UNIX PROGRAMMER'S MANUAL

Fifth Edition

K. Thompson

D. M. Ritchie

June, 1974

This manual was set by a Graphic Systems photo-
typesetter driven by the *troff* formatting program
operating under the UNIX system. The text of the
manual was prepared using the *ed* text editor.

PREFACE
to the Fifth Edition

The number of UNIX installations is now above 50, and many more are expected. None of these has exactly the same complement of hardware or software. Therefore, at any particular installation, it is quite possible that this manual will give inappropriate information.

INTRODUCTION TO THIS MANUAL

This manual gives descriptions of the publicly available features of UNIX. It provides neither a general overview (see "The UNIX Time-sharing System" for that) nor details of the implementation of the system (which remain to be disclosed).

Within the area it surveys, this manual attempts to be as complete and timely as possible. A conscious decision was made to describe each program in exactly the state it was in at the time its manual section was prepared. In particular, the desire to describe something as it should be, not as it is, was resisted. Inevitably, this means that many sections will soon be out of date.

This manual is divided into eight sections:

I. Commands
II. System calls
III. Subroutines
IV. Special files
V. File formats
VI. User-maintained programs
VII. Miscellaneous
VIII. Maintenance

Commands are programs intended to be invoked directly by the user, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in directory /bin (for binary programs). This directory is searched automatically by the command line interpreter. Some programs also reside in /usr/bin, to save space in /bin. Some programs classified as commands are located elsewhere; this fact is indicated in the appropriate sections.

System calls are entries into the UNIX supervisor. In assembly language, they are coded with the use of the opcode sys, a synonym for the trap instruction. In this edition, the C language interface routines to the system calls have been incorporated in section II.

A small assortment of subroutines is available; they are described in section III. The binary form of most of them is kept in the system library /lib/liba.a. The subroutines available from C and from Fortran are also included; they reside in /lib/libc.a and /lib/libf.a respectively.

The special files section IV discusses the characteristics of each system "file" which actually refers to an I/O device. The names in this section refer to the DEC device names for the hardware, instead of the names of the special files themselves.

The file formats section V documents the structure of particular kinds of files; for example, the form of the output of the loader and assembler is given. Excluded are files used by only one command, for example the assembler's intermediate files.

User-maintained programs (section VI) are not considered part of the UNIX system, and the principal reason for listing them is to indicate their existence without necessarily giving a complete description. The author should be consulted for information.

The miscellaneous section (VII) gathers odds and ends.

Section VIII discusses commands which are not intended for use by the ordinary user, in some cases because they disclose information in which he is presumably not interested, and in others because they perform privileged functions.

Each section consists of a number of independent entries of a page or so each. The name of the entry is in the upper corners of its pages, its preparation date in the upper middle. Entries within each section are alphabetized. The page numbers of each entry start at 1. (The earlier

hope for frequent, partial updates of the manual is clearly in vain, but in any event it is not feasible to maintain consecutive page numbering in a document like this.)

All entries are based on a common format, not all of whose subsections will always appear.

The *name* section repeats the entry name and gives a very short description of its purpose.

The *synopsis* summarizes the use of the program being described. A few conventions are used, particularly in the Commands section:

Boldface words are considered literals, and are typed just as they appear.

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 sign "—" is often taken to mean some sort of flag argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with "—".

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

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

A *see also* section gives pointers to related information.

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

The *bugs* section gives known bugs and sometimes deficiencies. Occasionally also the suggested fix is described.

At the beginning of this document is a table of contents, organized by section and alphabetically within each section. There is also a permuted index derived from the table of contents. Within each index entry, the title of the writeup to which it refers is followed by the appropriate section number in parentheses. This fact is important because there is considerable name duplication among the sections, arising principally from commands which exist only to exercise a particular system call.

This manual was prepared using the UNIX text editor *ed* and the formatting program *troff*.

HOW TO GET STARTED

This section provides the basic information you need to get started on UNIX: how to log in and log out, how to communicate through your terminal, and how to run a program.

Logging in. You must call UNIX from an appropriate terminal. UNIX supports ASCII terminals typified by the TTY 37, the GE Terminet 300, the Memorex 1240, and various graphical terminals. You must also have a valid user name, which may be obtained, together with the telephone number, from the system administrators. The same telephone number serves terminals operating at all the standard speeds. After a data connection is established, the login procedure depends on what kind of terminal you are using.

> *300-baud terminals:* Such terminals include the GE Terminet 300, most display terminals, Execuport, TI, and certain Anderson-Jacobson terminals. These terminals generally have a speed switch which should be set at "300" (or "30" for 30 characters per second) and a half/full duplex switch which should be set at full-duplex. (Note that this switch will often have to be changed since many other systems require half-duplex). When a connection is established, the system types "login:"; you type your user name, followed by the "return" key. If you have a password, the system asks for it and turns off the printer on the terminal so the password will not appear. After you have logged in, the "return", "new line", or "linefeed" keys will give exactly the same results.

> *TTY 37 terminal:* When you have established a data connection, the system types out a few garbage characters (the "login:" message at the wrong speed). Depress the "break" (or "interrupt") key; this is a speed-independent signal to UNIX that a 150-baud terminal is in use. The system then will type "login:," this time at the correct speed; you respond with your user name. From the TTY 37 terminal, and any other which has the "new-line" function (combined carriage return and linefeed), terminate each line you type with the "new-line" key (*not* the "return" key).

For all these terminals, it is important that you type your name in lower-case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and will translate all subsequent upper-case letters to lower case.

The evidence that you have successfully logged in is that the Shell program will type a "%" to you. (The Shell is described below under "How to run a program.")

For more information, consult *getty* (VIII), which discusses the login sequence in more detail, and *tty* (IV), which discusses typewriter I/O.

Logging out. There are three ways to log out:

> You can simply hang up the phone.

> You can log out by typing an end-of-file indication (EOT character, control "d") to the Shell. The Shell will terminate and the "login: " message will appear again.

> You can also log in directly as another user by giving a *login* command (I).

How to communicate through your terminal. When you type to UNIX, a gnome deep in the system is gathering your characters and saving them in a secret place. The characters will not be given to a program until you type a return (or new-line), as described above in *Logging in.*

UNIX typewriter I/O is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have the input characters interspersed. However, whatever you type will be saved up and interpreted in correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away all the saved characters.

On a typewriter input line, the character "@" kills all the characters typed before it, so typing mistakes can be repaired on a single line. Also, the character "#" erases the last character typed. Successive uses of "#" erase characters back to, but not beyond, the beginning of the line. "@" and "#" can be transmitted to a program by preceding them with "\". (So, to erase "\", you need two "#"s).

The ASCII "delete" (a.k.a. "rubout") character is not passed to programs but instead generates an *interrupt signal*. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor, for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The *quit* signal is generated by typing the ASCII FS character. It not only causes a running program to terminate but also generates a file with the core image of the terminated process. Quit is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent about whether you have a terminal with the new-line function or whether it must be simulated with carriage-return and line-feed. In the latter case, all input carriage returns are turned to new-line characters (the standard line delimiter) and both a carriage return and a line feed are echoed to the terminal. If you get into the wrong mode, the *stty* command (I) will rescue you.

Tab characters are used freely in UNIX source programs. If your terminal does not have the tab function, you can arrange to have them turned into spaces during output, and echoed as spaces during input. The system assumes that tabs are set every eight columns. Again, the *stty* command (I) will set or reset this mode. Also, there is a file which, if printed on TTY 37 or TermiNet 300 terminals, will set the tab stops correctly (*tabs* (VII)).

Section *tty* (IV) discusses typewriter I/O more fully. Section *kl* (IV) discusses the console typewriter.

How to run a program; The Shell. When you have successfully logged into UNIX, a program called the Shell is listening to your terminal. The Shell reads typed-in lines, splits them up into a command name and arguments, and executes the command. A command is simply an executable program. The Shell looks first in your current directory (see next section) for a program with the given name, and if none is there, then in a system directory. There is nothing special about system-provided commands except that they are kept in a directory where the Shell can find them.

The command name is always the first word on an input line; it and its arguments are separated from one another by spaces.

When a program terminates, the Shell will ordinarily regain control and type a "%" at you to indicate that it is ready for another command.

The Shell has many other capabilities, which are described in detail in section *sh*(I).

The current directory. UNIX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he also created a directory for you (ordinarily with the same name as your user name). When you log in, any file name you type is by default in this directory. Since you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their owners. As a matter of observed fact, few UNIX users protect their files from destruction, let alone perusal, by other users.

To change the current directory (but not the set of permissions you were endowed with at login) use *chdir* (I).

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with "/", the name of the root directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a "/") until finally the file name is reached. E.g.: */usr/lem/filex* refers to the file *filex* in the directory *lem;* *lem* is itself a subdirectory of *usr; usr* springs directly from the root directory.

If your current directory has subdirectories, the path names of files therein begin with the name of the subdirectory (no prefixed "/").

Without important exception, a path name may be used anywhere a file name is required.

Important commands which modify the contents of files are *cp* (I), *mv* (I), and *rm* (I), which respectively copy, move (i.e. rename) and remove files. To find out the status of files or directories, use *ls* (I). See *mkdir* (I) for making directories; *rmdir* (I) for destroying them.

