular buffer used to keep track of the last 64 executed source lines (after execution error only).
−**f** em1_flow	Print source file, marking executed lines with an asterisk. A bit map for all source lines tells which lines are executed.
−**c** em1_count	Count the number of times each source line was entered.
−**p** em1_profile	Estimate the number of memory cycles spent on each source line.

The file name given after each flag is the default; if the information has been placed in another file, that name can be given after the flag.

The most common use of edebug is to print the numbers of the last executed source lines if an execution error occurred. This information is always available on *em1_last* after run time errors, which is printed by default (−**l** option).

If the *source.p* argument is given, then a complete listing of the source program is produced, including all available debugging information. If at least one of the flag options of edebug is given, then only the debugging files for the mentioned flags are read.

EXAMPLES

Print out numbers of last 64 lines before run-time error:

```
edebug
```

Tell all lines executed during interpretation *prog.p*:

```
edebug −f prog.p
```

As above, but with the contents of *em1_flow* moved to file *oldflow*:

```
edebug −foldflow prog.p
```

FILES

em1_last, em1_flow, em1_count, em1_profile

SEE ALSO

pc(1), em1(1)

BUGS

Procedures with no statements are not counted, although they might have been executed.

NAME

em1 — calling program for Pascal interpreter

SYNOPSIS

em1 [−d] [−n] [loadfile [args ...]]

DESCRIPTION

em1 interpretively executes the named Pascal **em1** module ('e.out' if not specified).

−d Collect information useful for debugging programs. An estimate of the number of memory cycles used per source line is collected in the file *em1_profile*; a bit map of all source lines that have been executed is written into the file *em1_flow*; a count of line usage giving the number of times every source line was entered is placed in the file *em1_count*. **edebug**(1) can be used to print out this information.

−n Don't do debugging.

Debugging may have been previously specified with the **−d** flag of **pc**(1). If so, it can optionally be turned off at run-time with the **−n** flag here. If debugging was not previously specified, it can be activated by the **−d** flag here.

FILES

/lib/em1/em1-----t	interpreter for no floating point, no debugging
/lib/em1/em1--cfpt	interpreter for debugging, no floating point
/lib/em1/em1-r---t	interpreter for floating point, no debugging
/lib/em1/em1-rcfpt	interpreter for floating point and debugging
em1_profile	profile data
em1_count	source line count data
em1_flow	source line flow data
em1_last	last lines executed

SEE ALSO

edebug(1), pc(1)

DIAGNOSTICS

Error messages are self explanatory.

NAME

erase — graphics and text screen erase

SYNOPSIS

erase [**−dusrlchw**]

DESCRIPTION

erase provides simple and fancy ways to clear the screen.

erase called without a flag will clear the screen quickly and painlessly. Fancy erasures can be obtained by calling **erase** with one of the flag options, described below:

−d	(down)	Sweeps horizontally from top to bottom.
−u	(up)	Sweeps from bottom to top.
−s	(sandwich)	Closes in from top and bottom simultaneously.
−r	(right)	Sweeps from left to right.
−l	(left)	Sweeps from right to left.
−c	(compress)	Closes in from left and right. A horizontal sandwich.
−h	(hole)	Closes in from all sides.
−w	(wow!)	Special cosmic erase. Try it!

SEE ALSO

plot(1g), plot(3g), plot(5)

NAME

ex — text editor

SYNOPSIS

ex [**−**] [**−v**] [**−t** tag] [**−r**] [+*command*] [**−l**] name ...

DESCRIPTION

ex is a line editor contained within **vi(1)**. **ex** is a superset of another VENIX editor **ed**(1). Most users start off with **vi**, and then use **ex** at their convenience. Or, you can start editing in **ex**, and then go into **vi** by typing "**:vi**".

See "*An Introduction to Display Editing with Vi*" in the *User Guide* for more information about the display editor **vi**.

FILES

/usr/lib/ex*strings	error messages
/etc/termcap	describes capabilities of terminals
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

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

BUGS

The **undo** command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

undo never clears the buffer modified condition.

The **z** command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line '−' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

Entering/leaving ex

% **ex** *name*	edit *name*, start at end
% **ex** +*n name*	... at line *n*
% **ex** **−t** *tag*	start at *tag*
% **ex** *name* **...**	edit first; rest via **:n**
% **ex** **−R** *name*	read only mode
: x	exit, saving changes
: q!	exit, discarding changes

Ex states

Command	Normal and initial state. Input prompted for by **:**. Your kill character cancels partial command.
Insert	Entered by **a i** and **c**. Arbitrary text then terminates with line having only **.** character on it or abnormally with interrupt.
Open/visual	Entered by **open** or **vi**, terminates with **Q** or ^\.

Ex commands

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**	open	**o**	unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	*window*	**z**
insert	**i**	rewind	**rew**	*escape*	**!**
join	**j**	set	**se**	*lshift*	**<**
list	**l**	shell	**sh**	*print next*	CR
map		source	**so**	*resubst*	**&**
mark	**ma**	stop	**st**	*rshift*	**>**
move	**m**	substitute	**s**	*scroll*	**^D**

Ex command addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*

+	next	*x*,*y*	*x* through *y*
−	previous	′*x*	marked with *x*
+*n*	*n* forward	′′	previous context
%	1,$		

Specifying terminal type

% **setenv TERM** *type*	*csh*
$ TERM=*type*; **export TERM**	*sh*

Some terminal types

2621	43	adm31	dw1	h19
2645	733	adm3a	dw2	i100
300s	745	c100	gt40	mime
33	act4	dm1520	gt42	owl
37	act5	dm2500	h1500	PC
4014	adm3	dm3025	h1510	vt52

Initializing options

EXINIT	place **set**'s here in environment var.
set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x***?**	show value of option *x*

Useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
list		print ^I for tab, $ at end
magic		**. [*** special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^**D**
showmatch	sm	to) and } as typed
slowopen	slow	choke updates during insert
window		visual mode lines

wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

↑	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str***]**	any char in *str*
[↑*str***]**	... not in *str*
[*x*−*y***]**	... between *x* and *y*
*****	any number of preceding

NAME

expr — evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

expr evaluates numerical and string expressions. It is chiefly used within shell scripts.

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expr |expr
: yields the first **expr** if it is neither null nor '0', otherwise yields the second **expr**.

expr &expr
: yields the first **expr** if neither **expr** is null or '0', otherwise yields '0'.

expr relop **expr**
: where *relop* is one of < <= = != >= >, yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both **expr** are integers, otherwise lexicographic.

expr + expr
expr – expr
: addition or subtraction of the arguments.

expr * expr
expr / expr
expr % expr
: multiplication, division, or remainder of the arguments.

expr : expr
: The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of **ed**(1). The **\(...\)** pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

(**expr**) parentheses for grouping.

EXAMPLES

To assign 20 to the Shell variable *A*, add 1 and print it:

```
A=20
A=`expr $A + 1`
echo $A
```

(note the use of grave accents to execute **expr**).

To find the filename part (least significant part) of the pathname stored in variable *A*, which may or may not contain '/':

```
expr $A : '.*/\(.*\)' '|' $A
```

Note the quoted shell metacharacters.

expr is intended for use by shell scripts under the Bourne shell (see **sh**(1)). While it may also be used by C-shell scripts, better facilities exist that are built into that shell, which make **expr** unecessary. See **csh**(1).

SEE ALSO

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

DIAGNOSTICS

expr returns the following exit codes:

0	if the expression is neither null nor '0',
1	if the expression is null or '0',
2	for invalid expressions.

NAME

f77 — Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

f77 is the VENIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.r' are taken to be Ratfor source programs; these are first transformed by the **ratfor**(1) preprocessor, then compiled by **f77**.

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

The following options have the same meaning as in **cc**(1). See **ld**(1) for load-time options.

−c Suppress loading and produce '.o' files for each source file.

−p Prepare object files for profiling, see **prof**(1).

−O Invoke an object-code optimizer.

−S Compile the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

−f Use a floating point interpreter (for PDP-11's that lack floating point hardware).

−o *output*
Name the final output file *output* instead of **a.out**.

The following options are peculiar to **f77**.

−onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

−u	Make the default type of a variable 'undefined' rather than using the default Fortran rules.
−C	Compile code to check that subscripts are within declared array bounds.
−w	Suppress all warning messages. If the option is **−w66**, only Fortran 66 compatibility warnings are suppressed.
−F	Apply the Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to '.f', but do not compile.
−m	Apply the M4 preprocessor to each '.r' file before transforming it with the Ratfor preprocessor.
−R*x*	Use the string *x* as a Ratfor option in processing '.r' files.

Other arguments are taken to be either loader option arguments, or **f77**-compatible object programs, typically produced by an earlier run, or perhaps libraries of **f77**-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**.

EXAMPLES

Compile program *prog.f* into executable file *prog*:

```
f77 −o prog prog.f
```

Compile program *main.f part1.f part2.f* into object module:

```
f77 −c main.f part1.f part2.f
```

Produce executable file *prog* from object modules

```
f77 −o prog main.o part1.o part2.o
```

FILES

file.[fresc]	input file
file.o	object file
a.out	loaded output
/lib/f77pass1	compiler
/lib/c1	pass 2
/lib/c2	optional optimizer
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library, see section 3
ratfor(1)	Ratfor preprocessor

SEE ALSO

"Fortran 77" in the *Programming Guide*.
prof(1), cc(1), ld(1)

DIAGNOSTICS

The diagnostics produced by **f77** itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

BUGS

The Fortran 66 subset of the language has been exercised extensively; the newer features have not.

NAME

file — determine file type

SYNOPSIS

file file ...

DESCRIPTION

file performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, **file** examines the first 512 bytes and tries to guess its language. **file** will indicate if the file is ASCII text, command text (an executable shell script), or executable code. If the file is executable code, **file** will indicate whether the file is stripped.

BUGS

It often makes mistakes. In particular it often suggests that command files are C programs.

NAME

find — find files

SYNOPSIS

find pathname-list expression

DESCRIPTION

find recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, −*n* means less than *n* and *n* means exactly *n*.

−**name** *filename*
: True if the *filename* argument matches the current file name. Normal Shell argument syntax may be used if escaped (watch out for '[', '?' and '*').

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

−**type** *c* True if the type of the file is *c*, where *c* is **b**, **c**, **d** or **f** for block special file, character special file, directory or plain file.

−**links** *n* True if the file has *n* links.

−**user** *uname*
: True if the file belongs to the user *uname* (login name or numeric user ID).

−**group** *gname*
: True if the file belongs to group *gname* (group name or numeric group ID).

−**size** *n* True if the file is *n* blocks long (512 bytes per block).

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

−**atime** *n* True if the file has been accessed in *n* days.

−**mtime** *n* True if the file has been modified in *n* days.

−**exec** *command*
: True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an

escaped semicolon. A command argument '{ }' is replaced by the current pathname.

−ok *command*
: Like **−exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

−print Always true; causes the current pathname (to the matched file) to be printed to the standard output. If **−print** is not given, file names will not be printed.

−newer *file*
: True if the current file has been modified more recently than the argument *file*.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).
2) The negation of a primary ('!' is the unary **not** operator).
3) Concatenation of primaries (the **and** operation is implied by the juxtaposition of two primaries).
4) Alternation of primaries ('**−o**' is the **or** operator).

EXAMPLES

To find all files in directory */u1* called *animal*:

```
find /u1 −name animal −print
```

To remove all files named *a.out* or **.o* that have not been accessed for a week:

```
find / \( −name a.out −o −name '*.o' \)\
   −atime +7 −exec rm { } \;
```

To find all files owned by guest or demo in */u0*:

```
find /u0 \( −user guest −o −user demo) −print
```

Note the use of parentheses. Without them, the **−print** would apply only to **−user** *demo*; consequently, files owned by guest (**−user** *guest*) would never be listed.

FILES

/etc/passwd
/etc/group
/bin/pwd

SEE ALSO

sh(1), test(1), filsys(4)

BUGS

The syntax is painful.

NAME

format — format a floppy diskette

SYNOPSIS

format [−sdni] device

DESCRIPTION

format formats and automatically makes a file system on a floppy diskette. On PRO/VENIX, floppy formatting is not possible; **format** is useful only for creating file systems on factory-formatted diskettes.

File systems need only be created on diskettes which the user intends to mount (see **mount**(1)); they are not necessary on diskettes used for **tar** or **dump**(1).

All the blocks on the floppy diskette are checked and reported on the screen.

The following options may be used:

−s	Format a single-sided floppy diskette (if hardware permits).
−d	Format a double-sided floppy (if hardware permits). If no **−s** option is specified, the default **−d** is assumed.
−n	Do not make a file system on the floppy diskette.
−i	Ignore bad blocks; the file system will be made much more quickly.

device is the drive to format, either **/dev/f0** or **/dev/f1**.

EXAMPLES

Format floppy in drive 0:

```
format /dev/f0
```

Format a single-sided floppy in drive 0:

```
format −s /dev/f0
```

Format a single-sided floppy in drive 1 and do not make a file system:

```
format −s −n /dev/f1
```

or

```
format −sn /dev/f1
```

SEE ALSO

mkfs(1m)

NAME

fsck — file system consistency check and interactive repair

SYNOPSIS

fsck [**−syn**] [filesystem] ...

DESCRIPTION

fsck audits VENIX file systems for consistency and corrects any discrepancies. Since these corrections will, in general, result in the loss of data, **fsck** will request operator concurrence for each action. All questions should be answered by typing 'y' or 'n', followed by a newline. However, if **fsck** does not have write access to the file system, or the 'no' option, **−n**, is on, then all question will automatically be answered 'no'. Alternatively the 'yes' option, **−y**, will cause all questions to be answered 'yes'.

If no files are specified, the default file system list in **/etc/checklist** will be consulted.

The program consists of 6 phases. Some phases are skipped if they are not needed.

1. All block pointers in all files are examined, checking for pointers which are outside the file system (BAD) and for blocks which appear in more then one file (DUP). A table is made of all DUP blocks and all defective files are marked for clearing. Each error is printed, but no corrections take place at this time.

2. This phase is run only if phase 1 found DUP blocks. The file system is rescanned for more DUP blocks, which are marked for clearing.

3. The directory structure of the file system is checked by descending the directory trees, examining each entry. A count is kept of the number of directory references for each file. If any entry refers to an unallocated file, a file marked for clearing, or a file outside the file system, then the entry is printed, and if the operator agrees, it is removed. Refusing to remove a file will clear the mark, preserving the file and its subsequent entries.

4. All marked or unreferenced files are listed. With operator concurrence, each of these files is then cleared. In addition, any file whose link count does not agree with the number of references is listed, and, if the operator agrees, the link count is adjusted.

5. This phase is skipped if the forced salvage, **−s**, option is on. Otherwise the free list is examined and if any blocks are found which are outside the file system, or which have been previously encountered in a file or the free list, then the list is pronounced BAD and a salvage is called for.

6. If forced salvaging is enabled through the **−s** option, or salvaging was called for in phase 5 and the operator agrees, then a new free list is constructed.

The file system should be dismounted when any repair work is done; if this is not possible (for example if the root file system has to be repaired) care should be taken that the system is quiescent and that it is *rebooted immediately afterwards* so that the old, bad in-core copy of the super-block will not continue to be used. If the file system name ends in a '.sys' and it is repaired during the course of the **fsck**, then a "REBOOT" message is given to remind the user of this.

FILES

/etc/checklist	default file system list

SEE ALSO

"VENIX Maintenance" in the *Installation and System Manager's Guide*
ncheck(1), checklist(4), filsys(4)

DIAGNOSTICS

Several self-explanatory messages, usually with a choice of continuing or exiting.

BUGS

Since **fsck** is multi-pass, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

NAME

graph — draw a graph

SYNOPSIS

graph [option] ...

DESCRIPTION

graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the **plot**(1) filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "...", in which case they may be empty or contain blanks and numbers; labels never contain newlines.

The following options are recognized, each as a separate argument.

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).

−b Break (disconnect) the graph after each label in the input.

−c Character string given by next argument is default label for each point.

−g Next argument is grid style: 0, no grid; 1, frame with ticks; 2, full grid (default).

−l Next argument is label for graph.

−m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

−s Save screen, don't erase before plotting.

−x [l]
If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) **x** limits. Third argument, if present, is grid spacing on **x** axis. Normally these quantities are determined automatically.

−y [l]
Similarly for **y**.

−h Next argument is fraction of space for height.

−w Similarly for width.

−r Next argument is fraction of space to move right before plotting.

−u Similarly to move up before plotting.