For a fuller discussion of the file system, see "The UNIX Time-Sharing System," by the present authors, to appear in the Communications of the ACM; a version is also available from the same source as this manual. It may also be useful to glance through section II of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Writing a program. To enter the text of a source program into a UNIX file, use *ed* (I). The three principal languages in UNIX are assembly language (see *as* (I)), Fortran (see *fc* (I)), and C (see *cc* (I)). After the program text has been entered through the editor and written on a file, you can give the file to the appropriate language processor as an argument. The output of the language processor will be left on a file in the current directory named "a.out". (If the output is precious, use *mv* to move it to a less exposed name soon.) If you wrote in assembly language, you will probably need to load the program with library subroutines; see *ld* (I). The other two language processors call the loader automatically.

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the Shell in response to the "%" prompt.

The next command you will need is *db* (I). As a debugger, *db* is better than average for assembly-language programs, marginally useful for C programs (when completed, *cdb* (I) will be a boon), and virtually useless for Fortran.

Your programs can receive arguments from the command line just as system programs do. See *exec* (II).

Text processing. Almost all text is entered through the editor. The commands most often used to write text on a terminal are: *cat, pr, roff, nroff,* and *troff,* all in section I.

The *cat* command simply dumps ASCII text on the terminal, with no processing at all. The *pr* command paginates the text, supplies headings, and has a facility for multi-column output. *Troff* and *nroff* are elaborate text formatting programs, and require careful forethought in entering both the text and the formatting commands into the input file. *Troff* drives a Graphic Systems phototypesetter; it was used to produce this manual. *Nroff* produces output on a typewriter terminal. *Roff* (I) is a somewhat less elaborate text formatting program, and requires somewhat less forethought.

Surprises. Certain commands provide inter-user communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you.

To communicate with another user currently logged in, *write* (I) is used; *mail* (I) will leave a message whose presence will be announced to another user when he next logs in. The write-ups in the manual also suggest how to respond to the two commands if you are a target.

When you log in, a message-of-the-day may greet you before the first "%".

TABLE OF CONTENTS

I. COMMANDS

II. SYSTEM CALLS

III. SUBROUTINES

VII. MISCELLANEOUS

VIII. SYSTEM MAINTENANCE

xvii

NAME
        ar — archive and library maintainer

SYNOPSIS
        ar key afile name ...

DESCRIPTION
        Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the loader. It can be used, though, for any similar purpose.

        Key is one character from the set **drtux,** optionally concatenated with **v.** *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

        d means delete the named files from the archive file.

        r means replace the named files in the archive file. If the archive file does not exist, **r** will create it. If the named files are not in the archive file, they are appended.

        t prints 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.

        u is similar to **r** except that only those files that have been modified are replaced. If no names are given, all files in the archive that have been modified will be replaced by the modified version.

        x will 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 means 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. The following abbreviations are used:

                c copy
                a append
                d delete
                r replace
                x extract

FILES
        /tmp/vtm?   temporary

SEE ALSO
        ld (I), archive (V)

BUGS
        Option **tv** should be implemented as a table with more information.

        There should be a way to specify the placement of a new file in an archive. Currently, it is placed at the end.

        Since *ar* has not been rewritten to deal properly with the new file system modes, extracted files have mode 666.

        For the same reason, names are only maintained to 8 characters.

NAME

    as — assembler

SYNOPSIS

    as [—] name ...

DESCRIPTION

As assembles the concatenation of the named files. If the optional first argument — is used, all undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file **a.out.** It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

FILES

/etc/as2	pass 2 of the assembler
/tmp/atm[1-4]?	temporary
a.out	object

SEE ALSO

    ld(I), nm(I), db(I), a.out(V), 'UNIX Assembler Manual'.

DIAGNOSTICS

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

Symbol table overflow is not checked. x errors can cause incorrect line numbers in following diagnostics.

NAME
       cat — concatenate and print

SYNOPSIS
       cat file ...

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

              cat file

       is about the easiest way to print a file. Also:

              cat file1 file2 >file3

       is about the easiest way to concatenate files.

       If no input file is given *cat* reads from the standard input file.

       If the argument — is encountered, *cat* reads from the standard input file.

SEE ALSO
       pr(I), cp(I)

DIAGNOSTICS
       none; if a file cannot be found it is ignored.

BUGS
       cat x y >x and cat x y >y cause strange results.

NAME

cc — C compiler

SYNOPSIS

cc [−c] [−p] [−O] [−S] [−P] file ...

DESCRIPTION

Cc is the UNIX C compiler. It accepts three types of arguments:

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

The following flags are interpreted by *cc*. See *ld (I)* for load-time flags.

−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 the *monitor* subroutine (III) 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* (I).

−O   Invoke the experimental object-code optimizer.

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

−P   Run only the macro preprocessor on the named C programs, and leave the output on corresponding files suffixed '.i'.

Other arguments are taken to be either loader flag arguments, or C-compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out.**

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/lib/c[01]	compiler
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	runtime startoff for monitoring.
/lib/libc.a	builtin functions, etc.
/lib/liba.a	system library

SEE ALSO

"Programming in C— a tutorial," C Reference Manual, monitor (III), prof (I), cdb (I), ld (I).

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader. Of these, the most mystifying are from the assembler, in particular "m," which means a multiply-defined external symbol (function or data).

BUGS

NAME
cdb — C debugger

SYNOPSIS
cdb [core [a.out]]

DESCRIPTION
Cdb is a debugger for use with C programs. The first argument is a core-image file; if not given, "core" is used. The second argument is the object program (containing a symbol table); if not given "a.out" is used. An acceptable core and object file must both be present.

Commands to *cdb* consist of an address followed by a single command character. If no address is given the last-printed address is used. An address may be followed by a comma and a number, in which case the command applies to the appropriate number of successive addresses.

Addresses are expressions composed of names, decimal numbers, and octal numbers (which begin with "0") and separated by "+" and "−". Evaluation proceeds left-to-right. The construction "name[expression]" assumes that *name* is a pointer to an integer and is equivalent to the contents of the named cell plus twice the expression.

The command characters are:

/   print the addressed words in octal.

=   print the value of the addressed expression.

'   print the addressed bytes as characters.

"   take the address as a pointer to a sequence of characters, and print the characters up to a null byte.

&   If there is any symbol which has the same value as the address, print the symbol's name.

$ print a stack trace of the terminated program. The calls are listed in the order made; the actual arguments to each routine are given in octal.

SEE ALSO
cc (I), db (I), C Reference Manual

BUGS
It's still very feeble, even compared with *db (I)*. The stack trace is also pretty vulnerable to corruption, and often doesn't work.

NAME

chdir — change working directory

SYNOPSIS

chdir directory

DESCRIPTION

Directory becomes the new working directory. The process must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *chdir* would be ineffective if it were written as a normal command. It is therefore recognized and executed by the Shell.

SEE ALSO

sh(I)

BUGS

NAME

chmod — change mode

SYNOPSIS

chmod octal file ...

DESCRIPTION

The octal mode replaces the mode of each of the files. The mode is constructed from the OR of the following modes:

        4000 set user ID on execution
        2000 set group ID on execution
        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

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

SEE ALSO

ls(I)

BUGS

NAME

      chown — change owner

SYNOPSIS

      chown owner file ...

DESCRIPTION

      Owner becomes the new owner of the files. The owner may be either a decimal UID or a login name found in the password file.

      Only the owner of a file (or the super-user) is allowed to change the owner. Unless it is done by the super-user, the set-user-ID permission bit is turned off as the owner of a file is changed.

FILES

      /etc/passwd

BUGS

NAME
cmp — compare two files

SYNOPSIS
cmp file1 file2

DESCRIPTION
The two files are compared for identical contents. Discrepancies are noted by giving the offset and the differing words, all in octal.

SEE ALSO
diff (I), comm (I)

BUGS
If the shorter of the two files is of odd length, *cmp* acts as if a null byte had been appended to it. The *offset* is only a single-precision number.

NAME

    comm — print lines common to two files

SYNOPSIS

    comm [— [123]] file1 file2

DESCRIPTION

    Comm reads *file1* and *file2*, which should be in sort, and produces a three column output: lines only in *file1;* lines only in *file2;* and lines in both files.

    The output is written on the standard output.

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

SEE ALSO

    cmp (I), diff (I)

BUGS

NAME

cp — copy

SYNOPSIS

cp [—t] file1 file2

DESCRIPTION

The first file is copied onto the second. The mode and owner of the target file are preserved if it already existed; the mode of the source file is used otherwise.

If *file2* is a directory, then the target file is a file in that directory with the file-name of *file1*.

No one is quite sure what the flag —t does.

SEE ALSO

cat (I), pr (I), mv (I)

BUGS

Copying a file onto itself destroys its contents.

NAME
    cref — make cross reference listing

SYNOPSIS
    cref [—acilostux123] name ...

DESCRIPTION
    Cref makes a cross reference listing of program files in assembler or C format. The files named as arguments in the command line are searched for symbols in the appropriate syntax.

    The output report is in four columns:

    (1)   (2)   (3)   (4)
    symbol file   see   text as it appears in file
                       below

    Cref uses either an *ignore* file or an *only* file. If the —i option is given, it will take the next available argument to be an *ignore* file name; if the —o option is given, the next available argument will be taken as an *only* file name. *Ignore* and *only* files should be lists of symbols separated by new lines. If an *ignore* file is given, all the symbols in that file will be ignored in columns (1) and (3) of the output. If an *only* file is given, only symbols appearing in that file will appear in column (1). Only one of the options —i or —o may be used. The default setting is —i. Assembler predefined symbols or C keywords are ignored.