−t Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

EXAMPLES

Draw graph labeled "sample A" from coords in file "data.coords" and pipe to **plot** for drawing on Tektronix 4014 on device "/dev/tek"

```
graph −l "sample A" < data.coords | plot −T4014 > /dev/tek
```

SEE ALSO

spline(1), plot(1)

BUGS

graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

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

SYNOPSIS

grep [option] ... expression [file] ...

egrep [option] ... [expression] [file] ...

fgrep [option] ... [strings] [file]

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; unless the **−h** flag is used, the file name is shown if there is more than one input file.

grep patterns are limited regular expressions in the style of **ed**(1); it uses a compact nondeterministic 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. To determine which variation to use, match the complexity of the search expression and the speed in which it can be done.

The following options are recognized.

−v All lines but those matching are printed.

−c Only a count of matching lines is printed.

−l The names of files with matching lines are listed (once) separated by newlines.

−n Each line is preceded by its 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 No output is produced, only status.

−h Do not print filename headers with output lines.

−y Lower case letters in the pattern will also match upper case letters in the input (**grep** only).

−e *expression*
: Same as a simple *expression* argument, but useful when the *expression* begins with a −.

−f *file*
: The regular expression (**egrep**) or string list (**fgrep**) is taken from the *file*. This does not work for **grep.**

−x
: (Exact) only lines matched in their entirety are printed (**fgrep** only). Single quotes should be used for multiple word expressions.

Care should be taken when using the characters $ * [^ | ? ′ ″ ( ) and \ in the *expression* as they are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes ′ ′. For example:

```
grep −n ′/*′ *.c
```

prints out all comment lines in all C source files in the current directory.

fgrep searches for lines that contain one of the (newline-separated) *strings.*

egrep accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character matches that character.

The character ^ ($) matches the beginning (end) of a line.

A **.** matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the string. Ranges of ASCII character codes may be abbreviated as in 'a−z0−9'. A] may occur only as the first character of the string. A literal − must be placed where it can't be mistaken as a range indicator.

A regular expression followed by * (+, ?) matches a sequence of 0 or more (1 or more, 0 or 1) matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is [] then * + ? then concatenation then | and newline.

EXAMPLES

Search for all occurrences of the string ''xyz'' in file *prog.c*

```
grep xyz prog.c
```

Search the file *history.text* for the words given in file *spellerrs*.

```
fgrep –f spellerrs history.text
```

Count the number of lines with the words ''parrot'' or ''Parrot'':

```
grep –c '[Pp]arrot' prog.c
```

(note the use of single quotes to prevent the shell from interpreting the left and right brackets).

SEE ALSO

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

DIAGNOSTICS

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

BUGS

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

NAME

join — relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is '−', the standard input is used.

file1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

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

These options are recognized:

−**a***n*	In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
−**j***n m*	Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file.
−**o** *list*	Each output line comprises the fields specifed in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number.
−**t***c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

EXAMPLE

Two files, *offices* and *telephones,* can be joined to produce a table of names, offices and telephones.

```
offices:                 telephones:

Myron   rm 5             Myron   ext 7
Paul    rm 6             Paul    ext 6
Eliz    rm 1             Eliz    ext 9
Marc    rm 3             Marc    ext 4
John    rm 2             John    ext 5
```

First, sort *offices* and *telephones* by the name field, then do the join:

```
sort offices > temp1; sort telephones > temp2
join temp1 temp2
```

The output will be:

```
Eliz    rm 1    ext 9
John    rm 2    ext 5
Marc    rm 3    ext 4
Myron   rm 5    ext 7
Paul    rm 6    ext 6
```

SEE ALSO

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

BUGS

With default field separation, the collating sequence is that of **sort −b**; with **−t**, the sequence is that of a plain sort.

The conventions of **join, sort, comm**, **uniq**, **look** and **awk**(1) are wildly incongruous.

NAME

kill — terminate a process with extreme prejudice

SYNOPSIS

kill [−sig] processid ...

kill −l

DESCRIPTION

kill sends the TERM (terminate, 15) signal to the specified processes. If a signal name or number preceded by '−' is given as first argument, that signal is sent instead of terminate (see **signal**(2)). The signal names are listed by 'kill −l', and are as given in **/usr/include/signal.h** stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal; in particular 'kill −9 ...' is a sure kill. By convention, if process number 0 is specified, all members in the process group (i.e. processes resulting from the current login) are signaled.

The process number of an asynchronous process started with '&' is reported by the shell. Process numbers can also be found by using **ps**(1). Also, the last process started can be found in the Bourne shell **(sh)** variable, **$!**, or the C shell **(csh)** variable **$child**.

Normal users may only kill programs started by themselves. The super user may kill any program.

In rare cases, a killed process will not terminate because difficulties with a device prevent it from being closed.

EXAMPLES

Kill last process started:

```
kill $!
```

Kill process 3175:

```
kill 3175
```

Send "sure kill" to all background processes started by you:

```
kill −9 0
```

(Note that this will kill your shell and effectively log you out.)

DIAGNOSTICS

"No such process" if no process exists with that ID, or you are not its owner or the super-user.

SEE ALSO

ps(1), suspend(1), kill(2), signal(2)

BUGS

The message that the process was in fact killed will not be given until the shell does a wait; this happens when another command is executed.

NAME

ld — loader

SYNOPSIS

ld [option] file ...

DESCRIPTION

ld combines several object programs into one, resolves external references, and searches libraries. In most cases, it is recommended that the user not call **ld** directly, but instead from **cc** or **f77**(1), since they handle the proper loading sequences and will automatically pass appropriate flags to the loader.

In the simplest case several object *files* are given, and **ld** combines them, producing an object module which can be either executed or become the input for a further **ld** run. (In the latter case, the **−r** option must be given to preserve the relocation bits.) The output of **ld** is left on **a.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by **ranlib**(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named '__.SYMDEF', then it is understood to be a dictionary for the library such as produced by **ranlib**; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

ld understands several options. Except for **−l**, they should appear before the file names.

−s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by **strip**(1).

−u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−l*x* This option is an abbreviation for the library name **/lib/lib***x***.a**, where *x* is a string. If that does not exist, **ld** tries **/usr/lib/lib***x***.a**. A library is searched when its name is encountered, so the placement of a **−l** is significant. To correctly resolve all external references, a library must generally be loaded after the object file which makes the references.

−x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

−X Save local symbols except for those whose names begin with 'L'. This option is used by **cc**(1) to discard internally generated labels while retaining symbols local to routines.

−r Generate relocation bits in the output file so that it can be the subject of another **ld** run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

−d Force definition of common storage even if the **−r** flag is present.

−z# When the output file is executed, the stack will be placed below the data space. If no # is specified, then the stack will be 8kb (8192) in size; otherwise, the stack will be the **#** specified. This allows a more compact data segment, as normally a full 64kb is used. (RAINBOW/VENIX only).

−n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text. (VENIX/11 and PRO/VENIX only).

−i When the output file is executed, the program code and data areas will be placed in separate address spaces and the code will be shared among all users executing the file. On VENIX/11 and PRO/VENIX, this option can be used only on those processors with physically separate instruction and data space, such as the

11/44, or J-11. The only difference between this option and **−n** is that here the data starts at location 0.

−o	The *name* argument after **−o** is used as the name of the **ld** output file, instead of **a.out**.
−m	Program is to be "code-mapped," or overlayed in memory. This is only necessary for unusually large programs. The **−md** option causes the code-mapping table to be placed in the data rather than code segment of the program. See "Code-mapping Under VENIX" in the *Programming Guide*. (VENIX/11 and PRO/VENIX only).
−e	The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default, unless the **−z** flag is specified.
−T	The following argument specifies the hex base address for the text region. (RAINBOW/VENIX only).
−D	The following argument specifies the hex base address for the data region. (RAINBOW/VENIX only).
−B	The following argument specifies the hex base address for the bss region. (RAINBOW/VENIX only).

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
a.out	output file

SEE ALSO

ar(1), as(1), cc(1), nm(1)
ranlib(1), size(1), strip(1), lorder(1)

DIAGNOSTICS

Diagnostics are intended to be self-explanatory. The following diagnostic messages may be produced by the loader. Explanations are given where deemed necessary.

Bad 'use' or 'entry'	Missing argument following −u or −e flag
bad flag	
Bad format	Loader expects object modules or archives of object modules. This error can be produced if source files

given to **cc** don't end in '.c', as **cc** assumes they are object modules and passes them to the loader.

Bad output file

cannot create output

cannot create temp

cannot open

Entry point not in text
: Start-off point not located in text space

fast load buffer too small
: Not enough buffer space to load **ranlib**'ed library. Solution is to un-ranlib library by removing **__.SYMDEF** table-of-contents modules.

internal error: symbol not found

Local symbol botch
: Internal error. May be caused by use of older version of assembler with newer version of loader. Try recompiling sources and reloading.

Local symbol overflow
: Internal error

'−m' means relocation info is lost.

Mapping array size too big.
: Not enough room for code-mapping jump table of indicated size; decrease size specified with **−m** flag.

Mapping array too small.
: Not enough room in jump table for all entries; increase size with −m flag.

Multiply defined One symbol defined more than once.

No pages Internal buffer overrun.

No relocation bits
: Relocation information for module hase been stripped away, so can't relocate code.

No 'cmstart' Code-mapping start-off routine not present.

out of date (warning)
: Last modification date of library is later than when **ranlib**(1) was last run on it (this will occur even if the library is copied after **ranlib**'ed). Run **ranlib** on library again.

output error

premeof Premature end-of-file. Object file chopped off?

Relocation error Internal error

Symbol table overflow
: Too many symbols.

Text module too big.	Code-mapped segment greater than 8kb.
text overflow	Program too large (> 64kb); reduce code size, or code-map.
−v: arg missing	Argument must follow directly after −v

NAME

lex — generator of lexical analysis programs

SYNOPSIS

lex [−tvfn] [file] ...

DESCRIPTION

lex generates programs to be used in simple lexical analyis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, *lex.yy.c* is generated, to be compiled thus:

```
cc lex.yy.c −ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The following **lex** program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A−Z]   putchar(yytext[0]+'a'−'A');
[ ]+$
[ ]+    putchar(' ');
```

The options have the following meanings.

−t Place the result on the standard output instead of in file *lex.yy.c*.

−v Print a one-line summary of statistics of the generated analyzer.

−n Opposite of **−v**; **−n** is default.

−f 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

SEE ALSO

yacc(1)
"Lexical Analyzer Generator (LEX)" in the *Support Tools Guide*.

BUGS

Contrary to other documentation, **lex** swallows unrecognized input rather than pass it through.

NAME

lint — a C program verifier

SYNOPSIS

lint [**−abchnpuvx**] file ...

DESCRIPTION

lint attempts to detect features of the C program *files* which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to **lint**; these libraries are referred to by a conventional name, such as '−lm', in the style of **ld**(1).

Any number of the options in the following list may be used. The **−D**, **−U**, and **−I** options of **cc**(1) are also recognized as separate arguments.

−p Attempt to check portability to the *IBM* (mainframe) and *GCOS* dialects of C.

−h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

−b Report *break* statements that cannot be reached. (This is not the default because, unfortunately, most **lex** and many **yacc** outputs produce dozens of such comments).

−v Suppress complaints about unused arguments in functions.

−x Report variables referred to by extern declarations, but never used.

−a Report assignments of long values to int variables.

−c Complain about casts which have questionable portability.

−u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running **lint** on a subset of files out of a larger program).

−n Do not check compatibility against the standard library.

exit(2) and other functions which do not return are not understood; this causes various lies.

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

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

/*VARARGS*n**/
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*NOSTRICT*/
shuts off strict type checking in the next expression.

/*ARGSUSED*/
turns on the **−v** option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

FILES

/usr/lib/lint[12]	programs
/usr/lib/llib−lc	declarations for standard functions
/usr/lib/llib−port	declarations for portable functions

SEE ALSO

cc(1)
"A C Program Checker (lint)" in the *Programming Guide*.

NAME

ln — make a link

SYNOPSIS

ln name1 [name2]

DESCRIPTION

The **ln** command creates additional pathnames ("links") for the existing file *name1*. If *name2* is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of *name1*.

ln does not duplicate the file; it merely provides another name for the file. A file by any other (linked) name would smell as sweet ... because it is the same file. Any changes to the file are effective independently of the name by which it is known, and there is no way to distinguish the original file name from a "link" name that was added later. The size, protection mode, and contents of the file are identical no matter what name the file is accessed by.

Links are often made to create a convenient pathname for a frequently accessed file. Executable programs are sometimes linked to multiple names so that they can be called more conveniently. For example, the **ls**(1) command is named both **/bin/ls** and **/bin/l**; when executed, **ls** checks the name by which it was called, and sets the "long listing" option if that name was "l".

A link to a file can be removed with the **rm** command. This removes the given pathname to the file, but as long as there remains at least one pathname by which the file is known, the file itself is not removed.

It is forbidden to link to a directory or to link across file systems. For example, you cannot link a file in the user area to one in the system area.

A file may have up to 128 links. The number of links present to a given file may be determined with the **l** command, although there is no easy way to determine the other names to the file. (However, since all links to a file share a common inode, the output of the **ncheck**(1) command can be useful in tracing the additional link names).

EXAMPLES

Create a link to */usr/myron/prog.c* called *prog.c* in the current directory:

```
ln /usr/myron/prog.c
```

Link */usr/myron/prog* to */usr/paul/test/howdie*:

```
ln /usr/myron/prog /usr/paul/test/howdie
```

SEE ALSO

ls(1), rm(1), ncheck(1), link(2)

NAME

login — sign on

SYNOPSIS

login [username]

DESCRIPTION

The **login** command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See the *User Guide* for details on initially logging in.

If **login** is invoked without an argument, it asks for a user name, and, if appropriate, a password. Echoing is turned off during the typing of the password, so it will not appear on the terminal for others to see.

After a successful login, accounting files are updated and the user is informed of the existence of mail. The message-of-the-day is written out on the terminal. **login** initializes the user and group ID's and the working directory, then executes a command interpreter (usually **sh** or **csh**(1)), according to specifications found in a password file.

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/spool/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file

SEE ALSO

newgrp(1), mail(1), passwd(1), passwd(4), "Setting Up VENIX" in the *Installation and System Manager's Guide*.

DIAGNOSTICS

'Login incorrect,' if the name or the password is bad.
'No Shell' - can't run shell program given in /etc/passwd
'No Shell', 'cannot open password file', 'no directory': consult a programming counselor or the *Installation and System Manager's Guide*.

NAME

look — find lines in a sorted list

SYNOPSIS

look [**−df**] string [file]

DESCRIPTION

look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options **−d** and **−f** affect comparisons as in **sort**(1):

−d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

−f Fold. Upper case letters compare equal to lower case.

EXAMPLE

To find all lines in *report* which begin with "Section":

```
look −f Section report
```

FILES

/usr/dict/words (may not be installed on your system)

SEE ALSO

sort(1), grep(1)

NAME

lorder — find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see **ar**(1)) *files*. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by **tsort**(1) to find an ordering of a library suitable for one-pass access by **ld**(1).

This brash one-liner intends to build a new library from existing '.o' files:

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

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld(1), ar(1)

BUGS

The names of object files, in and out of libraries, must end with '.o'; nonsense results otherwise.

NAME

lpr — local printer spooler

lpstop — stop lpr spooling

SYNOPSIS

lpr [**−h** header] [**−b**] [**−n**] [**−z**] [files ...]

lpstop [**−apq**]

DESCRIPTION

lpr prints the contents of *file* (standard input default) on the local printing device. It makes a copy of the file, queues it with other pending output and returns to the user. The flags are:

−h Take the next argument as a header for the top of each page; otherwise, the file name is printed. (If the header is more than one word, or contains characters special to the shell, it should be quoted.) The date, user name and page numbers are also printed.

−b Print out a banner page before each output file.

−n Don't print out any headers, and don't print blank lines at the bottom of the page. This is useful when piping input to **lpr** which has already been paginated and otherwise formatted (for example by **pr** or **nroff**(1)).

−r Report to the user. Write a message to the terminal when printing is completed.

−z Used to tell **lpr** to start spooling files already queued in /tmp; this is used only when **lpr** is automatically called by a **plot** filter **(iplot** or **vplot)** (see **plot**(1)), or in the rare cases in which a user wishes to restart **lpr** after halting it other than by **lpstop**.

−c Don't map the standard line feed into the carriage return/line feed; implies **−n** option.

lpstop is used to stop **lpr** from spooling a file. It normally operates only on files owned by the user, and will stop the current file from spooling and remove any other of the user's files from the queue. The flags for **lpstop** are:

−a act on all files, not just those owned by the user (must be superuser to invoke).

−p only stop the current file from being printed; don't remove any files from the queue.

−q only remove the files from the queue; don't stop the current file from being printed.