    The —s option causes current symbols to be put in column 3. In the assembler, the current symbol is the most recent name symbol; in C, the current function name. The —l option causes the line number within the file to be put in column 3.

    The —t option causes the next available argument to be used as the name of the intermediate temporary file (instead of /tmp/crt??). The file is created and is not removed at the end of the process.

    Options:

    a   assembler format (default)
    c   C format input
    i   use *ignore* file (see above)
    l   put line number in col. 3 (instead of current symbol)
    o   use *only* file (see above)
    s   current symbol in col. 3 (default)
    t   user supplied temporary file
    u   print only symbols that occur exactly once
    x   print only C external symbols
    1   sort output on column 1 (default)
    2   sort output on column 2
    3   sort output on column 3

FILES
    /tmp/crt??   temporaries
    /usr/lib/aign   default assembler *ignore* file
    /usr/lib/cign   default C *ignore* file
    /usr/bin/crpost   post processor
    /usr/bin/upost   post processor for —u option
    /bin/sort   used to sort temporaries

SEE ALSO
    as (I), cc (I)

BUGS

NAME

    date − print and set the date

SYNOPSIS

    date [mmddhhmm[yy]]

DESCRIPTION

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

        date 10080045

    sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

DIAGNOSTICS

    'bad conversion' if the argument is syntactically incorrect.

BUGS

NAME
    db — debug

SYNOPSIS
    db [core [namelist]] [—]

DESCRIPTION
Unlike many debugging packages (including DEC's ODT, on which *db* is loosely based), *db* is not loaded as part of the core image which it is used to examine; instead it examines files. Typically, the file will be either a core image produced after a fault or the binary output of the assembler. *Core* is the file being debugged; if omitted **core** is assumed. *Namelist* is a file containing a symbol table. If it is omitted, the symbol table is obtained from the file being debugged, or if not there from **a.out**. If no appropriate name list file can be found, *db* can still be used but some of its symbolic facilities become unavailable.

For the meaning of the optional third argument, see the last paragraph below.

The format for most *db* requests is an address followed by a one character command. Addresses are expressions built up as follows:

1. A name has the value assigned to it when the input file was assembled. It may be relocatable or not depending on the use of the name during the assembly.

2. An octal number is an absolute quantity with the appropriate value.

3. A decimal number immediately followed by '.' is an absolute quantity with the appropriate value.

4. An octal number immediately followed by **r** is a relocatable quantity with the appropriate value.

5. The symbol . indicates the current pointer of *db*. The current pointer is set by many *db* requests.

6. A * before an expression forms an expression whose value is the number in the word addressed by the first expression. A * alone is equivalent to '*.'.

7. Expressions separated by + or blank are expressions with value equal to the sum of the components. At most one of the components may be relocatable.

8. Expressions separated by — form an expression with value equal to the difference to the components. If the right component is relocatable, the left component must be relocatable.

9. Expressions are evaluated left to right.

Names for registers are built in:

    r0 ... r5
    sp
    pc
    fr0 ... fr5

These may be examined. Their values are deduced from the contents of the stack in a core image file. They are meaningless in a file that is not a core image.

If no address is given for a command, the current address (also specified by ".") is assumed. In general, "." points to the last word or byte printed by *db*.

There are *db* commands for examining locations interpreted as numbers, machine instructions, ASCII characters, and addresses. For numbers and characters, either bytes or words may be examined. The following commands are used to examine the specified file.

/   The addressed word is printed in octal.

\   The addressed byte is printed in octal.

" The addressed word is printed as two ASCII characters.

' The addressed byte is printed as an ASCII character.

` The addressed word is printed in decimal.

? The addressed word is interpreted as a machine instruction and a symbolic form of the instruction, including symbolic addresses, is printed. Often, the result will appear exactly as it was written in the source program.

& The addressed word is interpreted as a symbolic address and is printed as the name of the symbol whose value is closest to the addressed word, possibly followed by a signed offset.

<nl> (i. e., the character "new line") This command advances the current location counter "." and prints the resulting location in the mode last specified by one of the above requests.

^ This character decrements "." and prints the resulting location in the mode last selected one of the above requests. It is a converse to <nl>.

% Exit.

Odd addresses to word-oriented commands are rounded down. The incrementing and decrementing of "." done by the <nl> and ^ requests is by one or two depending on whether the last command was word or byte oriented.

The address portion of any of the above commands may be followed by a comma and then by an expression. In this case that number of sequential words or bytes specified by the expression is printed. "." is advanced so that it points at the last thing printed.

There are two commands to interpret the value of expressions.

= When preceded by an expression, the value of the expression is typed in octal. When not preceded by an expression, the value of "." is indicated. This command does not change the value of ".".

: An attempt is made to print the given expression as a symbolic address. If the expression is relocatable, that symbol is found whose value is nearest that of the expression, and the symbol is typed, followed by a sign and the appropriate offset. If the value of the expression is absolute, a symbol with exactly the indicated value is sought and printed if found; if no matching symbol is discovered, the octal value of the expression is given.

The following command may be used to patch the file being debugged.

! This command must be preceded by an expression. The value of the expression is stored at the location addressed by the current value of ".". The opcodes do not appear in the symbol table, so the user must assemble them by hand.

The following command is used after a fault has caused a core image file to be produced.

$ causes the fault type and the contents of the general registers and several other registers to be printed both in octal and symbolic format. The values are as they were at the time of the fault.

For some purposes, it is important to know how addresses typed by the user correspond with locations in the file being debugged. The mapping algorithm employed by *db* is non-trivial for two reasons: First, in an **a.out** file, there is a 20(8) byte header which will not appear when the file is loaded into core for execution. Therefore, apparent location 0 should correspond with actual file offset 20. Second, addresses in core images do not correspond with the addresses used by the program because in a core image there is a header containing the system's per-process data for the dumped process, and also because the stack is stored contiguously with the text and data part of the core image rather than at the highest possible locations. *Db* obeys the following rules:

If exactly one argument is given, and if it appears to be an **a.out** file, the 20-byte header is skipped during addressing, i.e., 20 is added to all addresses typed. As a consequence, the header can be examined beginning at location −20.

If exactly one argument is given and if the file does not appear to be an **a.out** file, no mapping is done.

If zero or two arguments are given, the mapping appropriate to a core image file is employed. This means that locations above the program break and below the stack effectively do not exist (and are not, in fact, recorded in the core file). Locations above the user's stack pointer are mapped, in looking at the core file, to the place where they are really stored. The per-process data kept by the system, which is stored in the first part of the core file, cannot currently be examined (except by $).

If one wants to examine a file which has an associated name list, but is not a core image file, the last argument "−" can be used (actually the only purpose of the last argument is to make the number of arguments not equal to two). This feature is used most frequently in examining the memory file /dev/mem.

SEE ALSO

as (I), core (V), a.out (V), od (I)

DIAGNOSTICS

"File not found" if the first argument cannot be read; otherwise "?".

BUGS

There should be some way to examine the registers and other per-process data in a core image; also there should be some way of specifying double-precision addresses. It does not know yet about shared text segments.

NAME
    dc — desk calculator

SYNOPSIS
    dc [file]

DESCRIPTION
    Dc is an arbitrary precision integer arithmetic package. The overall structure of *dc* is a stacking (reverse Polish) calculator. The following constructions are recognized by the calculator:

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.

+

—

*

%

^   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.

s*x*   The top of the stack is popped and stored into a register named *x*, where *x* may be any character.

l*x*   The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value.

d   The top value on the stack is duplicated.

p   The top value on the stack is printed. The top value remains unchanged.

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.

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

[...]   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.

!   interprets the rest of the line as a UNIX 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.

o   The top value on the stack is popped and used as the number radix for further output.

z   The stack level is pushed onto the stack.

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

new-line   ignored except as the name of a register or to end the response to a ?.

space   ignored except as the name of a register or to terminate a number.

If a file name is given, input is taken from that file until end-of-file, then input is taken from the console. An example which prints the first ten values of n! is

    [la1+dsa*pla10>x]sx
    0sa1
    lxx

FILES
        /etc/msh   to implement '!'

DIAGNOSTICS
        (x) ? for unrecognized character x.
        (x) ? for not enough elements on the stack to do what was asked by command x.
        '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.

BUGS

NAME
        dd — convert and copy a file

SYNOPSIS
        dd [option=value] ...

DESCRIPTION
        Dd copies its specified input file to its specified output with possible conversions. The input and output block size may be specified to take advantage of raw physical I/O.

option	values
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=	input block size (default 512)
obs=	output block size (default 512)
bs=	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
count=n	copy only n input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

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

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks 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.

For example, to read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x*:

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

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

SEE ALSO
        cp (I)

BUGS
        The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. It is not clear how this relates to real life.

NAME
        diff — differential file comparator

SYNOPSIS
        diff [—] name1 name2

DESCRIPTION
        Diff tells what lines must be changed in two files to bring them into agreement. The normal output contains lines of these forms:

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

These lines resemble *ed* commands to convert file *name1* into file *name2*. The numbers after the letters pertain to file *name2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert file *name2* into *name1*. 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 '.'.

Under the — option, the output of *diff* is a script of *a, c* and *d* commands for the editor *ed*, which will change the contents of the first file into the contents of the second. In this connection, 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.

> (cat $2 ... $9; echo "1,$p") | ed — $1

Diff does an optimal and unfailing job of detecting the file differences, and also reports these differences side-by-side. However, *diff* uses a quadratic algorithm that usually slows to a crawl on 2000-line files.

SEE ALSO
        cmp (I), comm (I), ed (I)

DIAGNOSTICS
        'can't open input'
        'arg count'
        'jackpot' — To speed things up, the program uses hashing. You have stumbled on a case where there is a minuscule chance that this has resulted in an unnecessarily long list of differences being published. It's a curio that we'd like to see.

BUGS
        Editing scripts produced under the — option are naive about creating lines consisting of a single '.'.

NAME

dsw — delete interactively

SYNOPSIS

dsw [directory]

DESCRIPTION

For each file in the given directory ('.' if not specified) *dsw* types its name. If **y** is typed, the file is deleted; if **x**, *dsw* exits; if new-line, the file is not deleted; if anything else, *dsw* asks again.

SEE ALSO

rm(I)

BUGS

The name *dsw* is a carryover from the ancient past. Its etymology is amusing.

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, '.' 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.

BUGS

  Non-directories given as arguments (not under —a option) are not listed.

  Removable file systems do not work correctly since i-numbers may be repeated while the corresponding files are distinct. *Du* should maintain an i-number list per root directory encountered.

NAME

    echo — echo arguments

SYNOPSIS

    echo [arg ...]

DESCRIPTION

    Echo writes all its arguments in order as a line on the standard output file. It is mainly useful for producing diagnostics in command files.

BUGS

NAME

ed — text editor

SYNOPSIS

ed [—] [name]

DESCRIPTION

Ed is the standard text editor.

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. The optional — simulates an os command (see below) which 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.* There is only one 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. Every command which requires addresses has default addresses, so that the addresses can often be omitted.

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

Ed supports a limited form of *regular expression* notation. A regular expression is an expression which specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. The regular expressions allowed by *ed* are constructed as follows:

1. An ordinary character (not one of those discussed below) is a regular expression and matches that character.

2. A circumflex '^' at the beginning of a regular expression matches the null character at the beginning of a line.

3. A currency symbol '$' at the end of a regular expression matches the null character at the end of a line.

4. A period '.' matches any character but a new-line character.

5. A regular expression followed by an asterisk '*' matches any number of adjacent occurrences (including zero) of the regular expression it follows.

6. A string of characters enclosed in square brackets '[ ]' matches any character in the string but no others. If, however, the first character of the string is a circumflex '^' the regular expression matches any character but new-line and the characters in the string.

7. The concatenation of regular expressions is a regular expression which matches the concatenation of the strings matched by the components of the regular expression.

8. The null regular expression standing alone is equivalent to 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.

Addresses are constructed as follows. 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 by each command is dis-

cussed under the description of the command.

1. The character '.' addresses the current line.

2. The character '^' addresses the line immediately before the current line.

3. The character '$' addresses the last line of the buffer.

4. A decimal number n addresses the n-th line of the buffer.

5. "x' addresses the line associated (marked) with the mark name character x which must be a printable character. Lines are marked with the k command described below.

6. A regular expression enclosed in slashes '/' addresses the first line found by searching toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the beginning of the buffer.

7. A regular expression enclosed in queries '?' addresses the first line found by searching toward the beginning of the buffer and stopping at the first line found containing a string matching the regular expression. If necessary the search wraps around to the end of the buffer.

8. 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.

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, any command may be suffixed by 'p' (for 'print'). In that case, the current line is printed after the command is complete.

> (.)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 first line not changed.

> (.,.) 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.

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 (global) 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; if there were none, at the addressed line. This command differs from the *a* command only in the placement of the text.

(.)k*x*

The mark command associates or marks the addressed line with the single character mark name *x*. The ten most recent mark names are remembered. The current mark names may be printed with the *n* command.

(.,.)l

The list command will print the addressed lines in a way that is unambiguous. Non-graphic characters are printed in octal, prefixed characters are overstruck with a circumflex, and long lines are folded.

(.,.)m*a*

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

n

The *n* command will print the current mark names.

os

ov

After *os* character counts printed by *e, r,* and *w* are suppressed. *Ov* turns them back on.

(.,.)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 command.

q

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

($)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 remembered file name is not changed unless 'filename' is the very first file name mentioned. 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 regular expression that was matched. The special meaning of '&' in this context may be suppressed by preceding it by '\'.

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

> This command is the same as the global command except that the command list is executed 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 writeable by everyone). The remembered file name is *not* changed unless 'filename' is the very first file name mentioned. 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 typed.

($)=

> The line number of the addressed line is typed. '.' is unchanged by this command.

!UNIX command

> The remainder of the line after the '!' is sent to UNIX to be interpreted as a command. '.' is unchanged. The entire shell syntax is not recognized. See msh(VII) for the restrictions.

(.+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 (ASCII DEL) is sent, *ed* will print a '?' and return to its command level.

If invoked with the command name '—', (see init(VII)) *ed* will sign on with the message 'Editing system' and print '*' as the command level prompt character.

Ed has size limitations on the maximum number of lines that can be edited, on the maximum number of characters in a line, in a global's command list, in a remembered file name, and in the size of the temporary file. The current sizes are: 4000 lines per file, 512 characters per line, 256 characters per global command list, 64 characters per file name, and 64K characters in the temporary file (see BUGS).

FILES

> /tmp/etm?, temporary
> /etc/msh, to implement the '!' command.

DIAGNOSTICS

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

SEE ALSO

> A Tutorial Introduction to the ED Text Editor (internal memorandum)

BUGS

> The temporary file can grow to no more than 64K bytes.

NAME

eqn — typeset mathematics

SYNOPSIS

eqn [file] ...

DESCRIPTION

Eqn is a troff (I) preprocessor for typesetting mathematics on the Graphics Systems photo-typesetter. Usage is almost always

eqn file ... | troff

If no files are specified, *eqn* 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 or defined by *eqn,* so you can define them yourself to get centering, numbering, etc. All other lines are treated as comments, and passed through untouched.

Spaces, tabs, newlines, braces, double quotes, tilde and circumflex are the only delimiters. Braces "{}" are used for grouping. Use tildes "~" to get extra spaces in an equation.

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 2}* gives $e^{x^2+y^2}$. Fractions are made with **over**. *a over b* is $\frac{a}{b}$ and *1 over sqrt {ax sup 2 +bx+c}* is $\frac{1}{\sqrt{ax^2+bx+c}}$. **sqrt** makes square roots.

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.

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.

Diacritical marks are made with **dot, dotdot, hat, bar**: *x dot = f(t) bar* is $\dot{x}=\overline{f(t)}$. Default sizes and fonts can be changed with **size n** and various of **roman, italic,** and **bold**.

Keywords like *sum* ($\sum$) *int* ($\int$) *inf* (∞) and shorthands like >= ($\geq$) -> ($\rightarrow$), != ($\neq$), are recognized. Spell out Greek letters in the desired case, as in *alpha, GAMMA*. Mathematical words like sin, cos, log are made Roman automatically. Troff (I) four-character escapes like \(bs (⊚) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched.

SEE ALSO

A System for Typesetting Mathematics (Computer Science Technical Report #17, Bell Laboratories, 1974.)
TROFF Users' Manual (internal memorandum)
TROFF Made Trivial (internal memorandum)
troff (I), neqn (I)

BUGS

Undoubtedly. Watch out for small or large point sizes — it's tuned too well for size 10. Be cautious if inserting horizontal or vertical motions, and of backslashes in general.

- 1 -

NAME
        exit — terminate command file

SYNOPSIS
        exit

DESCRIPTION
        Exit performs a seek to the end of its standard input file. Thus, if it is invoked inside a file of commands, upon return from *exit* the shell will discover an end-of-file and terminate.

SEE ALSO
        if (I), goto (I), sh (I)

BUGS

NAME
>   fc — fortran compiler

SYNOPSIS
>   fc [−c] sfile1.f ... ofile1 ...

DESCRIPTION
>   *Fc* is the UNIX Fortran compiler. It accepts three types of arguments:

Arguments whose names end with '.f' are assumed to be Fortran source program units; they are compiled, and the object program is left on the file sfile1.o (i.e. the file whose name is that of the source with '.o' substituted for '.f').

Other arguments (except for −c) are assumed to be either loader flags, or object programs, typically produced by an earlier *fc* run, or perhaps libraries of Fortran-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.**

The −c argument suppresses the loading phase, as does any syntax error in any of the routines being compiled.

The following is a list of differences between *fc* and ANSI standard Fortran (also see the BUGS section):

1.   Arbitrary combination of types is allowed in expressions. Not all combinations are expected to be supported at runtime. All of the normal conversions involving integer, real, double precision and complex are allowed.

2.   DEC's **implicit** statement is recognized. E.g.: **implicit integer /i—n/**

3.   The types doublecomplex, logical*1, integer*1, integer*2 and real*8 (double precision) are supported.

4.   **&** as the first character of a line signals a continuation card.

5.   **c** as the first character of a line signals a comment.

6.   All keywords are recognized in lower case.

7.   The notion of 'column 7' is not implemented.

8.   G-format input is free form— leading blanks are ignored, the first blank after the start of the number terminates the field.

9.   A comma in any numeric or logical input field terminates the field.

10.   There is no carriage control on output.

11.   A sequence of *n* characters in double quotes '"' is equivalent to *n* **h** followed by those characters.

12.   In **data** statements, a hollerith string may initialize an array or a sequence of array elements.

13.   The number of storage units requested by a binary **read** must be identical to the number contained in the record being read.