EXAMPLES

Print the contents of all C source files in the current directory:

```
lpr *.c
```

List the contents of directory *sources* and pipe to **lpr**:

```
ls sources | lpr −h "directory sources"
```

Stop printing the current file, but don't remove other files from the printing queue:

```
lpstop −p
```

FILES

/tmp/lp????	temporaries
/tmp/id.lp	holds the lpr process id
/etc/passwd	to get user name
/dev/lp	output device

BUGS

Due to buffering of output by VENIX and printing hardware, **lpstop** may not be able to immediately stop the printing, or may not share your idea of what is the "current file." For the same reason, if the **−r** flag is used, the "lp finish" message may be slightly premature.

SEE ALSO

pr(1), cat(1)

NAME

l, ls — list contents of directory

SYNOPSIS

l [**−tasdrucifgmDFSBC**] name ...
ls [**−ltasdrucifgmDFSBC**] name ...

DESCRIPTION

For each directory argument, **ls** lists the contents of the directory; for each file argument, **ls** repeats its name and any other information requested. **l** is identical to **ls**, except that it always prints a long format listing (the **−l** option is default).

The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. There are several options:

−l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below). If the file is a special file the size field will instead contain the major and minor device numbers.

−t Sort by time modified (latest first) instead of by name, as is normal.

−a List all entries. In the absence of this option, entries whose names begin with '**.**' are not listed.

−s Give size in blocks, including indirect blocks, for each entry.

−d If argument is a directory, list only its name, not its contents (mostly used with **−l** to get status on 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 (**−t**) or printing (**−l**).

−c Use time of last modification to inode (mode, etc.) instead of last modification to file for sorting (**−t**) or printing (**−l**).

−i Print inode number in first column of the report for each file listed.

−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.

−g	Give group ID instead of owner ID in long listing.
−m	Suppress multi-column listing for short format, and instead list in single column.
−D	List only directories.
−F	List only plain files.
−S	List only special files (devices).
−B	List only block special files.
−C	List only character special files.

Listings with the **−l** option are of the form:

```
-rw-rw-r--    1    fred      4010   Aug 10  12:05   prog.c
drwxrwxrw-    3    fred        96   Aug  7  16:23   src
crw--w--w-    1    root       0,0   Sep 16  10:22   /dev/console
```

The first ten characters are interpreted as follows: the first character is

−	if the entry is a plain file;
d	if the entry is a directory;
b	if the entry is a block-type special file;
c	if the entry is a character-type special file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. 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 bit of the mode is on. See **chmod**(2) for the meaning of this mode.

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

The link count is given after the mode. For files, this will be 1 unless extra links have been made to the file. For directories, this will be at least 2 (since each directory has a '.' which is linked to it, in addition to its link to the parent directory), and will be more than two if the directory has subdirectories in it. See **ln**(1).

The user name (group name with **−g** option) is given next; the ID number is given if no name can be found. For plain files and directories, the next field contains the length of the file (in bytes); for device nodes, the major and minor device numbers are given.

The date the file was last modified (last accessed with **−u** option) is given next. The time of day is given if the date is in the last six months; otherwise, the year is given. The file name is given at the end.

EXAMPLES

List the contents of the current directory and directory */u0/data*:

```
ls . /u0/data
```

List all directories in the current directory:

```
ls −D
```

Give a long listing of the entry for directory *data* itself (not its contents):

```
l −d data
```

FILES

/etc/passwd	to get user ID's for long listings
/etc/group	to get group ID's for **ls −g**

NAME

m4 — macro processor

SYNOPSIS

m4 [files]

DESCRIPTION

m4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no arguments, or if an argument is '–', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

```
name(arg1,arg2, . . . , argn)
```

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string.

undefine removes the definition of the macro named in its argument.

ifdef If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word **unix** is predefined on UNIX and VENIX versions of **m4**.

changequote
Change quote characters to the first and second arguments. **changequote** without arguments restores the original values (i.e., left and right single quotes).

divert **m4** maintains 10 output streams, numbered 0 – 9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The **divert** macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum returns the value of the current output stream.

dnl reads and discards characters up to and including the next newline.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, –, *, /, %, ^ (exponentiation); relationals; parentheses.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or –1 if the second argument does not occur.

substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to **include**, except that it says nothing if the file is inaccessible.
syscmd	executes the VENIX command given in the first argument. No value is returned.
maketemp	fills in a string of XXXXX in its argument with the current process ID.
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.

SEE ALSO

"The M4 Macro Processor" in the *Support Tools Guide*.

NAME

mail — send or receive mail among users

SYNOPSIS

mail person ...
mail [**−r**] [**−q**] [**−p**] [**−f** file]

DESCRIPTION

mail with no argument prints a user's mail, message-by-message, in last-in, first-out order; the optional argument **−r** causes first-in, first-out order. If the **−p** flag is given, the mail is printed with no questions asked; otherwise, for each message, **mail** reads a line from the standard input to direct disposition of the message.

newline Go on to next message.

d Delete message and go on to the next.

p Print message again.

− Go back to previous message.

s [*file*] ...
Save the message in the named *files* ('mbox' default).

w [*file*] ...
Save the message, without a header, in the named *files* ('mbox' default).

m [*person*] ...
Mail the message to the named *persons* (yourself is default).

EOT (control − D)
Put unexamined mail back in the mailbox and stop.

q Same as EOT.

x Exit, without changing the mailbox file.

!*command*
Escape to the Shell to do *command*.

? Print a command summary.

An interrupt stops the printing of the current letter. The optional argument **−q** causes **mail** to exit after interrupts without changing the mailbox.

When *persons* are named, **mail** takes the standard input up to an end-of-file (or a line with just '.') and adds it to each *person*'s 'mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *person* is usually a user name recognized by **login**(1). On installations with **uucp** communications software, a recipient on a remote system can be specified by prefixing *person* by the system name and exclamation mark (see **uccp**(1)).

The **−f** option causes the named file, e.g. 'mbox', to be printed as if it were the mail file.

Each user owns his own mailbox, which is by default generally readable but not writable. The command does not delete an empty mailbox nor change its mode, so a user may make it unreadable if desired.

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

FILES

/usr/spool/mail/*	mailboxes
/etc/passwd	to identify sender and locate persons
mbox	saved mail
/tmp/ma*	temp file
dead.letter	unmailable text

EXAMPLES

Mail the contents of file *xletter* to *fred* and *joe*:

```
mail fred joe < xletter
```

Send a letter to *eve*; the letter's contents are read from the terminal:

```
mail eve
Madam, I'm Adam.
^D
```

SEE ALSO

write(1)

BUGS

There is a locking mechanism intended to prevent two senders from accessing the same mailbox, but it is not perfect and deadly embraces are possible.

NAME

make — maintain program groups

SYNOPSIS

make [−**f** makefile] [option] ... name ...

DESCRIPTION

make executes commands in *makefile* to update one or more target *names*. **name** is typically a program. If no −**f** option is present, *makefile* and *Makefile* are tried in order. If *makefile* is '−', the standard input is taken. More than one −**f** option may appear.

make updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target.

Sharp and newline surround comments.

The following makefile says that 'pgm' depends on two files 'a.o' and 'b.o', and that they in turn depend on '.c' files and a common file 'incl'.

```
pgm: a.o b.o
        cc a.o b.o −lm −o pgm
a.o: incl a.c
        cc −c a.c
b.o: incl b.c
        cc −c b.c
```

makefile entries of the form

```
string1 = string2
```

are macro definitions. Subsequent appearances of *$(string1)* are replaced by *string2*. If *string1* is a single character, the parentheses are optional.

make infers prerequisites for files for which *makefile* gives no construction commands. For example, a '.c' file may be inferred as prerequisite for a '.o' file and be compiled to produce the '.o' file. Thus the preceding example can be done more briefly:

```
        pgm: a.o b.o
                cc a.o b.o −lm −o pgm
        a.o b.o: incl
```

Prerequisites are inferred according to selected suffixes listed as the 'prerequisites' for the special name '.SUFFIXES'; multiple lists accumulate; an empty list clears what came before. Order is significant; the first possible name for which both a file and a rule as described in the next paragraph exist is inferred. The default list is

```
        .SUFFIXES: .out .o .c .e .r .f .y .l .s
```

The rule to create a file with suffix *s2* that depends on a similarly named file with suffix *s1* is specified as an entry for the 'target' *s1s2*. In such an entry, the special macro $* stands for the target name with suffix deleted, $@ for the full target name, $< for the complete list of prerequisites, and $? for the list of prerequisites that are out of date. For example, a rule for making optimized '.o' files from '.c' files is

```
        .c.o: ; cc −c −O −o $@ $*.c
```

Certain macros are used by the default inference rules to communicate optional arguments to any resulting compilations. In particular, 'CFLAGS' is used for **cc** options, 'LFLAGS' and 'YFLAGS' for **lex** and **yacc**(1) options.

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target '.SILENT' is in *makefile*, or the first character of the command is '@'.

Commands returning nonzero status (see **intro(1)**) cause **make** to terminate unless the special target '.IGNORE' is in *makefile* or the command begins with <tab><hyphen>.

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

Other options:

−i Equivalent to the special entry '.IGNORE:'.

−k	When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.
−n	Trace and print, but do not execute the commands needed to update the targets.
−t	Touch, i.e. update the modified date of targets, without executing any commands.
−r	Equivalent to an initial special entry '.SUFFIXES:' with no list.
−s	Equivalent to the special entry '.SILENT:'.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1)

BUGS

Some commands return nonzero status inappropriately. Use **−i** to overcome the difficulty.
Commands that are directly executed by the shell, notably **cd**(1), are ineffectual across newlines in **make**.

NAME

mesg — permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

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

FILES

/dev/tty*
/dev

SEE ALSO

write(1)

DIAGNOSTICS

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

NAME

mkdir — make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

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

mkdir requires write permission in the parent directory.

SEE ALSO

rm(1)

DIAGNOSTICS

mkdir returns the prompt if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

NAME

mkfs — construct a file system

SYNOPSIS

/etc/mkfs [**−b**] device [size | proto] [m n]

DESCRIPTION

mkfs constructs a file system structure on a device which is either a disk partition or a diskette. This must be done before the partition or diskette is mounted and used as a directory tree to store and manipulate files. Making a file system destroys any data previously on the device.

The **−b** flag causes **mkfs** to check the disk for bad blocks (unreadable), to gather them together and put them in an inaccessible file.

Most commonly, the second argument to **mkfs** is a number, which is used as the size of the device in blocks. A file system is constructed of that size with a single directory in it.

The optional parameters *m* and *n* specify respectively, the block interleave factor and number of sectors per track. (RAINBOW/VENIX only).

Otherwise, if the second argument is some file name *proto*, that file is used as a prototype description of the file system. This allows the creation of a file system with preconstructed files and directories in it.

The prototype description file consists of tokens separated by spaces or new lines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of blocks to be used for i-nodes. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

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

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

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, **mkfs** makes the entries '.' and '..' and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

If the prototype file cannot be opened and its name consists of a string of digits, **mkfs** builds a file system with a single empty directory on it. The size of the file system is the value of *proto* interpreted as a decimal number. The number of i-nodes is calculated as a function of the filsystem size. The boot program is left uninitialized.

If a file system size is specified rather than a prototype description, then the boot block is left uninitialized. The number of blocks used for i-nodes is calculated as a function of the file system size according to the equation:

```
iblocks = (size/2) / (42 + (size/1000))
```

There are 16 i-nodes per block. If this number is not suitable, you can specify a different number of blocks with a prototype file, as described above.

EXAMPLES

Create a file system of length *640* on */dev/f0*:

```
/etc/mkfs /dev/f0 640
```

A sample prototype specification:

```
/usr/boots/flboot
640 15
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

SEE ALSO

filsys(4), dir(4)

BUGS

There should be some way to specify links.

NAME

mknod — build special file (device node)

SYNOPSIS

/etc/mknod name [**b**] [**c**] major minor

DESCRIPTION

mknod makes a special file (device node). This must be done when adding a new device to the system, or creating a new entry to an existing device.

The first argument is the *name* of the entry. The second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (terminals and other devices, as well as raw versions of disk and tape). The last two arguments are numbers specifying the *major* device type (e.g. controller) and the *minor* device (e.g. unit, drive, or line number).

Special files usually reside in directory /**dev**. It is entirely permissible to have two or more nodes with identical numbers; this is sometimes done if a program prefers to call a device by some particular name. The numbering of existing device nodes can be seen with a long listing of directory /**dev** by **ls**(1).

mknod may be used only by the super-user. After creating a node, you may wish to change its permission mode (with **chmod**(2)) to restrict or expand other users' access to that device.

The major numbers are used by VENIX as an index into either the block or characters tables within the system source file **c.c**. The minor device number is particular to each driver; typically, it selects a channel on a particular controller. For hard disks, each minor device number corresponds to a different disk partition, specifically the index of the partition's entry in the partition table. The table is documented in the appropriate disk write-up in the device driver documentation. On some devices, a particular bit is or'ed into the minor device number to set some condition (e.g. to indicate different interleaving on floppy diskettes). See the appropriate device write-up.

All ''raw'' disk and tape devices can also be accessed ''asynchronously'', for high-speed data throughput. These asynchronous versions are conventionally named with an ''a'' instead of an extra ''r'', as in ''arl2''. These asynchronous versions differ from standard raw versions in their minor device numbers, which are the standard numbers with octal 0200 (decimal 128) added to them.

EXAMPLES

Make nodes for f0 and f1:

```
/etc/mknod /dev/f0 b 0 0
/etc/mknod /dev/f1 b 0 1
```

SEE ALSO

mknod(2)

NAME

more — file perusal filter for crt viewing

SYNOPSIS

more [**−d**] [**−f**] [**−l**] [−*n*] [+*linenumber*] [+*/pattern*] [name ...]

page [**−d**] [**−f**] [**−l**] [−*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. Other possibilites are enumerated later.

The command line options are:

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

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

−f This causes **more** to count logical, rather than screen lines. That is, long lines are not folded. This option is useful when reading input containing escape sequences with characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of such an escape sequence. Without the **−f** flag, **more** may think that lines containing these non-printing characters are longer than they actually are, and fold lines erroneously.

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

+*linenumber*
Start up at *linenumber*.

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

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

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

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

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

i<space>	Display *i* more lines, (or another screenful if no argument is given).
^D	(control – D) Display 11 more lines (a 'scroll'). If *i* is given, then the scroll size is set to *i*.
d	Same as ^D.
*i*z	Same as typing a space except that *i*, if present, becomes the new window size.
*i*s	Skip *i* lines and print a screenful of lines.
*i*f	Skip *i* screenfuls and print a screenful of lines.
q or Q	Exit from **more**.
=	Display the current line number.
v	Start up the editor **vi**(1) at the current line.
h	Help command; give a description of all the **more** commands.
i/expr	Search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains

unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

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

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

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

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

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

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

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

.
: (dot) Repeat the previous command.

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

At any time when output is being sent to the terminal, the user can hit the quit key (normally control – Z). **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.

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 terminal, then **more** acts just like **cat**(1), except that a header is printed before each file (if there is more than one).

EXAMPLES

Previewing **nroff**(1) output:

```
nroff –ms +2 doc.n | more
```

FILES

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

NAME

mount, umount — mount and dismount file system

SYNOPSIS

mount [dev name [−**r**]]

umount dev

DESCRIPTION

mount announces to the system that a removable file system is present on the block device *dev*, which must have a file-system structure on it (as prepared by **mkfs**(1)). *name* is normally the name of a directory which must exist already. *name* can only be a file in the unusual case that the root of the mounted file system is not a directory. It becomes the name of the newly mounted root.

The optional −**r** argument indicates that the file system is to be mounted read-only. In this case, no writing may be done to any files in the system.

umount instructs the system that the removable file system mounted on the hard or floppy disk is to be removed.

These commands maintain a table of mounted devices. If invoked without an argument, **mount** prints the table of all existing mounted devices.

Physically write-protected file systems must be mounted read-only (−**r**) or errors will occur when access times are updated, whether or not any explicit write is attempted.

mounts will fail if: the device is already mounted (often because someone forgot to **umount** the last floppy that was removed); it does not have a valid file system on it, as made by **mkfs**(1); it is not a block device; the specified directory is being used by somebody, or is somebody's current directory; or there are too many devices already mounted (only possible if the system has been configured incorrectly). **umount**s will fail if any file on the mounted device is still open, or if anyone is still sitting in a directory on the mounted file system.

EXAMPLES

Mount the removable file system present on the device */dev/f0* under the directory */f0*:

```
        mount /dev/f0 /f0
```

Remove the same file system from under */f0*. First, make sure you are off the file system.