14.   If the first character in an input file is "#", a preprocessor identical to the C preprocessor is called, which implements "#define" and "#include" preprocessor statements. (See the C reference manual for details.) The preprocessor does not recognize Hollerith strings written with *n* **h**.

In I/O statements, only unit numbers 0-19 are supported. Unit number *n* refers to file fort*nn;* (e.g. unit 9 is file 'fort09'). For input, the file must exist; for output, it will be created. Unit 5 is permanently associated with the standard input file; unit 6 with the standard output file. Also see *setfil* (III) for a way to associate unit numbers with named files.

FILES

file.f	input file
a.out	loaded output
f.tmp[123]	temporary (deleted)
/usr/fort/fc1	compiler proper
/lib/fr0.o	runtime startoff
/lib/filib.a	interpreter library
/lib/libf.a	builtin functions, etc.
/lib/liba.a	system library

SEE ALSO

ANSI standard, ld (I) for loader flags

Also see the writeups on the precious few non-standard Fortran subroutines, ierror and setfil (III)

DIAGNOSTICS

Compile-time diagnostics are given in English, accompanied if possible with the offending line number and source line with an underscore where the error occurred. Runtime diagnostics are given by number as follows:

1   invalid log argument
2   bad arg count to amod
3   bad arg count to atan2
4   excessive argument to cabs
5   exp too large in cexp
6   bad arg count to cmplx
7   bad arg count to dim
8   excessive argument to exp
9   bad arg count to idim
10   bad arg count to isign
11   bad arg count to mod
12   bad arg count to sign
13   illegal argument to sqrt
14   assigned/computed goto out of range
15   subscript out of range
16   real**real overflow
17   (negative real)**real
100 illegal I/O unit number
101 inconsistent use of I/O unit
102 cannot create output file
103 cannot open input file
104 EOF on input file
105 illegal character in format
106 format does not begin with (
107 no conversion in format but non-empty list
108 excessive parenthesis depth in format
109 illegal format specification
110 illegal character in input field
111 end of format in hollerith specification
112 bad argument to setfil
120 bad argument to ierror
999 unimplemented input conversion

Any of these errors can be caught by the program; see *ierror* (III).

BUGS

The following is a list of those features not yet implemented:
arithmetic statement functions
scale factors on input
Backspace statement.

NAME
     fed — edit associative memory for form letter

SYNOPSIS
     fed

DESCRIPTION
     Fed is used to edit a form letter associative memory file, **form.m,** which consists of named
     strings. Commands consist of single letters followed by a list of string names separated by a sin-
     gle space and ending with a new line. The conventions of the Shell with respect to '*' and '?'
     hold for all commands but **m.** The commands are:

     e name ...
          Fed writes the string whose name is *name* onto a temporary file and executes *ed.* On exit
          from the *ed* the temporary file is copied back into the associative memory. Each argument is
          operated on separately. Be sure to give an *ed w* command (without a filename) to rewrite *fed's*
          temporary file before quitting out of *ed.*

     d [name ...]
          deletes a string and its name from the memory. When called with no arguments **d** operates
          in a verbose mode typing each string name and deleting only if a **y** is typed. A **q** response re-
          turns to *fed's* command level. Any other response does nothing.

     m name1 name2 ...
          (move) changes the name of name1 to name2 and removes previous string name2 if one ex-
          ists. Several pairs of arguments may be given. Literal strings are expected for the names.

     n [name ...]
          (names) lists the string names in the memory. If called with the optional arguments, it just
          lists those requested.

     p name ...
          prints the contents of the strings with names given by the arguments.

     q
          returns to the system.

     c [p] [f]
          checks the associative memory file for consistency and reports the number of free headers
          and blocks. The optional arguments do the following:

          p   causes any unaccounted-for string to be printed.

          f   fixes broken memories by adding unaccounted-for headers to free storage and removing
               references to released headers from associative memory.

FILES
     /tmp/ftmp?   temporary
     form.m   associative memory

SEE ALSO
     form(I), ed(I), sh(I)

WARNING
     It is legal but unwise to have string names with blanks, ':' or '?' in them.

BUGS

NAME
 find — find files

SYNOPSIS
 find pathname expression

DESCRIPTION
 Find recursively descends 'the directory hierarchy from *pathname* 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(I)). If *onum* is prefixed by a minus sign, more flag bits (017777, see stat(II)) become significant and the flags will be 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*.

—group gname   As it is for —user so shall it be for —group (someday).

—size n   True if the file is *n* blocks long (512 bytes per block).

—atime n   True if the file has been accessed in *n* days.

—mtime n   True if the file has been modified in *n* days.

—exec command   True if the executed command returns exit status zero (most commands do). The end of the command is punctuated by an escaped semicolon. A command argument '{}' is replaced by the current pathname.

—ok command   Like —exec except that the generated command line is printed with a question mark first, and is executed only if the user responds y.

—print   Always true; causes the current pathname to be printed.

The primaries may be combined with these operators (ordered by precedence):

!   prefix *not*

—a   infix *and,* second operand evaluated only if first is true

—o   infix *or,* second operand evaluated only if first is false

(expression)   parentheses for grouping. (Must be escaped.)

To remove files named 'a.out' and '*.o' not accessed for a week:

     find / "(" —name a.out —o —name "*.o" ")" —a —atime +7 —a —exec rm {} ";"

FILES
 /etc/passwd

SEE ALSO
 sh (I), if(I), file system (V)

BUGS
 There is no way to check device type.
 Syntax should be reconciled with *if.*

NAME
    form — form letter generator

SYNOPSIS
    form proto arg ...

DESCRIPTION
    Form generates a form letter from a prototype letter, an associative memory, arguments and in a special case, the current date.

    If *form* is invoked with the *proto* argument *x*, the associative memory is searched for an entry with name *x* and the contents filed under that name are used as the prototype. If the search fails, the message '[x]:' is typed on the console and whatever text is typed in from the console, terminated by two new lines, is used as the prototype. If the prototype argument is missing, '{letter}' is assumed.

    Basically, *form* is a copy process from the prototype to the output file. If an element of the form [n] (where *n* is a digit from 1 to 9) is encountered, the *n*-th argument is inserted in its place, and that argument is then rescanned. If [0] is encountered, the current date is inserted. If the desired argument has not been given, a message of the form '[n]:' is typed. The response typed in then is used for that argument.

    If an element of the form [name] or {name} is encountered, the *name* is looked up in the associative memory. If it is found, the contents of the memory under this *name* replaces the original element (again rescanned). If the *name* is not found, a message of the form '[name]:' is typed. The response typed in is used for that element. The response is entered in the memory under the name if the name is enclosed in [ ]. The response is not entered in the memory but is remembered for the duration of the letter if the name is enclosed in {}.

    In both of the above cases, the response is typed in by entering arbitrary text terminated by two new lines. Only the first of the two new lines is passed with the text.

    If one of the special characters [{]}\ is preceded by a \, it loses its special character.

    If a file named 'forma' already exists in the user's directory, 'formb' is used as the output file and so forth to 'formz'.

    The file 'form.m' is created if none exists. Because form.m is operated on by the disc allocator, it should only be changed by using *fed*, the form letter editor, or *form*.

FILES
    form.m associative memory
    form?   output file (read only)

SEE ALSO
    fed (I), roff (I)

BUGS
    An unbalanced] or } acts as an end of file but may add a few strange entries to the associative memory.

NAME

goto — command transfer

SYNOPSIS

goto label

DESCRIPTION

Goto is only allowed when the Shell is taking commands from a file. The file is searched from the beginning for a line beginning with ':' followed by one or more spaces followed by the *label*. If such a line is found, the *goto* command returns. Since the read pointer in the command file points to the line after the label, the effect is to cause the Shell to transfer to the labelled line.

':' is a do-nothing command that is ignored by the Shell and only serves to place a label.

SEE ALSO

sh(I)

BUGS

NAME
        grep — search a file for a pattern

SYNOPSIS
        grep [−v] [−b] [−c] [−n] expression [file] ...

DESCRIPTION
        Grep will search the input files (standard input default) for each line containing the regular expression. Normally, each line found is printed on the standard output. If the −v flag is used, all lines but those matching are printed. If the −c flag is used, each line printed is preceded by its line number. If the −n flag is used, each line is preceded by the name of the input file and its relative line number in that file. If the −b flag is used, each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context. If interrupt is hit, the file and line number last searched is printed.

        For a complete description of the regular expression, see ed (I). Care should be taken when using the characters $ * [ˆ | ( ) and \ in the regular expression as they are also meaningful to the Shell. It is generally necessary to enclose the entire *expression* argument in quotes.

SEE ALSO
        ed (I), sh (I)

BUGS
        Lines are limited to 512 characters; longer lines are truncated.

NAME
          if — conditional command

SYNOPSIS
          if expr command [arg ...]

DESCRIPTION
          If evaluates the expression *expr*, and if its value is true, executes the given *command* with the given arguments.

          The following primitives are used to construct the *expr*:

          —r file   true if the file exists and is readable.

          —w file   true if the file exists and is writable.

          s1 = s2   true if the strings *s1* and *s2* are equal.

          s1 != s2   true if the strings *s1* and *s2* are not equal.

          (command)   The bracketed command is executed to obtain the exit status. Status zero is considered *true*. The command must not be another *if*.

          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 *if* and hence must be surrounded by spaces. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO
          sh (I), find (I)

BUGS

NAME
         kill — do in an unwanted process

SYNOPSIS
         kill —[signo] processid ...

DESCRIPTION
         Kills the specified processes. The processid of each asynchronous process started with '&' is re-
         ported by the shell. Processid's can also be found by using *ps* (I).

         The killed process must have been started from the same typewriter as the current user, unless
         he is the super-user.

         If a signal number preceded by "—" is given as an argument, that signal is sent instead of *kill* (see
         signal (II)).

SEE ALSO
         ps (I), sh (I), signal (II)

BUGS
         Clearly people should only be allowed to kill processes owned by them, and having the same
         typewriter is neither necessary nor sufficient.

NAME

     ld — link editor

SYNOPSIS

     ld [—sulxrnd] name ...

DESCRIPTION

     Ld combines several object programs into one; resolves external references; and searches libraries. In the simplest case the names of several object programs are given, and *d* 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 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, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

     Ld understands several flag arguments which are written preceded by a '—'. Except for —l, they should appear before the file names.

—s   'squash' 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.*

—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   This option is an abbreviation for a library name. —l alone stands for '/lib/liba.a', which is the standard system library for assembly language programs. —lx stands for '/lib/libx.a' where *x* is any character. A library is searched when its name is encountered, so the placement of a —l is significant.

—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.

—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.

—d   force definition of common storage even if the —r flag is present (used for reloc (VIII)).

—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 the the first possible 4K word boundary following the end of the text.

FILES

     /lib/lib?.a   libraries
     a.out   output file

SEE ALSO

     as(I), ar(I)

BUGS

NAME
    ln — make a link

SYNOPSIS
    ln name1 [name2]

DESCRIPTION
A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc) may have several links to it. There is no way to distinguish a link to a file from its original directory entry; any changes in the file are effective independently of the name by which the file is known.

Ln creates a link to an 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*.

It is forbidden to link to a directory or to link across file systems.

SEE ALSO
    rm(I)

BUGS
There is nothing particularly wrong with *ln*, but *tp* doesn't understand about links and makes one copy for each name by which a file is known; thus if the tape is extracted several copies are restored and the information that links were involved is lost.

NAME

login — sign onto UNIX

SYNOPSIS

login [username]

DESCRIPTION

The *login* command is used when a user initially signs onto UNIX, 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 'How to Get Started' for how to dial up initially.

If *login* is invoked without an argument, it will ask for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of *mailbox* and message-of-the-day files.

Login is recognized by the Shell and executed directly (without forking).

FILES

/tmp/utmp   accounting
/usr/adm/wtmp accounting
mailbox   mail
/etc/motd   message-of-the-day
/etc/passwd   password file

SEE ALSO

init (VIII), getty (VIII), mail (I)

DIAGNOSTICS

'login incorrect,' if the name or the password is bad. 'No Shell,', 'cannot open password file,' 'no directory': consult a UNIX programming counselor.

BUGS

If the first login is unsuccessful, it tends to go into a state where it won't accept a correct login. Hit EOT and try again.

NAME

      lpr — on line print

SYNOPSIS

      lpr [—] [+] [+—]file ...

DESCRIPTION

      Lpr arranges to have the line printer daemon print the file arguments.

      Normally, each file is printed in the state it is found when the line printer daemon reads it. If a particular file argument is preceded by **+**, or a preceding argument of **+** has been encountered, then *lpr* will make a copy for the daemon to print. If the file argument is preceded by —, or a preceding argument of — has been encountered, then *lpr* will unlink (remove) the file.

      If there are no arguments, then the standard input is read and on-line printed. Thus *lpr* may be used as a filter.

FILES

      /usr/lpd/*   spool area
      /etc/passwd   personal ident cards
      /etc/lpd   daemon

SEE ALSO

      lpd (VIII), passwd (V)

BUGS

NAME

ls — list contents of directory

SYNOPSIS

ls [—ltasdruif] 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. 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; usually those beginning with '.' are suppressed

—s give size in 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)

—i print i-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.

The mode printed under the —l option contains 10 characters which are interpreted as follows: the first character is

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;
—   if the entry is a plain 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

Finally, 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.

FILES

/etc/passwd to get user ID's for ls —l.

BUGS

NAME

mail — send mail to another user

SYNOPSIS

mail [**−yn**]
mail letter person ...
mail person

DESCRIPTION

Mail without an argument searches for a file called *mailbox,* prints it if present, and asks if it should be saved. If the answer is **y,** the mail is renamed *mbox,* otherwise it is deleted. *Mail* with a **−y** or **−n** argument works the same way, except that the answer to the question is supplied by the argument.

When followed by the names of a letter and one or more people, the letter is prepended to each person's *mailbox.* When a *person* is specified without a *letter,* the letter is taken from the sender's standard input up to an EOT. Each letter is preceded by the sender's name and a postmark.

A *person* is either a user name recognized by **login,** in which case the mail is sent to the default working directory of that user, or the path name of a directory, in which case *mailbox* in that directory is used.

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

FILES

/etc/passwd	to identify sender and locate persons
mailbox	input mail
mbox	saved mail
/tmp/mtm?	temp file

SEE ALSO

login (I)

BUGS

NAME
        man — run off section of UNIX manual

SYNOPSIS
        man [section] [title ...]

DESCRIPTION
        Man is a shell command file that will locate and run off one or more sections of this manual.
        Section is the section number of the manual, as an Arabic not Roman numeral, and is optional.
        Title is one or more section names; these names bear a generally simple relation to the page cap-
        tions in the manual. If the *section* is missing, **1** is assumed. For example,

            man man

        would reproduce this page.

FILES
        /usr/man/man?/*

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

NAME
       mesg — permit or deny messages

SYNOPSIS
       mesg [n] [y]

DESCRIPTION
       Mesg with argument **n** forbids messages via *write* by revoking non-user write permission on the
       user's typewriter. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reverses the
       current permission. In all cases the previous state is reported.

FILES
       /dev/tty?

SEE ALSO
       write (I)

DIAGNOSTICS
       '?' if the standard input file is not a typewriter

BUGS

NAME

      mkdir — make a directory

SYNOPSIS

      mkdir dirname ...

DESCRIPTION

      Mkdir creates specified directories in mode 777. The standard entries '.' and '..' are made automatically.

SEE ALSO

      rmdir(I)

BUGS

NAME
     mv − move or rename a file

SYNOPSIS
     mv name1 name2

DESCRIPTION
     Mv changes the name of *name1* to *name2*. If *name2* is a directory, *name1* is moved to that direc-
     tory with its original file-name. Directories may only be moved within the same parent directory
     (just renamed).

     If *name2* already exists, it is removed before *name1* is renamed. If *name2* has a mode which for-
     bids writing, *mv* prints the mode and reads the standard input to obtain a line; if the line begins
     with **y**, the move takes place; if not, *mv* exits.

     If *name2* would lie on a different file system, so that a simple rename is impossible, *mv* copies the
     file and deletes the original.

BUGS
     It should take a −**f** flag, like *rm,* to suppress the question if the target exists and is not writable.

NAME
        neqn — typeset mathematics on terminal

SYNOPSIS
        neqn [file] ...

DESCRIPTION
        Neqn is an nroff (I) preprocessor. The input language is the same as that of eqn (I). Normal
        usage is almost always

                neqn file ... | nroff

        Output is meant for terminals with forward and reverse capabilities, such as the Model 37 tele-
        type or GSI terminal.

        If no arguments are specified, *neqn* reads the standard input, so it may be used as a filter.

SEE ALSO
        eqn (I), gsi (VI)

BUGS

        Because of some interactions with *nroff* there may not always be enough space left before and
        after lines containing equations.

NAME

nice — run a command at low priority

SYNOPSIS

nice command [arguments]

DESCRIPTION

Nice executes *command* at low priority.

SEE ALSO

nohup(I), nice(II)

BUGS

NAME

nm — print name list

SYNOPSIS

nm [**−cnru**] [name]

DESCRIPTION

Nm prints the symbol table from the output file of an assembler or loader run. 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.

If no file is given, the symbols in **a.out** are listed.

Options are:

−c list only C-style external symbols, that is those beginning with underscore '_'.

−n sort by value instead of by name

−r sort in reverse order

−u print only undefined symbols.

FILES

a.out

BUGS

NAME
        nohup — run a command immune to hangups

SYNOPSIS
        nohup command [arguments]

DESCRIPTION
        Nohup executes *command* with hangups, quits and interrupts all ignored.

SEE ALSO
        nice(I), signal(II)

BUGS

NAME
         nroff − format text

SYNOPSIS
         nroff [+*n*] [−*n*] [−n*n*] [−m*x*] [−s] [−h] [−q] [−i] files

DESCRIPTION
         Nroff formats text according to control lines embedded in the text files. *Nroff* will read the standard input if no file arguments are given. The non-file option arguments are interpreted as follows:

         +*n*   Output will commence at the first page whose page number is *n* or larger.

         −*n*   will cause printing to stop after page *n*.

         −n*n*   First generated (not necessarily printed) page is given number *n*; simulates ".pn *n*".

         −m*x*   Prepends a standard macro file; simulates ".so /usr/lib/tmac.*x*".

         −s   Stop prior to each page to permit paper loading. Printing is restarted by typing a 'newline' character.

         −h   Spaces are replaced where possible with tabs to speed up output (or reduce the size of the output file).

         −q   Prompt names for insertions are not printed and the bell character is sent instead; the insertion is not echoed.

         −i   Causes the standard input to be read after the files.