```
        cd /
        umount /dev/f0
```

FILES

/etc/mtab:	mount table

SEE ALSO

mount(2), mtab(4)

BUGS

Mounting file systems full of garbage may crash the system.
Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

mv — move or rename files and directories

SYNOPSIS

mv file1 file2

mv file ... directory

DESCRIPTION

mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, **mv** prints the mode (see **chmod**(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place; if not, **mv** exits.

In the second form, one or more *files* are moved to the *directory* with their original filenames.

mv refuses to move a file onto itself.

SEE ALSO

cp(1), chmod(2)

EXAMPLES

Give the file *prog1* in the current directory the new name *prog2*:

```
mv prog1 prog2
```

Move the files *chap1*, *chap2*, and *chap3* in the current directory to the directory */u0/report*:

```
mv chap1 chap2 chap3 /u0/report
```

BUGS

If *file1* and *file2* 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.

mv should take **−f** flag, like **rm**, to suppress the question if the target exists and is not writable.

NAME

ncheck — generate names from i-numbers

SYNOPSIS

ncheck [−**i** numbers] [−**a**] [filesystem]

DESCRIPTION

ncheck with no arguments generates a pathname vs. inode number list of all filesystems in the default list in **/etc/checklist**. The −**i** option reduces the report to only those files whose i-numbers follow. The −**a** option allows printing of the names '.' and '..', which are ordinarily suppressed.

A filesystem may be specified.

The report is in no useful order, and probably should be sorted.

EXAMPLES

To list and sort all the files in the user area:

```
ncheck /dev/rw0.usr | sort −n
```

FILES

/etc/checklist	default filesystem list

SEE ALSO

fsck(1), sort(1)

NAME

neqn, checkeq — typeset mathematics

SYNOPSIS

neqn [**−d**xy] [**−s**n] [**−f**n] [file] ...
checkeq [file] ...

DESCRIPTION

neqn is an **nroff**(1) preprocessor. (For a tutorial on **neqn,** see ''Mathematics Typesetting Program'' in the *Document Processing Guide*.) Usage is almost always

```
neqn file ... | nroff
```

If no files are specified, **neqn** reads from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as **neqn** input. Delimiters may be set to characters *x* and *y* with the command-line argument **−d***xy* or (more commonly) with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program **checkeq** reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within **neqn** are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup.** Thus *x* **sub** *i* makes x_i, *a* **sub** *i* **sup** *2* produces a_i^2, and *e* **sup** {*x* **sup** *2* + *y* **sup** gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a* **over** *b* yields $\frac{a}{b}$.

sqrt makes square roots: *1* **over sqrt** {*ax* **sup** *2* +*bx*+*c*} results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n \to \infty} \sum_0^n x_i$ is made with **lim from** {*n*−> *inf* } **sum from** *0* **to** *n x* **sub** *i*.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: **left [** *x* **sup** *2* + *y* **sup** *2* **over** *alpha* **right]** ˜=˜*1* produces $\left[x^2+\frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: **pile** {*a* **above** *b* **above** *c*} produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: **matrix** { **lcol** { *x* **sub** *i* **above** *y* **sub** *2* } **ccol** { *1* **above** *2* } } produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x* **dot** = *f(t)* **bar** is $\dot{x} = \overline{f(t)}$, *y* **dotdot bar** ˜=˜ *n* **under** is $\bar{\ddot{y}} = \underline{n}$, and *x* **vec** ˜=˜*y* **dyad** is $\vec{x} = \overleftrightarrow{y}$.

Fonts can be changed with **roman**, **italic**, **bold**, and **font** *n*. Fonts can be changed globally in a document by **gfont** *n*, or by the command-line arguments −s*n* and −**f***n*. Note that **nroff** will try to underline anything in italic fonts, and send bold-on/bold-off codes (if appropriate) to get typewriter hardware to do bold font.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined, or existing keywords redefined, with **define**: **define** *thing* % *replacement* % defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords like **sum** ($\sum$) **int** ($\int$) **inf** (∞) and shorthands like > = ($\geq$) and ! = ($\neq$) are recognized. Greek letters are spelled out in the desired case, as in **alpha** or **GAMMA**. Mathematical words like sin, cos, log are made Roman automatically. **nroff**(1) four-character escapes like \(dd (‡) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with **nroff** when all else fails.

EXAMPLE

To process equations inside the file *report*, and print out the result:

```
neqn report | nroff -T450 | lpr
```

Note that **nroff** must be used with the **−T** option.

SEE ALSO

nroff(1), tbl(1)
"Mathematics Typesetting Program" and the "Nroff User's Manual" in the *Document Processing Guide.*

BUGS

neqn does its best, but depending on the hardware, certain constructions may not be entirely aesthetically pleasing.

To embolden digits, parens, etc., it is necessary to quote them, as in 'bold "12.3"'.

SPECIAL CHARACTER DEFINITIONS

The following **nroff** character definitions, found in file **/usr/pub/eqnchar**, are for constructing characters that are not available on printers. These definitions are primarily intended for use with **neqn**. The command usage is

neqn /usr/pub/eqnchar [files] | **nroff** [options]

eqnchar contains definitions for the following characters:

ciplus	⊕	| |	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	■
-wig	~ –	*hbar*	‾ *h*	*bullet*	•
> *wig*	≳	*ppd*	⊥	*prop*	∝
< *wig*	≲	<->	↔	*empty*	∅
= *wig*	≅	< = >	⇔	*member*	
star	*	|	≮	*nomem*	/
bigstar	✱	| >	≯	*cup*	∪
= *dot*	≐	*ang*	∠	*cap*	∩
orsign	\\/	*rang*	∟	*incl*	⊑
andsign	/\\	*3dot*	⋮	*subset*	⊂
= *del*	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°		

NAME

newgrp — log in to a new group

SYNOPSIS

newgrp group

DESCRIPTION

newgrp changes the group identification of its caller, analogously to **login**(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

A password is demanded if the group has a password and the user himself does not.

When most users log in, they are members of the group named 'other'. **newgrp** is known to the shell, which executes it directly without a fork.

FILES

/etc/group, /etc/passwd

SEE ALSO

login(1), group(4)

BUGS

Group passwords are not fully implemented; while there is a field in **/etc/group** for an encrypted password, there is no **passwd**-like command to set it.

NAME

nice, nohup — run a command at low priority

SYNOPSIS

nice [−number] command [arguments]

nohup command [arguments]

DESCRIPTION

nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. '− −10'. If the priority is −100 or less, then the process becomes 'real-time' and receives the absolute maximum precedence (usually at great expense to other processes); see **nice**(2).

nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority of the command is incremented by 5. In other words, it is a way to make sure that CTRL−C's and −Z's do not interfere with the execution of the command. For this reason, the output of the command is sent to the file **nohup.out** in the current directory, rather than to the terminal. **nohup** should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal.

When running the C shell **(csh)**, **nice** and **nohup** are built-in commands and take different arguments. See **csh**(1) for details.

EXAMPLES

Compile a program at low priority in the background:

```
nice −20 cc prog.c &
```

FILES

nohup.out	standard output and standard error file under **nohup**

SEE ALSO

nice(2)

DIAGNOSTICS

nice returns the exit status of the subject command.

NAME

nm — print name list

SYNOPSIS

nm [−gnopru] [file ...]

DESCRIPTION

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

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

Options are:

−g	Print only global (external) symbols.
−n	Sort numerically rather than alphabetically.
−o	Prepend file or archive element name to each output line rather than only once.
−p	Don't sort; print in symbol-table order.
−r	Sort in reverse order.
−u	Print only undefined symbols.

EXAMPLES

Print symbol table for *prog* in numerical order:

```
nm prog
```

Find all symbols in *prog* that contain the string "don":

```
nm prog | grep don
```

SEE ALSO

ar(1), ar(4), a.out(4), size(1), strip(1)

NAME

nroff — text formatting

SYNOPSIS

nroff [option] ... [file] ...

DESCRIPTION

nroff formats text in the named *files* for printing on typewriter-like devices. Its capabilities are described in the "Nroff User's Manual," in the *Document Processing Guide*.

If no *file* argument is present, the standard input is used. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input, and is often used to attach special macro files to a text file for processing. The options, which may appear in any order so long as they appear before the files, are:

−**o***list*	Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. For example, −**o2,6** prints only the second and sixth pages. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end.
−**n***N*	Number the first generated page *N*.
−**s***N*	Stop every *N* pages. **nroff** will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.
−**m***name*	Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*. This is a convenient way to specify different page formatting macro's from the command line.
−**r***aN*	Set register *a* (one-character) to *N*.
−**i**	Read standard input after the input files are exhausted.
−**q**	Invoke the simultaneous input-output mode of the **rd** request, as described in the "Nroff User's Manual."
−**T***name*	Prepare output for specified terminal. Known *names* are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), and **450** for the DASI-450 (Diablo Hyterm).
−**e**	Produce equally-spaced words in adjusted lines, using full terminal resolution.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

EXAMPLES

Run off document "doc.1" with −ms macros, formatted for Diablo Hyterm and send it to the terminal screen:

```
nroff −T450 −ms doc.1
```

Run off document *proposal*, with macro set *myron* and send it to the printer spooler with a no-header flag:

```
nroff −mmyron −T450 proposal | lpr −c
```

FILES

666/usr/lib/suftabsuffix hyphenation tables
/tmp/ta*temporary file
/usr/lib/tmac/tmac.*standard macro files
/usr/lib/term/*terminal driving tables

SEE ALSO

Document Processing Guide
neqn(1), tbl(1)
col(1)

MS MACRO SET

This package of **nroff** macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through **col**(1).

To use this set, the **−ms** flag must be given on the **nroff** command line. See "Using the −ms Macros" in the *Document Processing Guide* for details.

The macro requests are defined below. Many **nroff** requests are unsafe in conjunction with this package, however these requests may be used with impunity after the first .PP:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n = 1 single, n = 2 double space
.na	no alignment of right margin

Output of the **neqn** and **tbl**(1) preprocessors for equations and tables is acceptable as input.

The following requests are available:

REQUEST	INITIAL VALUE	CAUSE BREAK	EXPLANATION
.1C	yes	yes	One column format on a new page.
.2C	no	yes	Two column format.
.AB	no	yes	Begin abstract.
.AE	-	yes	End abstract.
.AI	no	yes	Author's institution follows. Suppressed in TM.
.AT	no	yes	Print 'Attached' and turn off line filling.
.AU *x y*	no	yes	Author's name follows. *x* is location and *y* is extension, ignored except in TM.
.B *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.B1	no	yes	Begin text to be enclosed in a box.
.B2	no	yes	End text to be boxed & print it.
.BT	date	no	Bottom title, automatically invoked at foot of page. May be redefined.
.BX *x*	no	no	Print *x* in a box.
.CS *x*...	-	yes	Cover sheet info if TM format, suppressed otherwise. Arguments are number of text pages, other pages, total pages, figures, tables, references.
.CT	no	yes	Print 'Copies to' and enter no-fill mode.
.DA *x*	nroff	no	'Date line' at bottom of page is *x*. Default is today.
.DE	-	yes	End displayed text. Implies .KE.
.DS *x*	no	yes	Start of displayed text, to appear verbatim line-by-line. *x* =I for indented display (default), =L for left-justified on the page, =C for centered, =B for make left-justified block, then center whole block. Implies .KS.

.EG	no	-	Print document in BTL format for 'Engineer's Notes.' Must be first.
.EN	-	yes	Space after equation produced by **neqn**.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.FE	-	yes	End footnote.
.FS	no	no	Start footnote. The note will be moved to the bottom of the page.
.I *x*	no	no	Italicize (underline) *x*; if *x* missing, italic text follows.
.IM	no	no	Print document in BTL format for an internal memorandum. Must be first.
.IP *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.KE	-	yes	End keep. Put kept text on next page if not enough room.
.KF	no	yes	Start floating keep. If the kept text must be moved to the next page, float later text back to this page.
.KS	no	yes	Start keeping following text.
.LG	no	no	Make letters larger.
.LP	yes	yes	Start left-blocked paragraph.
.MF	-	-	Print document in BTL format for 'Memorandum for File.' Must be first.
.MR	-	-	Print document in BTL format for 'Memorandum for Record.' Must be first.
.ND	date	no	Use date supplied (if any) only in special BTL format positions; omit from page footer.
.NH **n**	-	yes	Same as .SH, with section number supplied automatically. Numbers are multilevel, like 1.2.3, where *n* tells what level is wanted (default is 1).

.NL	yes	no	Make letters normal size.
.OK	-	yes	'Other keywords' for TM cover sheet follow.
.PP	no	yes	Begin paragraph. First line indented.
.PT	pg #	-	Page title, automatically invoked at top of page. May be redefined.
.QE	-	yes	End quoted (indented and shorter) material.
.QP	-	yes	Begin single paragraph which is indented and shorter.
.QS	-	yes	Begin quoted (indented and shorter) material.
.R	yes	no	Roman text follows.
.RE	-	yes	End relative indent level.
.RP	no	-	Cover sheet and first page for released paper. Must precede other requests.
.RS	-	yes	Start level of relative indentation. Following .IP's are measured from current indentation.
.SG *x*	no	yes	Insert signature(s) of author(s), ignored except in TM. *x* is the reference line (initials of author and typist).
.SH	-	yes	Section head follows, font automatically bold.
.SM	no	no	Make letters smaller.
.TA *x*...	5...	no	Set tabs in ens. Default is 5 10 15 ...
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TL	no	yes	Title follows.
.TM *x*...	no	-	Print document in BTL technical memorandum format. Arguments are TM number, (quoted list of) case number(s), and file number. Must precede other requests.
.TR *x*	-	-	Print in BTL technical report format; report number is *x*. Must be first.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.UL *x*	-	no	Underline argument.

.UX	-	no	'UNIX'; first time used, adds footnote 'UNIX is a trademark of Bell Laboratories.'

NAME

od — octal dump

SYNOPSIS

od [**−bcdox**] [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 argument characters are:

b	Interpret bytes in octal.
c	Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null = \0, backspace = \b, formfeed = \f, newline = \n, return = \r, tab = \t; others appear as 3-digit octal numbers.
d	Interpret words in decimal.
o	Interpret words in octal.
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 a '+'.

Dumping continues until end-of-file, unless you hit CTRL − C to exit.

EXAMPLES

Do an octal dump of file ''xyz'' starting from block 10, interpreting bytes in ASCII:

```
od −c xyz 10.b
```

Do an octal dump of device ''/dev/w0.sys'':