FILES
         /usr/lib/suftab   suffix hyphenation tables
         /tmp/rtm?   temporary
         /usr/lib/tmac.?   standard macro files

SEE ALSO
         NROFF User's Manual (internal memorandum).
         neqn (I)

BUGS

REQUEST REFERENCE AND INDEX

Request Form	Initial Value	If no Argument	Cause Break	Explanation
I. Page Control				
.pl +N	N=66	N=66	no	Page length.
.bp +N	N=1	-	yes	Begin page.
.pn +N	N=1	ignored	no	Page number.
.po +N	N=0	N=prev	no	Page offset.
.ne N	-	N=1	no	Need N lines.
.mk	none	-	no	Mark current line.
.rt	-	-	no	Return to marked line.
II. Text Filling, Adjusting, and Centering				
.br	-	-	yes	Break.
.fi	fill	-	yes	Fill output lines.
.nf	fill	-	yes	No filling and adjusting.
.ad c	adj,norm	adjust	no	Adjust mode on.
.na	adjust	-	no	No adjusting.
.ce N	off	N=1	yes	Center N input text lines.
III. Line Spacing and Blank Lines				
.ls +N	N=1	N=prev	no	Line spacing.
.sp N	-	N=1	yes	Space N lines
.lv N	-	N=1	no	Save N lines
.sv N	-	N=1	no	"
.os	-	-	no	Output saved lines.
.ns	space	-	no	No-space mode on.
.rs	-	-	no	Restore spacing.
.xh	off	-	no	Extra-half-line mode on.
IV. Line Length and Indenting				
.ll +N	N=65	N=prev	no	Line length.
.in +N	N=0	N=prev	yes	Indent.
.ti +N	-	N=1	yes	Temporary indent.
V. Macros, Diversion, and Line Traps				
.de xx	-	ignored	no	Define or redefine a macro.
.am xx	-	ignored	no	Append to a macro.
.ds xx	-	ignored	no	Define or redefine string.
.as xx	-	ignored	no	Append to a string.
.rm xx	-	-	no	Remove macro name.
.di xx	-	end	no	Divert output to macro "xx".
.da xx	-	end	no	Divert and append to "xx".
.wh -N xx	-	-	no	When; set a line trap.
.ch -N -M	-	-	no	Change trap line.
.ch xx -M	-	-	no	"
VI. Number Registers				
.nr ab +N -M	-		no	Number register.
.nr a +N -M	-		no	"
.nc c	\n	\n	no	Number character.
.ar	arabic	-	no	Arabic numbers.
.ro	arabic	-	no	Lower case roman numbers.
.RO	arabic	-	no	Upper case roman numbers.

VII. Input and Output Conventions and Character Translations

.ta N,M,...	-	none	no	Pseudotabs setting.
.tc c	space	space	no	Tab replacement character.
.lc c	.	.	no	Leader replacement character.
.ul n	-	N=1	no	Underline input text lines.
.cc c	.	.	no	Basic control character.
.c2 c	'	'	no	Nobreak control character.
.ec c	-	\	no	Escape character.
.li N	-	N=1	no	Accept input lines literally.
.tr abcd....	-	-	no	Translate on output.

VIII. Hyphenation.

.nh	on	-	no	No hyphenation.
.hy	on	-	no	Hyphenate.
.hc c	none	none	no	Hyphenation indicator character.

IX. Three Part Titles.

.tl 'left'center'right'	-		no	Title.
.lt N	N=65	N=prev	no	Length of title.

X. Output Line Numbering.

.nm +N M S I		off	no	Number mode on or off, set parameters.
.np M S I	-	reset	no	Number parameters set or reset.

XI. Conditional Input Line Acceptance

.if !N anything	-		no	If true accept line of "anything".
.if c anything	-		no	"
.if !c anything	-		no	"
.if N anything	-		no	"

XII. Environment Switching.

.ev N	N=0	N=prev	no	Environment switched (pushed down).

XIII. Insertions from the Standard Input Stream

.rd prompt	- . '	bell	no	Read insert.
.ex	-	-	no	Exit.

XIV.   Input File Switching

.so filename	-	-	no	Switch source file (push down).
.nx filename		-	no	Next file.

XV.   Miscellaneous

.tm mesg	-	-	no	Typewriter message
.ig	-	-	no	Ignore.
.fl	-	-	yes	Flush output buffer.
.ab	-	-	no	Abort.

NAME
    od — octal dump

SYNOPSIS
    od [—abcdho] [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:

    a   interprets words as PDP-11 instructions and dis-assembles the operation code. Unknown operation codes print as ???.

    b   interprets bytes in octal.

    c   interprets bytes in ascii. Unknown ascii characters are printed as \?.

    d   interprets words in decimal.

    h   interprets words in hex.

    o   interprets words in octal.

    The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used. Thus *od* can be used as a filter.

    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. (A block is 512 bytes.) If the file argument is omitted, the offset argument must be preceded by '+'.

    Dumping continues until end-of-file.

SEE ALSO
    db (I)

BUGS

NAME

opr — off line print

SYNOPSIS

opr [——[*id*]] [—] [+] [+—]file ...

DESCRIPTION

Opr arranges to have the 201, data phone daemon submit a job to the Honeywell 6070 to print the file arguments. Normally, the output appears at the GCOS central site. If the first argument is ——, followed by an optional two-character remote station ID, the output is remoted to that station. If no station ID is given, R1 is assumed. (Station R1 has an IBM 1403 printer.)

Normally, each file is printed in the state it is found when the data phone daemon reads it. If a particular file argument is preceded by +, or a preceding argument of + has been encountered, then *opr* will make a copy for the daemon to print. If the file argument is preceded by —, or a preceding argument of — has been encountered, then *opr* will unlink (remove) the file.

If there are no arguments except for the optional ——, then the standard input is read and off-line printed. Thus *opr* may be used as a filter.

FILES

/usr/dpd/*	spool area
/etc/passwd	personal ident cards
/etc/dpd	daemon

SEE ALSO

dpd (VIII), passwd (V)

BUGS

— X X   Station X

— V   remove

— m   mail

NAME
        passwd — set login password

SYNOPSIS
        passwd name password

DESCRIPTION
        The *password* is placed on the given login name. This can only be done by the person corresponding to the login name or by the super-user. An explicit null argument ("") for the password argument will remove any password from the login name.

FILES
        /etc/passwd

SEE ALSO
        login(I), passwd(V), crypt(III)

BUGS

NAME
pfe — print floating exception

SYNOPSIS
pfe

DESCRIPTION
Pfe examines the floating point exception register and prints a diagnostic for the last floating point exception.

SEE ALSO
signal (II)

BUGS
Since the system does not save the exception register in a core image file, the message refers to the last error encountered by anyone. Floating exceptions are therefore volatile.

NAME

pr — print file

SYNOPSIS

pr [**−h** *header*] [*−n*] [*+n*] [*−wn*] [*−ln*] [*−t*] [*name* . . .]

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 header (if any), and the page number. If there are no file arguments, *pr* prints its standard input, and is thus usable as a filter.

Options apply to all following files but may be reset between files:

−n   produce *n*-column output

+n   begin printing with page *n*

−h   treat the next argument as a header

−wn for purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72

−ln take the length of the page to be *n* lines instead of the default 66

−t   do not print the 5-line header or the 5-line trailer normally supplied for each page

If there is a header in force, it is printed in place of the file name. Interconsole messages via write(I) are forbidden during a *pr*.

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(I), cp(I)

DIAGNOSTICS

none; files not found are ignored

BUGS

NAME
> prof — display profile data

SYNOPSIS
> **prof** [—v] [—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 or 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. If the —v option is used, all printing is suppressed and a profile plot is produced on the 611 display.
>
> 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.

FILES
> mon.out   for profile
> a.out   for namelist
> /dev/vt0   for plotting

SEE ALSO
> monitor(III), profil(II), cc(I)

BUGS

NAME
      ps — process status

SYNOPSIS
      ps [aklx]

DESCRIPTION
      Ps prints certain indicia about active processes. The **a** flag asks for information about all processes with teletypes (ordinarily only one's own processes are displayed); **x** asks even about processes with no typewriter; **l** asks for a long listing. Ordinarily only the typewriter number (if not one's own), the process number, and an approximation to the command line are given. If the **k** flag is specified, the special file */dev/rk0* is used in place of */dev/mem*. This is used for post-mortem system debugging.

      The long listing is columnar and contains

      A number encoding the state (last digit) and flags (first 1 or 2 digits) of the process.

      The priority of the process; high numbers mean low priority.

      A number related in some unknown way to the scheduling heuristic.

      The last character of the control typewriter of the process.

      The process unique number (as in certain cults it is possible to kill a process if you know its true name).

      The size in blocks of the core image of the process.

      The last column if non-blank tells the core address in the system of the event which the process is waiting for; if blank, the process is running.

      Ps makes an educated guess as to the file name and arguments given when the process was created by examining core memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES
      /unix   system namelist
      /dev/mem   core memory
      /dev/rf0   swap device
      /dev/rk0   optional mem file

SEE ALSO
      kill (I)

BUGS
      The command has assumptions built into it about the number of typewriters that exist and what hardware is used to interface them. It also has built into it the name of the device used for swapping.

NAME
       pwd − print working directory pathname

SYNOPSIS
       pwd

DESCRIPTION
       Pwd prints the pathname of the working (current) directory.