```
od /dev/w0.sys
```

SEE ALSO

adb(1)

NAME

passwd — change login password

SYNOPSIS

passwd [name]

DESCRIPTION

passwd changes (or installs) a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough and repeat your choice several times. To remove a password, type only a carriage return.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(4), crypt(3)
Robert Morris and Ken Thompson, *Password Security: A Case History*

NAME

pc — Pascal compiler

SYNOPSIS

pc arguments file ...

DESCRIPTION

This product is based on the Pascal system developed by Andrew S. Tanenbaum, Johan W. Stevenson and Hans van Staveren of the Vrije Universiteit, Amsterdam, The Netherlands.

This program enables you to compile Pascal programs. It offers two alternatives to run these programs: interpretation or compilation to true PDP-11 code. With interpretation, your program is compiled into object code for a virtual machine, called EM-1, which is simulated by an interpreter **em1**(1). This way of compilation is fast and gives many runtime checks.

If your program is debugged and if fast execution is needed, you can compile your program to PDP-11 machine code for direct execution. Compilation time will be longer, memory requirements will be larger and runtime checks are off, but execution is about seven times faster. Keep these two alternatives in mind while reading.

pc accepts the following flags:

−C Compiled code. Produce a compiled **a.out** file for fast execution. An **e.out** file for interpretation is default.

−E Produce a complete listing of the Pascal source program. Normally for each error, one message, including the source line number, is given.

−e List erroneous lines only.

−w Suppress warning messages.

−d If interpreting, collect debugging information. See **em1**(1) for what this accomplishes. This information can be looked at with **edebug**(1).

−c Stop when an intermediate EM-1 object module is made. These modules may at a later time be fully compiled or linked with other modules.

−f If your machine does not have floating point hardware and if you use reals in your program and if you want an executable version (**−C**), then you must give the **−f** flag to load the floating point package.

−o Take the next argument as a name of the object code file, instead of the default **e.out** or **a.out**.

−l*xxx* Include library **/usr/lib/libxxx.a** (if compiling) or **/usr/lib/em1_xxx.a** (if interpreting). Used to include "libpc" (**−l***pc*) or "libmon" (**−l***mon*); see **libpc**(3) and **libmon**(2).

−L Produce a library module (".l") file as output. These modules may be archived with the VENIX archiver **ar**. Main modules should not be placed in this format.

−r Try to run the object code file. No arguments can be passed to your program this way, so it is only useful in simple cases.

−ss, **−sm**, **−sl**
Indicate that your program is small, medium or large. This causes some of the programs to use more or less memory for tables. The default is **−sm**. On machines without separate I and D spaces, the **−sl** flag may cause errors.

−O Optimize. Optimizing is default when compiling is done (**−C** flag).

−L Produce library modules of intermediate code. These modules can be linked faster than normal intermediate modules, and may be archived. Library modules are always optimized. Only separate Pascal procedures should be made into library modules, never main programs.

−I On machines without separate I and D space the maximum size Pascal program that can be compiled is about 1000 lines. To compile larger programs the **−I** flag can be given to run a (slow) interpreted version of the compiler.

−S This flag is only effective with **−C**. It causes translation of compact EM-1 code into a VENIX load module to stop when the VENIX assembly code has been produced.

−P Run only the preprocessor for Pascal source files starting with a '#'. Leave the result in a file with suffix ".i".

−Dname = def
−Dname
Define the **name** to the preprocessor, as if by '#define'. If no definition is given, the name is defined as 1.

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

All flags must be provided as separate arguments. If compiling executable code, unknown flags are passed to the loader.

pc recognizes the following extensions in file names:

.p	Pascal source program.
.k	Unoptimized EM-1 intermediate files.
.m	Optimized EM-1 intermediate files.
.l	Library modules. These may be archived with the standard VENIX archiver **ar**(1).
.i	Preprocessed Pascal sources as produced by the C preprocessor **/lib/cpp**.

All arguments without a suffix or with an unrecognized suffix are also passed to the loaders, as are flags.

FILES

/bin/as	assembler
/bin/ld	loader
/usr/bin/em1	EM-1 interpreter
/lib/pc/pc_pem	Pascal compiler
/lib/pc/pc_pem.out	idem, interpreted
/lib/pc/pc_opt	optimizer
/lib/pc/pc_pdp	translate to PDP-11 code
/lib/pc/pc_ass	EM-1 assembler and loader
/lib/pc/pc_makelib	make library module
/lib/pc/pc_rt0.o	executable run time startoff
/lib/pc/pc_frt0.o	executable run time startoff if you want to use the software floating point package
/lib/pc/pc_bss.o	end of bss segment
/lib/pc_emlib.a	executable EM-1 run time library
/lib/pc_prlib.a	executable Pascal run time library
/lib/em1_pr.a	interpretable Pascal run time library

−lpc	library with external Pascal routines
/tmp	used for temporary files
/lib/cpp	C preprocessor

SEE ALSO

em1(1)

[1] K.Jensen & N.Wirth *PASCAL, User Manual and Report*, Springer-Verlag.

[2] J.W.Stevenson *VU−Pascal Reference Manual*.

[3] The latest version of ISO standard proposal ISO/TC97/SC5−N462, November 1979.

[4]

em1(1)	EM-1 interpreter
pc_prlib(3)	Pascal run time routines
libpc(3)	library of external routines
libmon(2)	library of system call routines

DIAGNOSTICS

The diagnostics are intended to be self-explanatory.

AUTHOR

Johan Stevenson, Vrije Universiteit, Amsterdam.

NAME

pie — draw a pie chart

SYNOPSIS

pie [option] ...

DESCRIPTION

pie with no options takes a series of <label number> pairs from the standard input as the label and value of pie wedges for a pie chart. The delimiter between fields in the input is a tab. The chart is encoded on the standard output for display by **plot**(1) filters.

Wedge values are summed to find the total amount the pie represents. Both wedge value and percentage can be optionally displayed along with the wedge label.

The following options are recognized, each as a separate argument.

−c **color**. Specify colors by entering the first letter(s) as separate argument(s). Colors available are:

blue	**g**reen	**c**yan
red	**m**agenta	**y**ellow
white		

The argument(s) entered set color in corresponding, consecutive regions. There are 10 regions total, divided as follows:

1 − 6 wedge colors	7 label links (no legend)
8 labels	9 title
10 frame	

You may enter less than the total number of regions.

−e **no erase**. Save the screen (don't erase before drawing). This is useful for creating composite graphic images.

−f **no frame**. Suppress frame around the window area. All titles and text are normally held inside the frame bounds.

−l **legend**. Draw a legend key above the pie instead of conventional labels. This option automatically sets fill mode on all pie wedges.

−p **pattern**. Next argument(s) are fill patterns for each consecutive wedge drawn. There are six (6) patterns to choose from (try them out to see what they are). **−p** alone sets pattern default at 1 2 3 4 5 0. If this option is not given, then exploded wedges ONLY are filled.

−r **rotate**. Next argument is clockwise rotation of the pie in degrees.

−t **total percent**. User defines a total percentage of a pie to be drawn, rather than a whole pie. If percentage values are used in labelling, then the total of these values will equal the above "percentage", not 100%. The argument entered must be real and between 0.0 and 1.0.

−T **title**. When given alone, a title will be expected as the first line in the standard input. (The program assumes normally that a title is not present). If a string follows the flag on the command-line, then this is taken to be the title and no title line will be read from the input. Long titles may be broken into two lines by inserting the character '!' (exclamation mark) where line break is desired.

−x **explode label**. Any and all wedges with a label containing characters matching the given string argument(s) will be drawn exploded from the chart. Multiple strings, separated by spaces, may be used to match several wedges. Strings that contain blanks must be surrounded by quotes.

−X **explode number**. Similar to **−x**, but wedges are indicated by the order of input to pie through the data stream, and are chosen by integer argument(s). (**−X 1** explodes the first wedge, **−X 1 2 3** explodes wedges 1 thru 3).

−v[pd] **data value**. Decides whether to print actual data value, percentage of the pie, both, or neither at the beginning of each wedge's label. **−vp** prints the percentage of the pie for each wedge, and is the default behavior; **−v** suppresses both percentage and data from the label; **−vdp** prints the data followed by percentage, and **−vpd** vice-versa.

−w **window**. Specify a window area on the screen. Next four (4) arguments are left, bottom, right, and top of window area in screen coordinates. (screen coordinates are 0.0 to 1.0). For example, **−w 0 .5 .5 1.0** will put a pie in the upper left corner. The image will automatically be scaled to fit within the window.

EXAMPLES

A sample of pie input (including title) for a chart of percentages of gross sales for several products would look as follows:

```
INVENTORY BREAKDOWN FOR 1983
Ball point pens          300.00
Paper clips              150.00
Writing pads             775.00
Pencil erasers           115.00
Notebooks                950.00
```

The first line of input data is interpreted as the title, or heading, to be displayed above the pie chart. Given that the **−T** flag is used in the command line.

To make a pie chart from the file ''pie.data'' on your device, highlighting the wedge representing notebooks, use this command line:

```
        pie −T −x books < pie.data  | plot
or      pie −T −X 5 < pie.data      | plot
```

To get the chart on the standard output, for saving as the file ''chart'', use:

```
pie −T −x books < pie.data > chart
```

Then chart can be printed on any graphics output device by typing:

```
plot < chart
```

NOTES

In order to run **pie**, the graphics shell parameter $GTERM must be set in your environment. You may place this in your **.profile** or **.login** for convenience.
When the legend option is used, a maximum of six (6) different shadings may be shown at one time.
There is a maximum of 20 wedges which can be displayed in a pie chart. All wedge values must be positive. (If a negative value is entered, its positive value will be used). Labels may overlap if the number of wedges becomes sufficiently large and the legend option is not used.

SEE ALSO

plot(1), scat(1g), bar(1g), chart(1g)

NAME

plot — graphics filters

SYNOPSIS

plot [**–T**terminal]

DESCRIPTION

plot reads plotting instructions (see **plot**(5)) from the standard input and produces graphic output for a specified *terminal*.

If no *terminal* type is specified, the environment parameter $GTERM (see **environ**(5)) is used. The $GTERM parameter can be set in your **.profile** or **.login** for convenience. If $GTERM is not set, then the environment parameter $TERM will be used. Terminals available are:

pro	DEC PRO monochrome graphics screen. On this device, the screen should be erased before any plotting is done. (see **erase**(1g).)
cpro	DEC PRO color graphics screen. The screen conditions for the pro option apply here also. You must have an extended bit map board (color board) to use this filter! (See **setscreen**(1g) for instructions on how to set up your color monitor).
4014	Tektronix 4014 storage scope.
ids	Integral Data System 460/560 printer.

EXAMPLES

Plot instructions in ''pdata'' for DEC PRO monochrome monitor (GTERM = pro):

```
plot < pdata
```

Create a graph:

```
graph <graphdata | plot
```

Take standard plotting instructions produced by program ''drawlines'' and plot on IDS printer:

```
drawlines | plot –Tids
```

FILES

/bin/lpr	spooler
/usr/bin/pplot	filter for DEC PRO monochrome graphics
/usr/bin/cpplot	filter for DEC PRO color graphics

/usr/bin/tek	filter for Tektronix 4014
/usr/bin/iplot	filter for IDS 460/560 only
/tmp/lpXXXX	temporary plot image (XXXX = unique number)

SEE ALSO

erase(1g), setscreen(1g), graph(1g), lpr(1), plot(3g), plot(5)

NAME

pr — print file

SYNOPSIS

pr [option] ... [file] ...

DESCRIPTION

pr produces a printed listing of one or more *files*. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, **pr** prints its standard input.

Options apply to all following files but may be reset between files:

—*n*	Produce *n*-column output. If an input text line is wider than the calculated column width, the input text is trimmed on the right.
+*n*	Begin printing with page *n*.
−**h**	Take the next argument as a page header.
−**w***n*	For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.
−**l***n*	Take the length of the page to be *n* lines instead of the default 66 (*n* must be greater than 10).
−**t**	Do not print the 5-line header or the 5-line trailer normally supplied for each page.
−**s***c*	Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.
−**m**	Print all *files* simultaneously, each in one column.

Inter-terminal messages via **write**(1) are forbidden during a **pr**.

EXAMPLES

Print the file ''list'' in 4-column format:

```
pr −4 list
```

Print the list files ''names'' and ''addresses'' simultaneously, with no headers and a page width of 80 characters:

```
pr −w80 −t −m names addresses
```

FILES

/dev/tty?	to suspend messages

SEE ALSO

cat(1)

DIAGNOSTICS

There are no diagnostics when **pr** is printing on a terminal.

NAME

prep — prepare text for statistical processing

SYNOPSIS

prep [**−diop**] file ...

DESCRIPTION

prep reads each *file* in sequence and writes it on the standard output, one 'word' to a line. A word is a string of alphabetic characters and imbedded apostrophes, delimited by space or punctuation. Hyphenated words are broken apart; hyphens at the end of lines are removed and the hyphenated parts are joined. Strings of digits are discarded.

The following option letters may appear in any order:

−d Print the word number (in the input stream) with each word.

−i Take the next argument as an 'ignore' file. These words will not appear in the output. (They will be counted, for purposes of the **−d** count).

−o Take the next argument as an 'only' file. Only these words will appear in the output. (All other words will also be counted for the **−d** count).

−p Include punctuation marks (single nonalphanumeric characters) as separate output lines. The punctuation marks are not counted for the **−d** count.

The 'ignore' and 'only' files contain only words, one per line.

EXAMPLES

Count the occurrences of the word "and" in the file *letter*:

```
prep −o word letter | wc −w
```

where the file *word* contains the word "and".

SEE ALSO

deroff(1)

NAME

prof — display profile data

SYNOPSIS

prof [**−a**] [**−l**] [file]

DESCRIPTION

prof interprets the file **mon.out** produced by the **monitor** subroutine. Under default modes, the symbol table in the named object file (**a.out** default) is read and correlated with the **mon.out** profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the **−a** option is used, all symbols are reported rather than just external symbols. If the **−l** option is used, the output is listed by symbol value rather than decreasing percentage.

In order for the number of calls to a routine to be tallied, the **−p** option of **cc** must have been given when the file containing the routine was compiled. This option also arranges for the **mon.out** file to be produced automatically.

FILES

mon.out	for profile
a.out	for namelist

SEE ALSO

monitor(3), profil(2), cc(1), plot(1)

BUGS

Beware of quantization errors.

NAME

ps — print system status information

SYNOPSIS

ps [**−lm**] [sec]

DESCRIPTION

ps displays information on current processes, on a per user basis. The first line gives the date, amount of free core (in K bytes), number of used swap blocks per total swap blocks available and number of used process slots per total process slots available. The next lines give the users' names, terminal names, process ID's, priorities, sizes (in K bytes), flags, status, estimated user and system time as percentages, and commands.

The process size does not include shared text space or shared data space. The user/system percentages are listed as zero unless **ps** is told to run repeatedly every *sec* seconds.

A '*' in the user name column indicates that the process is a child of one above. Login processes and processes without parents are listed with user ID names.

The flags shown are:

D	using shared data
I	locked in memory for raw I/O
L	locked in memory
S	swapped out
T	using shared text

If *sec* is specified, then the information will be redisplayed every *sec* seconds.

ps understands several flag arguments:

−l List all processes. Normally only user processes are listed (not including root unless the invoker is root). The **−l** flag causes all user, root, and system processes to be listed.

−m List only processes owned by me (the invoker).

The command name field includes the program name and the first arguments. The name may be blank or garbage if the process was swapped out for a time. Commands of the form '−#' are processes formed by **/etc/init** waiting for a login.

The "Other CPU" percentage time indicates the amount of time spent on all processes not displayed, as well as CPU idle time.

FILES

/venix	System namelist.
/dev	Searched to find terminal device names.
/tmp/ds_syms	Searched for symbol names on subsequent **ps** execution.
/etc/passwd	user names.

SEE ALSO

kill(1), suspend(1), size(1), adb(1)

BUGS

having nice priority −20. Percentage times given are not always correct.

NAME

pscreen — screen dump to graphics line printer
sscreen — screen save: copy image from display memory to file.
lscreen — screen load: copy image from file to display memory.

SYNOPSIS

pscreen [**−d**]
sscreen [filename]
lscreen [mode] [filename]

DESCRIPTION

Commands of the **pscreen** family deal with display memory on the video screen.

pscreen reads the contents of the video display memory (called a "screen dump"), and prints the image on a specified line printer via **lpr**(1). The **−d** option produces a double-sized copy. Devices supported are:

DEC LA100 letter-quality graphics printer
DEC LA50 graphics printer (will not support **−d** option)

sscreen copies the contents of the display memory into the specified file. If a filename is not passed as an argument, then the default file **/tmp/xscr.save** is used.

lscreen loads display memory from a file (default **/tmp/xscr.save**). In addition, the writing mode may be varied with a command-line option. Writing modes are bit operations between pixel information in the data file and the display memory which is mapped to the screen. The five available writing mode options are described below:

−xor **Exclusive-or** mode allows multiple images to be overlaid onto the same screen, and then removed separately while retaining the underlying image.
[exclusive 'OR' data to screen: memory ^ = data]

−mov **Absolute-move** mode overwrites anything on the screen and is good for clearing off previous images.
[move data to screen: memory = data]

−mvc **Move-complement** mode overwrites the screen with a reverse image, creating reverse-video effects.
[move complement of data to screen: memory = ~data]

–bis **Bit-set** mode writes only the set (turned-on) bits onto the screen. The current image is not destroyed. This mode is useful for creating composite images.
['OR' data to screen: memory |= data]

–bic **Bit-clear** mode writes only the clear (turned-off) bits onto the screen.
['AND' complement of data to screen: memory &= ~data]

Mode **mov** is the default.

The **sscreen** command, when called, will save EVERYTHING on the screen, including the user prompt! If you wish to omit the prompt in, for example, a graphics screen save, then both the **plot**(1g) and **sscreen** commands must be entered on the same command line:

```
plot < pict.image ; sscreen
```

Putting a series of commands in a shell script will also omit the prompt.

If an extended bit map board (color board) is present in the machine, then a full color screen may be saved and reloaded.

EXAMPLE

To save the graphics byte stream *picture.image* in image format, type:

```
plot < picture.image
sscreen pict.save
```

Now, the file may be retrieved at any time by typing:

```
lscreen pict.save
```

FILES

/tmp/xscr.save

SEE ALSO

plot(1g), plot(3g), plot(5), erase(1g), setscreen(1g)

NAME

ptx — permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. **ptx** produces output in the form:

```
.xx 'tail' 'before keyword' 'keyword and after' 'head'
```

The user may prepare an **nroff** macro definition for '.xx' for custom formatting of the index. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter; the default line length is 100 characters.

−w *n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

−g *n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

−o *only*
Use as keywords only the words given in the *only* file.

−i *ignore*
Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use /usr/lib/eign as the *ignore* file.

−b *break*
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

−r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

FILES

/bin/sort	
/usr/lib/eign	(not distributed with VENIX)

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME
pwd — working directory name

SYNOPSIS
pwd

DESCRIPTION
pwd prints the pathname of the working (current) directory.

SEE ALSO
cd(1)

NAME

quot — summarize file system ownership

SYNOPSIS

quot [option] ... [filesystem] ...

DESCRIPTION

quot prints the number of blocks in the one or more named *filesystems* currently owned by each user. If no *filesystems* are named, a list of default file system names is taken from **/etc/checklist**. The following options are available:

−n	Cause the pipeline **ncheck** *filesystem* | **sort** *+0n* | **quot** *−n filesystem* to produce a list of all files and their owners.
−c	Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.
−f	Print count of number of files as well as space owned by each user.

EXAMPLES

Summarize the number of files owned as well as the number of blocks used by each user on the file system **/dev/w0.usr**:

```
quot −f /dev/w0.usr
```

FILES

/etc/checklist	list of default file system names
/etc/passwd	to get user names

SEE ALSO

ls(1), df(1), checklist(4)

BUGS

Holes in files are counted as if they actually occupied space.

NAME

ranlib — convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

ranlib converts each *archive* to a form which can be loaded more rapidly by the loader, by adding a table of contents named **__.SYMDEF** to the beginning of the archive. It uses **ar**(1) to reconstruct the archive, so sufficient temporary file space must be available in the file system containing the current directory.

For medium to large size libraries, **ranlib** can significantly improve loading time.

SEE ALSO

ld(1), ar(1)

BUGS

Because generation of a library by **ar** and randomization by **ranlib** are separate, phase errors are possible. The loader **ld**(1) warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

NAME

restor — incremental file system restore

SYNOPSIS

restor key [arguments]

DESCRIPTION

restor is used exclusively to read any medium dumped with the **dump** command, and restore it to a file system on disk. The default device read from is floppy drive 0 (**/dev/f0**) on PRO/VENIX, RAINBOW/VENIX, and VENIX/11 on Micro PDP-11 systems; it is magtape drive 0 (**/dev/mt0**) on other VENIX/11 systems. This default may be overridden with the **f** modifier (see below).

The *key* specifies what is to be done. *key* is one or more of the following characters:

t	The date that the medium was made and the date that was specified in the **dump** command are printed. A list of all the i-numbers on the medium is also given.
r	The medium is read and loaded into the file system specified in *arguments*. This should not be done lightly (see below).
x	Each file on the medium is individually extracted into a file whose name is the file's i-number. If there are *arguments*, they are interpreted as i-numbers and only they are extracted.
c	If the medium overflows, increment the last character of its name and continue on that drive. (Normally it asks you to change media).
f	Read the dump from the next argument file instead of the default (**/dev/f0** for PRO/VENIX and RAINBOW/VENIX; **/dev/mt0** for VENIX/11).
i	All read and checksum errors are reported, but will not cause termination.
w	In conjunction with the **x** option, before each file is extracted, its i-number is typed out. To extract this file, you must respond with a **y**.

The **x** option is used to retrieve individual files. If the i-number of the desired file is not known, it can be discovered by following the file system directory search algorithm. First retrieve the *root* directory whose i-number is 1. List this file with **ls −fi 1**. This will give names and i-numbers of sub-directories. Iterating, any file may be retrieved.

The **r** option should only be used to restore a complete dump medium onto a clear file system or to restore an incremental dump medium onto this. (**restor** will ask you for confirmation before writing over the file system; hit CR to continue).

A **dump** followed by a **mkfs** and a **restor** can be used to save files, increase the size of a file system, and restore the files. The file system **restor**ed should be the same size or larger than the original file system.

EXAMPLES

Make new file system on */dev/w0.usr*, and restore from the default device:

```
/etc/mkfs /dev/w0.usr 13740
restor r /dev/w0.usr
```

Another **restor** can be done to get an incremental dump in on top of this.

FILES

/dev/f0	(PRO/VENIX and RAINBOW/VENIX default)
/dev/mt0	(VENIX/11 default)

SEE ALSO

dump(1m), mkfs(1m)

DIAGNOSTICS

There are various diagnostics involved with reading the media and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one medium, it may ask you to change media. Reply with a newline when the next medium has been mounted.

BUGS

There is redundant information on the medium that could be used in case of reading problems. Unfortunately, **restor** doesn't use it.

NAME

rm, rmdir — remove (unlink) files

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 in octal, as in

```
rm: file 775 mode
```

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** (force) option is given.

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, when specified with **−r**, as in **rm −ir**, verification is requested for each directory.

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

EXAMPLES

Remove all ".bak" files in current directory:

```
rm *.bak
```

Remove directory "olddir" and all of its contents:

```
rm −r olddir
```

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

scat — draw a scatter graph

synopsis

scat [option] ...

DESCRIPTION

scat with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The normal delimiter between pairs is a tab. The graph is encoded on the standard output for display by the **plot**(1) filters.

The pair coordinates of a point can also be followed by a nonnumeric string (again, the delimiter is a tab). The string is normally printed as a label beginning on the point, unless the legend option **−bl** is used on the command line. Labels never contain newlines.

The following options are recognized, each as a separate argument.

−b[l] **break**. Disconnect the graph after each label in the input, producing multiple lines. Attributes of these lines may be set in other options. If **l** is present, the labels are arranged together in the form of a key legend located above the graph.

−c[ls] **color**. Specify colors by entering the first letter(s) as separate argument(s). Colors available are:

blue	**g**reen	**c**yan
red	**m**agenta	**y**ellow
white		

The argument(s) entered set color in corresponding, consecutive regions. There are 15 regions total, divided as follows:

1 − 5 line colors	6 − 10 symbol colors
11 graph & ticks	12 all labels except title
13 title	14 gridlines
15 frame	

If the **cl** option is given, a line-preference coloring is established, in which all symbols will bear the same color. The user then need only enter one (1) argument for all symbol colors.
If the **cs** option is given, an analogous symbol-preference is established.
You may always enter less than the maximum number of arguments.

−e **no erase**. Save screen, don't erase before plotting. This is useful for creating composite graphic images.

−f **no frame**. Suppress frame around the window area. All titles and text are normally held inside the frame bounds.

−g **grid**. Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

−l **lines**. Next argument(s) are mode (style) of connecting lines: Patterns are 0 − blank, 1 − solid, 2 − dotted, 3 − dashed. Note: some devices may carry different line styles.

−s **symbols**. Next argument(s) given are symbol convention used in consecutive graphing lines. Symbol patterns are: 0 − box, 1 − circle, 2 − triangle, 3 − star, 4 − asterisk, 5 − point. If a single character other than a numerical digit is entered, then that character will be used as the symbol. Default pattern is 0,1,2,3,4.

−t **transpose**. Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis, and **−y** to the horizontal axis).

−T **title**. When given alone, the program will read the first three lines of standard input as the title, x axis label, and y axis label for the chart. If a string follows the flag on the command-line, then this is taken to be the title, only a title will be displayed on the graph, and no title or label lines will be read from standard input. Long titles may be broken into two lines by inserting the character '!' (exclamation mark) where line break is desired.

−x[la] **x axis**. If **l** is present, x axis is logarithmic. Next 3 arguments in succession, if entered, are lower and upper x limits, and grid spacing on x axis. All of these values have defaults and need not be entered. To ensure logarythmic axes, upper limit must be greater than the lower limit, and lower limit > 0.
If **a** is present, numbered automatic abscissas are applied (they are missing from the input). The next 2 arguments in succession, if entered, are spacing (default 1), and the starting point for automatic abscissas (default 0).
If a specified lower limit exceeds the upper limit, the axis is reversed. If one of the bounds is zero or negative, then a request for logarythmic axis will be ignored.
The **−xal** or **−xla** option is honored, but pay attention when using it!

−y[l] **y axis**. Similarly for y, except no auto-abscissa mode.

−w **window**. Specify window area on screen. Next four(4) arguments are left, bottom, right, and top of window area in screen coordinates. (screen coords are 0.0 to 1.0). For example, the command-line option **−w .5 .5 1.0 1.0** will put a graph in the upper-right hand corner of the screen.

EXAMPLES

Draw a graph from coords in a file called "receipts", and pipe it to the plot filter:

```
scat < receipts | plot
```

NOTES

In order to run **scat**, the graphics termcaps parameter $GTERM must be set in the environment. It is convenient to do this in your **.profile** or **.login** for convenience.
scat stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

SEE ALSO

spline(1), plot(1), bar(1g), pie(1g), chart(1g)

NAME

sed — stream editor

SYNOPSIS

sed [**−n**] [**−e** script] [**−f** sfile] [file] ...

DESCRIPTION

sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. (For an introduction to **sed** see chapter 2 in the *Document Processing Guide*). The **−f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **−e** option and no **−f**'s, the flag **−e** may be omitted. The **−n** option suppresses the default output.

A script consists of editing commands, one per line, of the following form:

> [address [, address]] function [arguments]

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

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

> The escape sequence '\n' matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

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

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

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

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

An argument denoted *text* consists of one or more lines, all but the last of which end with '\' to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an 's' command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An argument denoted *rfile* or *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a\
text

Append. Place *text* on the output before reading the next input line.

(2) b *label*

Branch to the ':' command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) c\
text

Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2) D Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) g Replace the contents of the pattern space by the contents of the hold space.

(2) G Append the contents of the hold space to the pattern space.

(2) h Replace the contents of the hold space by the contents of the pattern space.

(2) H Append the contents of the pattern space to the hold space.

(1) i\
text Insert. Place *text* on the standard output.

(2) l List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two digit ASCII, and long lines are folded.

(2) n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) N Append the next line of input to the pattern space with an embedded newline. (The current line number changes).

(2) p Print. Copy the pattern space to the standard output.

(2) P Copy the initial segment of the pattern space through the first newline to the standard output.

(1) q Quit. Branch to the end of the script. Do not start a new cycle.

(2) r *rfile*
Read the contents of *rfile*. Place them on the output before reading the next input line.

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

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) t *label*
Test. Branch to the ':' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a 't'. If *label* is empty, branch to the end of the script.

(2) w *wfile*
Write. Append the pattern space to *wfile*.

(2) x Exchange the contents of the pattern and hold spaces.

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

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

(0) : *label*
: This command does nothing; it bears a *label* for 'b' and 't' commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching '}' only when the pattern space is selected.

(0) An empty command is ignored.

EXAMPLES

To print only the 7th through 34th lines of a file:

```
sed −n ′7,34p′ < file
```

To delete the first lines in a file through a line with the string "START", substitute the string "hello there" for all occurrences of the string "hello", and delete all lines starting with one containing the string "crumple" until the end of the file, create the file *sedscript* containing:

```
1,/START/d
s/hello/hello there/g
/crumple/,$d
```

and then give the command:

```
sed −f sedscript < input > output
```

SEE ALSO

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

NAME

setscreen — set video screen mode

SYNOPSIS

setscreen [**–cm**]

DESCRIPTION

The color monitor on the DEC PRO requires a higher intensity display than the monochrome monitor. **setscreen** allows the user to set the screen to the proper intensity. The choices are:

mode	description
c	1024x240 color graphics
m	1024x240 monochrome graphics

If an extended bit map board (color board) is present, the color palette will be reset to its default colors (see **plot**(3g)). When the command is given, the mode is stored in permanent memory. Thus, the screen may be reset at a later time by typing in **setscreen** without a flag. The permanent memory can be changed by using the appropriate flag, but should be done only if the monitor is changed.

EXAMPLES

Set display for a color monitor:

```
setscreen –c
```

Set display for a monochrome monitor:

```
setscreen –m
```

Reset display to current mode:

```
setscreen
```

NOTES

Do not set to color mode if you have a monochrome monitor; it can damage the screen!

SEE ALSO

pscreen(1g), plot(1g)

NAME

sh — Bourne shell (command interpreter)

SYNOPSIS

sh [**−ceiknrstuvx**] [arg] ...

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. See subsection ''Invocation'' for the meaning of arguments to the shell. ''An Introduction to the Shell'' in the *User Guide* provides tutorial information.

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 200+**status** if it terminates abnormally (see **signal**(2) for a list of status values).

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

A *list* is a sequence of one or more *pipelines* separated by ;, **&**, **&&** or || and optionally terminated by ; or **&**. ; and **&** have equal precedence which is lower than that of **&&** and || **&&** and || also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol **&&** (||) causes the *list* following to be executed only if the preceding *pipeline* returns a zero (non zero) value. Newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. 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 in the **for** word list. If **in** *word* ... is omitted then

in "**$@**" is assumed. Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*]
: 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.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
: The *list* following **if** is executed and if it returns zero the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero the *list* following **then** is executed. Failing that the **else** *list* is executed.

while *list* [**do** *list*] **done**
: A **while** command repeatedly executes the **while** *list* and if its value is zero executes the **do** *list* otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list*. **until** may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a subshell.

{ *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 in esac for while until do done { }

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 newlines 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*] ...

$ {*parameter*}
: A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters *** @ # ? – $!**. The value, if any, of the parameter is substituted. The braces are required only when *parameter parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its

name. 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. **$0** is set from argument zero when the shell is invoked.

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

${*parameter* = *word*}
: If *parameter* is not set 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 then substitute its value; otherwise, print *word*. and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter* + *word*}
: If *parameter* is set 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, for example, echo ${d – \`pwd\`} will only execute *pwd* if *d* is unset.)

The following *parameters* are automatically set by the shell.

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

The following *parameters* are used but not set by the shell.

HOME	The default argument (home directory) for the **cd** command.
PATH	The search path for commands (see **execution**).
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 **newline.**

Blank interpretation
After parameter and command substitution, any 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 /, and the character /, must be matched explicitly.

*****	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the characters enclosed. A pair of characters separated by – matches any character lexically between the pair.

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

; & ( ) | < > newline space tab

A character may be **quoted** by preceding it with a \\ . **\\newline** is ignored. All characters enclosed between a pair of quote marks ('), except a single quote, are quoted. Inside double quotes ("") 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 newline is typed and further input

is needed to complete a command then the secondary prompt (**$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 (by seeking to the end); otherwise the file is created.

<< *word*
The shell input is read up to a line the same as *word*, or 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, **\ newline** is ignored, and **** is used to quote the characters **\ $ `** and the first character of *word*.

<&*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 to be a duplicate of file descriptor 1.

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

Environment
The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see **exec**(2). 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. Note also that no modification in the value of **export**ed parameters is retained once the current process exits and returns to the calling shell.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to *parameters*. Thus these two lines are equivalent

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

If the −**k** flag is set, all keyword arguments are placed in the environment, even if they occur after the command name. The following prints 'a=b c' and '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. (But see also **trap**).

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 an **exec**(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (:). The default path is **:/bin:/usr/bin**. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands
The following commands are executed in the shell process and except where specified no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. *file* Read and execute commands from *file* and return. The search path **$PATH** is used to find the directory containing *file*.

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

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

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

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

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

exit [*n*] Causes a non-interactive 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 exit from the shell).

export [*name* ...]
: The given names are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given then a list of exportable names is printed.

login [*arg* ...]
: Equivalent to **exec login** *arg* ....

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

read *name* ...
: One line is read from the standard input; successive words of the input are assigned to the variables *name* in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

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

set [**−eknptuvx** [*arg* ...]]

−e	If non-interactive then exit immediately if a command fails.
−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.
−	Turn off the **−x** and **−v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**.

Remaining arguments are positional parameters and are assigned, in order, to **$1**,**$2**, etc. If no arguments are given then the values of all names are printed.

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

times Print the accumulated user and system times for processes run from the shell. Special shell commands (handled directly by the shell) do not cause these figures to increase.

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. 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 invoked commands. If *n* is 0 then the command *arg* is executed on exit from the shell, otherwise upon receipt of signal *n* as numbered in **signal**(2). *trap* with no arguments prints a list of commands associated with each signal number.

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

wait Wait for all currently active child processes. The return code from this command is that of the last process waited for.

Invocation

If the first character of argument zero is –, commands are read from **$HOME/.profile**, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

–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. 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 (as told by **gtty**) then this shell is *interactive*. In this case the terminate signal SIGTERM (see **signal**(2)) is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptable). In all cases SIGQUIT is ignored by the shell.

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

FILES

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

SEE ALSO

test(1), exec(2),
''An Introduction to the Shell'' in the *User Guide*

DIAGNOSTICS

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 **exit**(1)). The following diagnostic messages may be given by the shell (explanations are given as deemed necessary):

Alarm call — Alarm signal (signal 14) received; may be due to program which sets alarm and exits before catching it.

arg list too long — Too many arguments were passed to a command, usually due to use of '*' or other wildcards matching many file names. May be solved by not using long pathnames in wildcards. (For example, instead of typing:

```
pr /u0/fred/src/*
```

type:

```
cd /u0/fred/src
pr *
```

Maximum number of characters passed as arguments is roughly 1500.

Asynchronous I/O — Asynchronous I/O complete signal (signal 16) received.

Asynchronous I/O error (IOT trap) — Signal 6 (IOT trap or Async I/O error) received.

bad directory — May be due to attempt to **cd** to directory in which you have read but not execute permission.

bad number — Illegal number used with trap or exit.

bad option(s) — Bad flag passed to shell.

bad substitution — Problems evaluating shell variables.

Bad system call	Program running made illegal system call. This should never happen normally; may be due to corrupted binary program.
bad trap	Problems with "trap" of signals.
Bus error	Hardware problems, or possibly illegal memory access.
cannot create	File can not be created — you may not have write permission in directory, or file may already exist and not have write permission for you.
cannot execute	File can not be executed — no execute permission for you.
cannot make pipe	See system administrator — pipe device may be incorrectly setup.
cannot open	File can not be opened — does not exist, or no permission for you.
cannot shift	Problems in "shift" command on command-line arguments. (No more arguments left to shift.)
core dumped	Memory image of program dumped in file called "core". Occurs if program dies under certain conditions — commonly due to illegal memory access caused by a pointer bug.
Stack overflow	The stack has overflowed a program compiled and linked with the **−z** flag (see **ld**(1), **cc**(1))
end of file	
Floating exception	Floating point error: due either to use of floating point instruction without hardware or simulator, or floating point error condition (e.g. divide by zero). See **−f** flag in **cc**(1).
Hangup	Hangup signal (signal 1) received. Normally generated by modems.
Illegal instruction	Illegal instruction signal (signal 4) received; binary program is corrupted?
illegal io	Due to attempt to redirect I/O of built-in shell command (e.g. "cd" or "read").

is not an identifier	Bad syntax in shell commands. May be given when shell expects a variable name and none is given.
is read only	Attempt to change a variable previously set "read only".
Killed	Program with given process ID has been killed.
Memory fault	Program used memory illegally — commonly due to pointer bug.
no space	Shell has run out of memory space; reduce number of variables, or make shell script smaller.
not found	Given program can not be found. The shell looks in all directories given in the PATH variable, generally the current directory, /bin and /usr/bin.
parameter not set	Shell variable not assigned a value.
Quit	Quit signal (signal 3) received.
syntax error	
Terminated	Terminate signal (signal 15) received. May be generated when program is stopped with kill command.
text busy	Attempt to execute "pure" program in file that is open for I/O, or attempt to write to file of pure program being executed. "Pure" programs are generated by the loader (**ld**(1)) so that their code portions are shared in memory by many users, and should not be overwritten.
too big	Program is too big to run; run the **size**(1) command to check the size of the program.
Trace/BPT trap	Trap signal (signal 5) received.
unexpected	Shell key-word (like "for" or "if") used inappropriately, or shell script ended in the middle of such a do/done or other construction.

you have mail	Good news, we hope. Run the "mail" command to read your mail.

BUGS

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 the file by another name.

NAME

size — size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

size prints the (decimal) number of bytes required by the text, initialized data, and bss (uninitialized data) portions, and their sum in octal and decimal, of each object-file argument. If no file is specified, **a.out** is used.

SEE ALSO

a.out(4), nm(1), strip(1)

NAME

sleep — suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

If *time* is positive, **sleep** suspends execution for *time* seconds. If *time* is negative, the **sleep** lasts for the absolute value of *time* in 1/60ths of a second. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while :
do
        command
        sleep 37
done
```

time should be between −32768 and 32767.

Because of system overhead, the sleep may be delayed an arbitrary amount, and there is very little point in trying to time a sleep with better than one second accuracy.

SEE ALSO

alarm(2), sleep(3)

NAME

sort — sort or merge files

SYNOPSIS

sort [**−mubdfinrtxc**] [+pos1 [−pos2]] ... [**−o** name] [**−T** directory] [name] ...

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. It performs different types of sort on multiple column text files with uniform field delimiters. 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. In the case of two equal letter fields, one uppercase and the other lowercase, the uppercase comes first in sorted order.

i Ignore characters outside the ASCII range 040−0176 in non-numeric comparisons; lines with the outside characters are appended at the end of the sorted list.

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' or the delimiter separating fields is *x*, rather than white space.

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 nonblank 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 nonempty nonblank 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. A quick check for properly sorted files.

m Merge only, the input files are already sorted.

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

T The next argument is the name of a directory in which temporary files should be made.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison. This option prevents duplicate occurrences from being printed out.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words. First, capitalized words differ from uncapitalized and come first in sorted order. In the second example, there is no difference between capitalized and uncapitalized words.

```
sort list
sort −u −f list
```

Print the password file (**passwd**(4)) sorted by user id number (the third colon-separated field).

```
sort −t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options −**um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort −um +0 −1 dates
```

FILES

/usr/tmp/stm*, /tmp/*: first and second tries for temporary files

SEE ALSO

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

DIAGNOSTICS

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

BUGS

Very long lines are silently truncated.

NAME

spell, spellin, spellout — find spelling errors

SYNOPSIS

spell [−**v**] ... [file] ...

/usr/dict/spellin [list]

/usr/dict/spellout [−**d**] list

DESCRIPTION

spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

spell ignores most **nroff**, **tbl** and **neqn**(1) constructions.

Under the −**v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. (Since **spell** is a shell script, changes to default settings can be easily made). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g. thier = thy − y + ier) that would otherwise pass.

Two routines help maintain the hash lists used by **spell**. Both expect a list of words, one per line, from the standard input. **spellin** adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. **spellout** looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option −**d**) the hash list.

EXAMPLES

Check file *paper1* for spelling mistakes; collect output in *errs*:

```
spell paper1 > errs
```

Use **fgrep** (see **grep**(1)) to find lines that have lines matching words in *errs*:

```
fgrep −f errs paper1
```

(This may produce extraneous lines if a word in *errs*, while itself not a correctly spelled word, is contained within a legitimate word).

Add new words in file *nwords* to the hash list of good words, to produce a new hash list:

```
/usr/dict/spellin /usr/dict/hlista < nwords > /usr/dict/newhlista
```

Add new words in file *bwords* to the hashed *stop* list of spelling errors which otherwise slip through:

```
/usr/dict/spellout /usr/dict/hstop < bwords > /usr/dict/newhstop
```

There are approximately 24,000 words provided in the hashed word list.

FILES

D=/usr/dict/hlista	hashed spelling list
S=/usr/dict/hstop	hashed stop list
H=/usr/dict/spellhist	history file
/usr/lib/spell	

SEE ALSO

deroff(1), sort(1), tee(1), sed(1)

BUGS

Spell creates temporary files, **temp** and **temp1**, in the current directory, and then removes them when it's done. If you break out of **spell**, you may want to remove **temp** and **temp1** yourself.

NAME

spline — interpolate smooth curve

SYNOPSIS

spline [option] ...

DESCRIPTION

spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by **graph**(1).

The following options are recognized, each as a separate argument.

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

−k The constant k used in the boundary value computation

$$y''_0 = ky''_1, \quad y''_n = ky''_{n-1}$$

is set by the next argument. By default $k = 0$.

−n Space output points so that approximately **n** intervals occur between the lower and upper x limits. (Default **n** = 100).

−p Make output periodic, i.e. match derivatives at ends. First and last input values should normally agree.

−x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1)

DIAGNOSTICS

When data is not strictly monotone in x, **spline** reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

NAME

split — split a file into pieces

SYNOPSIS

split [–*n*] [file [name]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if '–' is given in its stead, then the standard input file is used.

EXAMPLES

To split files *lnames* into segments of 17 lines calling each one *partialaa*, *partialab*, etc.

```
split –17 lnames partial
```

NAME

strip — remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

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

The effect of **strip** is the same as use of the **−s** option of **ld**(1).

FILES

/tmp/stm?	temporary file

SEE ALSO

ld(1)

NAME

stty — set terminal options

SYNOPSIS

stty [option ...]
STTY [option ...]

DESCRIPTION

stty sets certain I/O options on the current output terminal. With no argument, it reports the current settings of the options.

STTY is identical to **stty**, but given an upper-case name so that upper-case-only terminals can also set the terminal modes.

stty can be used to set the terminal modes on any device merely by redirecting the standard output to that device through the shell '>' construction.

If your terminal has been placed into certain non-standard modes, you will have to use line-feeds (^J) instead of carriage returns at the end of lines.

The option strings are selected from the following set:

scroll	stop every 20 lines of screen output until a ^Q or any other character is typed; useful for keeping text from scrolling off CRT screens
–scroll	disable **scroll** option
crt	make 'delete' echo as a backspace-space-backspace to erase a character from a CRT screen
–crt	make 'delete' echo a '<' to indicate a deleted character, appropriate for hard-copy terminals
raw	raw mode input (no erase, kill, interrupt, quit, EOT; parity bit passed back)
–raw	negate raw mode
cooked	same as **–raw**
cbreak	make each character available to **read**(2) as received; no erase and kill
–cbreak	make characters available to **read** only when newline is received

−nl	allow carriage return for newline, and output CR−LF for carriage return or newline
nl	accept only newline to end lines
echo	echo back every character typed
−echo	do not echo characters
lcase	map upper case to lower case
−lcase	do not map case
−tabs	replace tabs by spaces when printing
tabs	preserve tabs
ek	reset erase and kill characters back to normal (''DELETE key'' and ''^U'', respectively)
erase *c*	set erase character to *c*. *c* can be of the form '^*X*' which is interpreted as a 'control−X'.
kill *c*	set kill character to *c*. '^*X*' works here also.
even	allow even parity
−even	disallow even parity
odd	allow odd parity
−odd	disallow odd parity (parity checking is not supported on all interfaces)
hup	hang up dataphone on last close.
−hup	do not hang up dataphone on last close. (this option is not supported on all interfaces)
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.

EXAMPLES

Set current terminal modes appropriate to normal CRT usage, with the terminal output stopping every 20 lines waiting for a ^Q.

```
stty cooked echo crt −nl scroll
```

Set terminal *tty02* to *300* baud:

```
stty 300 > /dev/tty02
```

SEE ALSO

ioctl(2)

NAME

su — substitute user id temporarily

SYNOPSIS

su [userid]

DESCRIPTION

su changes the user's identity temporarily to that of *userid* (a user name) and invokes that user's shell (normally **sh** or **csh**(1)), without changing the current directory or the user environment (see **environ**(5)). If the new user has set a login password, that same password must be provided before the substituted identity is granted. The new user ID stays in force until the shell exits, (that is, you issue a ^D to logout) after which the user is returned to his original shell and directory. Changes of the current directory or environment, made under the substituted identity, are lost when the user resumes his original identity.

If no *userid* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

SEE ALSO

sh(1), csh(1)

NAME

sum — sum and count blocks in a file

SYNOPSIS

sum file

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinguishable from end of file on most devices; check the block count.

NAME

suspend, resume — suspend and resume execution of a process

SYNOPSIS

suspend pid
resume pid

DESCRIPTION

suspend execution of the process with process id number *pid*. Processes being traced cannot be suspended.

resume restarts the suspended process with number *pid*. If the processes do not have the same effective ID as you, you must be the super-user to suspend or resume them.

SEE ALSO

kill(1), suspend(2)

NAME

tail — deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**]] [file]

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +*number* from the beginning, or −*number* from the end of the input. *number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b** or **c**. When no units are specified, counting is by lines. When no *number* is specified, default is '−10'.

EXAMPLES

Print file *bumble.bee* from the 50th line on:

```
tail +50 bumble.bee
```

SEE ALSO

dd(1)

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

tar — tape archiver

SYNOPSIS

tar [key] [name ...]

DESCRIPTION

tar saves and restores directories and files on floppy diskettes, magtape or other media. Each directory name given is dumped with all files and (recursively) subdirectories in it.

tar's actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be saved or restored.

tar will prompt the user to change the volume (disk or tape) when it is filled up, allowing for the saved files to be spread across multiple volumes. The **s** option can be used to specify the number of blocks on the volume. **tar** will not prompt the user to change media when extracting files from a previous back-up, so the user must either run the **tar** command once for each volume, or select the volume which contains the needed file.

The default output device is floppy drive 0 (/**dev/f0**) on PRO/VENIX, RAINBOW/VENIX, and VENIX/11 on Micro PDP-11 systems; it is magtape drive 0 (/**dev/mt0**) on other VENIX/11 systems. This default may be overridden with the **f** modifier (see below).

The function portion of the key is specified by one of the following letters:

c Create (initialize) a new diskette; **c** clears the diskette of all files, and writes at the beginning of the diskette. This command implies **r**.

r The named files are written on the end of the diskette or tape. The **c** function implies this, and the medium must have been previously initialized with a **c**. If the output is longer than one volume, subsequent volumes are written on from the beginning; that is, they are automatically initialized, and previous information on them is overwritten.

x The named files are extracted from the diskette. If the named file matches a directory whose contents had been written onto the diskette, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the medium is extracted. Note that if multiple entries specifying the same file are on the diskette, the last one (most recently saved) overwrites all earlier.

t The names of the specified files are listed each time they occur on the diskette. If no file argument is given, all of the names on the diskette are listed.

u The named files are added to the diskette if either they are not already there or have been modified since last put on the diskette. This option can not be used to update previous **tar**'s which have spread across multiple media; only the currently loaded diskette is looked at when deciding what files need updating. (However, the files written out may spread over several volumes as normal).

The following characters may be used in addition to the letter which selects the function desired.

v Normally **tar** does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the diskette entries than just the name. When used with the **c**, **r**, or **t** options, it also causes **tar** to list the total number of blocks used on the volume, giving the user a rough idea of what room is left on the volume for more files.

s The next argument gives the size of the output device in blocks.

d Only save or restore files modified within the last number of days specified by the next argument. **tar** will calculate and print the effective date. When using this flag to selectively restore files from tape or disk, the date compared against for each file is the last-modified date of the file when it was **tar**'ed out, not the date that **tar** occurred.

w causes **tar** to print the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is performed. Any other input means don't do it.

f causes **tar** to use the next argument as the name of the output device instead of the default name. If the name of the file is '−', tar writes to standard output or reads from standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a filter chain. **tar** can also be used to move hierarchies with the command

```
cd fromdir; tar cf − . | (cd todir; tar xf −)
```

b causes **tar** to use the next argument as the blocking factor for diskette (or tape) records. The default is 1, the maximum is 20. The block size is determined automatically when reading disks (key letters 'x' and 't'). On magtape, if block size is other than 1, a **f** key (above) should be used to specify the raw version of the device. On devices other than magtape, block size is not normally important.

l tells **tar** to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m tells **tar** to not restore the modification times. The mod time will be the time of extraction.

0,...,7 A digit 0 through 7 as a modifier selects the unit number to use on the floppy or tape drive (see notes on **f** key to determine your default). Thus on systems whose default device is floppy drive 0 (**/dev/f0**), files may be saved on floppy drive 1 by typing **tar c1...** . This is equivalent to typing **tar cf /dev/f1...** .

The number of blocks used by **tar** when writing a volume can be calculated by adding the number of blocks in all files, plus one additional header block per file, plus 2 blocks for the volume trailer.

tar will not understand '*', '?' or other wildcard notation when extracting files.

tar will not work with mountable diskettes; using **tar** to write on a mountable diskette will destroy it, and attempting to extract files from a mountable disk using **tar** will cause error messages. Similarly, attempts to mount a **tar** format diskette are ill-fated.

FILES

/dev/f?
/tmp/tar*
awk, sort, pwd, mkdir(1)

DIAGNOSTICS

Complains about bad key characters and diskette read/write errors. Complains if enough memory is not available to hold the link tables.

EXAMPLES

To store the directories *nile* and *cairo* on the default device (floppy or tape):

```
tar c nile cairo
```

To add the file *pyramids* in the directory *cairo*:

```
tar r cairo/pyramids
```

To add another file *sphinx* to the end of the last file on floppy unit 1:

```
tar rvf /dev/f1 sphinx
```

tar, when used with the **v** flag, will also tell you how many blocks have been used on the medium.

To create a new diskette in drive 1 holding only those files in *cairo* modified in the last seven days:

```
tar cdf 7 /dev/f1 cairo
```

To view the files on the default **tar** device, simply type:

```
tar tv
```

To extract all files:

```
tar xv
```

To extract a specific file in a directory, for example, file *pyramids* in the directory *cairo*:

```
tar xv cairo/pyramids
```

tar always restores files using the same pathnames given when the files were saved. This means that full pathnames (that is, names beginning with a '/') should be avoided. For example, suppose that user Dennis has a home directory

```
/usr/dennis
```

and a directory beneath that:

```
/usr/dennis/letters
```

If directory *letters* is saved using the command

```
tar c /usr/dennis/letters
```

a subsequent extraction, with

```
tar x
```

will always bring back the old files to */usr/dennis/letters*, overwriting any files of the same name in that directory. For this reason, it is desirable to save files according to a "relative" name, for example from Dennis' home directory (*/usr/dennis*):

```
cd                              (change directory to /usr/dennis)
tar c letters
```

Now the command

```
tar x
```

will extract the saved files into whatever directory Dennis is in when he issues the command. So to restore the saved files to a temporary directory, the following commands could be used:

```
cd
mkdir temp
cd temp
tar x
```

Now the extracted files are placed in

```
/usr/dennis/temp/letters
```

while the directory

```
/usr/dennis/letters
```

is untouched. Of course, if Dennis really wishes to extract his files into */usr/dennis/letters*, and overwrite any copies of the same files on disk, he can always use the commands

```
cd
tar x
```

and this will happen.

NOTES

Unlike some other versions of **tar**, VENIX **tar** saves directory and device node entries as well as ordinary files. This feature does not affect compatibility with other versions of **tar**, except that device nodes saved by VENIX **tar** may be turned into zero-length ordinary files when extracted by other versions of **tar**, and directory mode and owner information may be lost. Extraneous error messages may also be produced.

BUGS

There is no way to ask for the *n*-th occurrence of a file.

The **u** option can be slow.

The **b** option should not be used with archives that are going to be updated. If the archive is on a disk file the **b** option should not be used at all, as updating an archive stored in this manner can destroy it.

The current limit on file name length is 100 characters.

The modification times of non-empty directories extracted by **tar** will always be the time of extraction, not the time saved, since the modification time becomes updated when the first file is created inside the directory.

NAME

tbl – format tables for nroff

SYNOPSIS

tbl [files] ...

DESCRIPTION

tbl is a preprocessor for formatting tables for **nroff**(1). The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and reformatted. Details are given in the reference manual. The **col** filter is often used in conjunction with **tbl** to provide line backspacing.

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

Household Population

Town	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Branchburg	1644	3.49
Bridgewater	7897	3.81
Far Hills	240	3.19

If this table is in a file called ''newjersey'', which also has text, the command line to print the file will be:

```
tbl newjersey | nroff | col | lpr -n
```

If no arguments are given, **tbl** reads the standard input, so it may be used as a filter. When it is used with **neqn** the **tbl** command should be first, to minimize the volume of data passed through pipes.

SEE ALSO

nroff(1), neqn(1)
''Table Formatting Program'' in the *Document Processing Guide*

NAME

tee — pipe fitting

SYNOPSIS

tee [**−i**] [**−a**] [file] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *files*. Option **−i** ignores interrupts; option **−a** causes the output to be appended to the *files* rather than overwriting them.

EXAMPLES

Send the output from program *xprog* to the terminal, and also save in file *xprog.output*:

```
xprog | tee xprog.output
```

NAME

test — condition command

SYNOPSIS

test expr

DESCRIPTION

test evaluates the expression *expr*, and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. **test** returns a non zero exit if there are no arguments.

The following primitives are used to construct *expr*.

−**r** file	true if the file exists and is readable.
−**w** file	true if the file exists and is writable.
−**f** file	true if the file exists and is not a directory.
−**d** file	true if the file exists and is a directory.
−s file	true if the file exists and has a size greater than zero.
−**t** [fildes]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
−**z** s1	true if the length of string *s1* is zero.
−**n** s1	true if the length of the string *s1* is nonzero.
s1 = s2	true if the strings *s1* and *s2* are equal.
s1 **!**= s2	true if the strings *s1* and *s2* are not equal.
s1	true if *s1* is not the null string.
n1 −**eq** n2	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons −**ne**, −**gt**, −**ge**, −**lt**, or −**le** may be used in place of −**eq**.

These primaries may be combined with the following operators:

!	unary negation operator.
−**a**	binary **and** operator.
−**o**	binary **or** operator.
(expr)	parentheses for grouping.

−a has higher precedence than **−o**. Notice that all the operators and flags are separate arguments to **test**. Notice also that parentheses are meaningful to the Shell and must be escaped.

EXAMPLES

The following shell script checks if the name given as argument is an existing file:

```
if test −f $1
then echo "$1 is an existing file"
else echo "$1 is not an existing file"
fi
```

SEE ALSO

sh(1), find(1), expr(1)

NAME

time — time a command

SYNOPSIS

time command

DESCRIPTION

The given *command* is executed; after it is complete, **time** prints the elapsed "real" time during the *command*, the time spent in user routines, and the time spent handling system calls. The difference between the sum of the user and system times, and the elapsed time, is accounted for by time spent by the system executing other precesses or waiting for I/O to complete. Times are reported in seconds.

The times are printed on the diagnostic output stream.

time exists as a built-in command in the C shell, and provides slightly different information; see **csh**(1).

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tk — paginator for the Tektronix 4014

SYNOPSIS

tk [**−t**] [−*N*] [−**p***L*] [file]

DESCRIPTION

The output of **tk** is intended for a Tektronix 4014 terminal. **tk** arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page **tk** waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command !*command* will send the *command* to the shell.

The command line options are:

−t	Don't wait between pages; for directing output into a file.
−*N*	Divide the screen into *N* columns and wait after the last column.
−**p***L*	Set page length to *L* lines.

SEE ALSO

pr(1)

NAME

touch — update date last modified of a file

SYNOPSIS

touch [**−c**] file ...

DESCRIPTION

touch attempts to set the modified date of each *file*. This is done by reading a character from the file and writing it back.

If a *file* does not exist, an attempt will be made to create it unless the **−c** option is specified.

NAME

tr — translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **−cds** may be used:

−c complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal. The replacement action of *string2* takes place for all ACSII characters that are not specified in *string1*.

−d deletes all input characters in *string1*.

−s squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a* − *b* means a range of characters from *a* to *b* in increasing ASCII order. The character '\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A '\' followed by any other character stands for that character.

EXAMPLES

Create a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. Note that the **−c** flag replaces all characters that are not letters with a newline. The second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr −cs A−Za−z '\012' <file1 >file2
```

SEE ALSO

ed(1)

BUGS

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

true does nothing, successfully. **false** does nothing, unsuccessfully. They are typically used in input to the Bourne shell, **sh**(1), such as:

```
while true
do
        command
done
```

(Note that **true** can be done more efficiently simply with the null command ':').

SEE ALSO

sh(1)

DIAGNOSTICS

true has exit status zero, **false** nonzero.

NAME

tsort — topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

tsort produces on the standard output a totally ordered list of items consistent with a particular ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty — get terminal name

SYNOPSIS

tty

DESCRIPTION

tty prints the pathname of the user's terminal.

DIAGNOSTICS

'not a tty' if the standard input file is not a terminal.

SEE ALSO

who(1), stty(1)

NAME

uniq — report repeated lines in a file

SYNOPSIS

uniq [−**udc** [+n] [−n]] [input [output]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see **sort**(1). If the −**u** flag is used, just the lines that are not repeated in the original file are output. The −**d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the −**u** and −**d** mode outputs.

The −**c** option supersedes −**u** and −**d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

NAME

uucp, uulog — unix to unix copy

SYNOPSIS

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

uulog [option] ...

DESCRIPTION

uucp is used for file transfer between computers running UNIX or VENIX. It copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form

system-name!pathname

where 'system-name' is taken from a list of system names which **uucp** knows about. Shell metacharacters ?*[] appearing in the pathname part will be expanded on the appropriate system. Note: when using the C shell (csh), exclamation points must be preceded by a '\' to prevent interpretation.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~*user*; where *user* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

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

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

The following options are interpreted by *uucp*.

−d Make all necessary directories for the file copy.

−c Use the source file when copying out rather than copying the file to the spool directory.

−m Send mail to the requester when the copy is complete.

uulog maintains a summary log of **uucp** and **uux**(1) transactions in the file '/usr/spool/uucp/LOGFILE' by gathering information from partial log files named '/usr/spool/uucp/LOG.*.?'. It removes the partial log files.

The options cause **uulog** to print logging information:

−**s***sys*	Print information about work involving system *sys*.
−**u***user*	Print information about work done for the specified *user*.

FILES

/usr/include/indent.h	— system name
/usr/spool/uucp	— spool directory
/usr/lib/uucp/*	— other data and program files

SEE ALSO

uux(1), mail(1)
"UUCP Implementation Description" in the *Installation and System Manager's Guide.*

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

BUGS

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

NAME

uux — unix to unix command execution

SYNOPSIS

uux [–] command-string

DESCRIPTION

uux will gather 0 or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

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

File names may be one of

(1) a full pathname;

(2) a pathname preceded by ~*xxx*; where *xxx* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

The '–' option will cause the standard input to the **uux** command to be the standard input to the command-string.

For example, the command

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

will get the f1 files from the usg and pwba machines, execute a **diff**(1) command and put the results in f1.diff in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments. When using the C shell (**csh**(1)), exclamation marks fall into this category also.

FILES

/usr/include/indent.h	— system name
/usr/uucp/spool	— spool directory
/usr/uucp/*	— other data and programs

SEE ALSO

uucp(1)
"UUCP Implementation Description" in the *Installation and System Manager's Guide.*

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from **uux**. Typically, a restricted site will permit little other than the receipt of mail via **uux**.

BUGS

Only the first command of a shell pipeline may have a 'system-name!'. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do.
The shell tokens << and >> are not implemented.
There is no notification of denial of execution on the remote machine.

NAME

vi — screen oriented (visual) display editor

SYNOPSIS

vi filename

DESCRIPTION

vi (visual) is a display oriented text editor. "An Introduction to Display Editing with Vi" in the *User Guide* describes how to use **vi**.

The line editor **ex**(1) is part of **vi**. **ex** and **vi** run the same code; it is possible to get to the command mode of **ex** from within **vi** and vice-versa. **vi** may put you into **ex** if it doesn't recognize your terminal setting in the shell environment. If this happens, simply type:

```
: set term=vt52 (or your terminal type)
: vi
```

and you will go into visual mode.

FILES

/etc/termcap	describes capabilities of terminals

SEE ALSO

ex (1), termcap(5), environ(5)

BUGS

vi takes about 10 seconds to fire up; it's not a small program.

Software tabs using **^T** work only immediately after the **autoindent**.

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The **wrapmargin** option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The **source** command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to **ex** command mode, execute them, and then reenter the screen editor with **vi** or **open**.

Entering/leaving vi

% **vi** *name*	edit *name* at top
% **vi** +*n name*	... at line *n*
% **vi** + *name*	... at end
% **vi** *name* **...**	edit first; rest via **:n**
% **vi** +*/pat name*	search for *pat*
% **view** *name*	read only mode

The display

Last line	Error messages, echoing input to **:** / **?** and **!**, feedback about i/o and large changes.
@ lines	On screen only, not in file.
~ lines	Lines past end of file.
^*x*	Control characters, ^? is delete.
tabs	Expand to spaces, cursor at last.

vi states

Command	Normal and initial state. Others return here. ESC (escape) cancels partial command.
Insert	Entered by **a i A I o O c C s S R**. Arbitrary text then terminates with ESC character, or abnormally with interrupt.
Last line	Reading input for **:** / **?** or **!**; terminate with ESC or CR to execute, interrupt to cancel.

Counts before vi commands

line/column number	**z G |**
scroll amount	**^D ^U**
replicate insert	**a i A I**
repeat effect	most rest

Simple commands

dw	delete a word
de	... leaving punctuation
dd	delete a line
3dd	... 3 lines
i*text*ESC	insert text *abc*
cw*new*ESC	change word to *new*
ea*s*ESC	pluralize word
xp	transpose characters

Interrupting, cancelling

ESC	end insert or incomplete cmd
^?	(delete or rubout) interrupts
^L	reprint screen if **^?** scrambles it

File manipulation

:w	write back changes
:wq	write and quit
:q	quit
:q!	quit, discard changes
:e *name*	edit file *name*
:e!	reedit, discard changes
:e + *name*	edit, starting at end
:e +*n*	edit starting at line *n*
:e #	edit alternate file
^↑	synonym for **:e #**
:w *name*	write file *name*
:w! *name*	overwrite file *name*
:sh	run shell, then return
:!*cmd*	run *cmd*, then return
:n	edit next file in arglist
:n *args*	specify new arglist
:f	show current file and line
^G	synonym for **:f**

Positioning within file

^F	forward screenful
^B	backward screenful
^D	scroll down half screen
^U	scroll up half screen
G	goto line (end default)
/pat	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/pat/ + *n*	n'th line after *pat*
?*pat*? − *n*	n'th line before *pat*
]]	next section/function
[[	previous section/function
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z −	... at bottom
z .	... at center
*/pat/***z** −	*pat* line at bottom
z*n* .	use *n* line window

Marking and returning

``	previous context
''	... at first non-white in line
m*x*	mark position with letter *x*
`*x*	to mark *x*
'*x*	... at first non-white in line

Line positioning

H	home window line
L	last window line
M	middle window line

+	next line, at first non-white
−	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

↑	first non white
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
^**H**	same as ←
space	same as →
f*x*	find *x* forward
F*x*	**f** backward
t*x*	upto *x* forward
T*x*	back upto *x*
;	repeat last **f F t** or **T**
,	inverse of ;
|	to specified column
%	find matching **(** **{** **)** or **}**

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back **W**
E	to end of **W**

Corrections during insert

^H	erase last character
^W	erases last word
erase	your erase, same as **^H**
kill	your kill, erase input this line
\	escapes **^H**, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R	replace characters

Operators (double to affect lines)

d	delete
c	change
<	left shift
>	right shift
!	filter through command
y	yank lines to buffer

Miscellaneous operations

C	change rest of line
D	delete rest of line
s	substitute chars
S	substitute lines

J	join lines
x	delete characters
X	... before cursor
Y	yank lines

Yank and put

p	put back lines
P	put before
"*x***p**	put from buffer *x*
"*x***y**	yank to buffer *x*
"*x***d**	delete into buffer *x*

Undo, redo, retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d***p**	retrieve *d*'th last delete

NAME

wait — await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the **wait**(2) system call must be executed in the parent process, the shell itself executes **wait**, without creating a new process.

SEE ALSO

sh(1)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus can't be waited for.

NAME

wall — write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

wall reads its standard input until an end-of-file. In other words, you type in the message to be distributed and then type CTRL – D. **wall** then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

'Cannot send to ...' when the open on a user's **tty** file fails.

NAME

wc — word count

SYNOPSIS

wc [**−lwc**] [name ...]

DESCRIPTION

wc counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters **l**, **w**, or **c**.

EXAMPLES

List the number of lines in file ''xyz.c'':

```
wc −l xyz.c
```

NAME

who — who is on the system

SYNOPSIS

who [who-file] [**am I**]

DESCRIPTION

who, without an argument, lists the login name, terminal name, and login time for each current VENIX user.

Without an argument, **who** examines the **/etc/utmp** file to obtain its information. This tells you who is on the system currently.

To find out who *was* on the system, **/usr/adm/wtmp** can be provided as the who-file in the command line. Read **ac**(1) to find out more about **/usr/adm/wtmp** which keeps track of all logins.

who lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with '/dev/' suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in the place of the device name, and a fossil time indicative of when the system went down.

'who am I' (and also 'who are you'), tells the name of the login user and the terminal in use.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(4)

NAME

write — write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message

Message from yourname yourttyname...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point **write** writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the **mesg**(1) command. At the outset writing is allowed. Certain commands, in particular **nroff** and **pr**(1), disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, **write** calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using **write**: when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a distinctive signal—**(o)** for 'over' is conventional—that the other may reply. **(oo)** for 'over and out' is suggested when conversation is about to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

SEE ALSO

mesg(1), who(1), mail(1), wall(1)

NAME

yacc — yet another compiler-compiler

SYNOPSIS

yacc [**−vd**] grammar

DESCRIPTION

yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user (defaults for the last two are described in the Yacc documentation). **lex**(1) is useful for creating lexical analyzers usable by **yacc**. The library compiler switch is −ly.

If the **−v** flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **−d** flag is used, the file *y.tab.h* is generated with the *define* statements that associate the **yacc**-assigned 'token codes' with the user-declared 'token names'. This allows source files other than *y.tab.c* to access the token codes.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex (1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
"Yet Another Compiler Compiler (yacc)" in the *Support Tools Guide.*

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one **yacc** process can be active in a given directory at a time.

Printed in U.S.A.

AA-BM36A-TH