SEE ALSO
       chdir (I)

BUGS
       The algorithm sometimes fails on crossing mounted file systems.

NAME

     rew — rewind tape

SYNOPSIS

     rew [[m]digit]

DESCRIPTION

     Rew rewinds DECtape or magtape drives. The digit is the logical tape number, and should range from 0 to 7. if the digit is preceded by **m**, *rew* applies to magtape rather than DECtape. A missing digit indicates drive 0.

FILES

     /dèv/tap?
     /dev/mt?

BUGS

NAME
     rm − remove (unlink) files

SYNOPSIS
     rm [−f] [−r] name ...

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 nei-
     ther read nor write permission on the file itself.

     If there is no write permission to a file designated to be removed, *rm* will print the file name, its
     mode and then read a line from the standard input. If the line begins with y, the file is removed,
     otherwise it is not. The optional argument −f prevents this interaction.

     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.
     To remove directories *per se* see rmdir(I).

FILES
     /etc/glob to implement the −r flag

SEE ALSO
     rmdir(I)

BUGS
     When *rm* removes the contents of a directory under the −r flag, full pathnames are not printed
     in diagnostics.

NAME

        rmdir — remove directory

SYNOPSIS

        rmdir dir ...

DESCRIPTION

        Rmdir removes (deletes) directories. The directory must be empty (except for the standard en-
        tries '.' and '..', which *rmdir* itself removes). Write permission is required in the directory in
        which the directory appears.

BUGS

        Needs a **—r** flag. Actually, write permission in the directory's parent is *not* required.

NAME
       roff — format text

SYNOPSIS
       roff [+*n*] [−*n*] [−s] [−h] file ...

DESCRIPTION
       Roff formats text according to control lines embedded in the text in the given files. Encountering a nonexistent file terminates printing. Incoming interconsole messages are turned off during printing. The optional flag arguments mean:

       +*n*   Start printing at the first page with number *n*.

       −*n*   Stop printing at the first page numbered higher than *n*.

       −s   Stop before each page (including the first) to allow paper manipulation; resume on receipt of an interrupt signal.

       −h   Insert tabs in the output stream to replace spaces whenever appropriate.

       A Request Summary is attached.

FILES
       /usr/lib/suftab   suffix hyphenation tables
       /tmp/rtm?   temporary

SEE ALSO
       nroff (I), troff (I)

BUGS
       Roff is the simplest of the runoff programs, but is virtually undocumented.

REQUEST SUMMARY

Request	Break	Initial	Meaning
.ad	yes	yes	Begin adjusting right margins.
.ar	no	arabic	Arabic page numbers.
.br	yes	-	Causes a line break — the filling of the current line is stopped.
.bl n	yes	-	Insert of n blank lines, on new page if necessary.
.bp +n	yes	n=1	Begin new page and number it n; no n means '+1'.
.cc c	no	c=	Control character becomes 'c'.
.ce n	yes	-	Center the next n input lines, without filling.
.de xx	no	-	Define macro named 'xx' (definition ends on line beginning '..').
.ds	yes	no	Double space; same as '.ls 2'.
.ef t	no	t=""	Even foot title becomes t.
.eh t	no	t=""	Even head title becomes t.
.fi	yes	yes	Begin filling output lines.
.fo	no	t=""	All foot titles are t.
.hc c	no	none	Hyphenation character set to 'c'.
.he t	no	t=""	All head titles are t.
.hx	no	-	Title lines are suppressed.
.hy n	no	n=1	Hyphenation is done, if n=1; and is not done, if n=0.
.ig	no	-	Ignore input lines through a line beginning with '..'.
.in +n	yes	-	Indent n spaces from left margin.
.ix +n	no	-	Same as '.in' but without break.
.li n	no	-	Literal, treat next n lines as text.
.ll +n	no	n=65	Line length including indent is n characters.
.ls +n	yes	n=1	Line spacing set to n lines per output line.
.m1 n	no	n=2	Put n blank lines between the top of page and head title.
.m2 n	no	n=2	n blank lines put between head title and beginning of text on page.
.m3 n	no	n=1	n blank lines put between end of text and foot title.
.m4 n	no	n=3	n blank lines put between the foot title and the bottom of page.
.na	yes	no	Stop adjusting the right margin.
.ne n	no	-	Begin new page, if n output lines cannot fit on present page.
.nn +n	no	-	The next n output lines are not numbered.
.n1	no	no	Number output lines; start with 1 each page
.n2 n	no	no	Number output lines; stop numbering if n=0.
.ni +n	no	n=0	Line numbers are indented n.
.nf	yes	no	Stop filling output lines.
.nx filename		-	Change to input file 'filename'.
.of t	no	t=""	Odd foot title becomes t.
.oh t	no	t=""	Odd head title becomes t.
.pa +n	yes	n=1	Same as '.bp'.
.pl +n	no	n=66	Total paper length taken to be n lines.
.po +n	no	n=0	Page offset. All lines are preceded by N spaces.
.ro	no	arabic	Roman page numbers.
.sk n	no	-	Produce n blank pages starting next page.
.sp n	yes	-	Insert block of n blank lines.
.ss	yes	yes	Single space output lines, equivalent to '.ls 1'.
.ta N M ...		-	Pseudotab settings. Initial tab settings are columns 9,17,25,...
.tc c	no	c=' '	Tab replacement character becomes 'c'.
.ti +n	yes	-	Temporarily indent next output line n space.
.tr abcd..	no	-	Translate a into b, c into d, etc.
.ul n	no	-	Underline the letters and numbers in the next n input lines.

NAME

        sh — shell (command interpreter)

SYNOPSIS

        sh [name [arg1 ... [arg9]]]

DESCRIPTION

        Sh is the standard command interpreter. It is the program which reads and arranges the execution of the command lines typed by most users. It may itself be called as a command to interpret files of commands. Before discussing the arguments to the Shell used as a command, the structure of command lines themselves will be given.

        Commands. Each command is a sequence of non-blank command arguments separated by blanks. The first argument specifies the name of a command to be executed. Except for certain types of special arguments discussed below, the arguments other than the command name are passed without interpretation to the invoked command.

        If the first argument is the name of an executable file, it is invoked; otherwise the string '/bin/' is prepended to the argument. (In this way most standard commands, which reside in '/bin', are found.) If no such command is found, the string '/usr' is further prepended (to give '/usr/bin/command') and another attempt is made to execute the resulting file. ·(Certain lesser-used commands live in '/usr/bin'.) If the '/usr/bin' file exists, but is not executable, it is used by the Shell as a command file. That is to say it is executed as though it were typed from the console. If all attempts fail, a diagnostic is printed.

        Command lines. One or more commands separated by '|' or '^' constitute a chain of *filters*. The standard output of each command but the last is taken as the standard input of the next command. Each command is run as a separate process, connected by pipes (see pipe(II)) to its neighbors. A command line contained in parentheses '( )' may appear in place of a simple command as a filter.

        A *command line* consists of one or more pipelines separated, and perhaps terminated by ';' or '&'. The semicolon designates sequential execution. The ampersand causes the preceding pipeline to be executed without waiting for it to finish. The process id of such a pipeline is reported, so that it may be used if necessary for a subsequent *wait* or *kill*.

        Termination Reporting. If a command (not followed by '&') terminates abnormally, a message is printed. (All terminations other than exit and interrupt are considered abnormal.) Termination reports for commands followed by '&' are given upon receipt of the first command subsequent to the termination of the command, or when a *wait* is executed. The following is a list of the abnormal termination messages:

            Bus error
            Trace/BPT trap
            Illegal instruction
            IOT trap
            EMT trap
            Bad system call
            Quit
            Floating exception
            Memory violation
            Killed

        If a core image is produced, '— Core dumped' is appended to the appropriate message.

        Redirection of I/O. There are three character sequences that cause the immediately following string to be interpreted as a special argument to the Shell itself. Such an argument may appear anywhere among the arguments of a simple command, or before or after a parenthesized command list, and is associated with that command or command list.

An argument of the form '<arg' causes the file 'arg' to be used as the standard input (file descriptor 0) of the associated command.

An argument of the form '>arg' causes file 'arg' to be used as the standard output (file descriptor 1) for the associated command. 'Arg' is created if it did not exist, and in any case is truncated at the outset.

An argument of the form '>>arg' causes file 'arg' to be used as the standard output for the associated command. If 'arg' did not exist, it is created; if it did exist, the command output is appended to the file.

For example, either of the command lines

        ls >junk; cat tail > >junk
        (ls; cat tail) >junk

creates, on file 'junk', a listing of the working directory, followed immediately by the contents of file 'tail'.

Either of the constructs '>arg' or '>>arg' associated with any but the last command of a pipeline is ineffectual, as is '<arg' in any but the first.

In commands called by the Shell, file descriptor 2 refers to the standard output of the Shell before any redirection. Thus filters may write diagnostics to a location where they have a chance to be seen.

Generation of argument lists. If any argument contains any of the characters '?', '*' or '[', it is treated specially as follows. The current directory is searched for files which *match* the given argument.

The character '*' in an argument matches any string of characters in a file name (including the null string).

The character '?' matches any single character in a file name.

Square brackets '[...]' specify a class of characters which matches any single file-name character in the class. Within the brackets, each ordinary character is taken to be a member of the class. A pair of characters separated by '−' places in the class each character lexically greater than or equal to the first and less than or equal to the second member of the pair.

Other characters match only the same character in the file name.

For example, '*' matches all file names; '?' matches all one-character file names; '[ab]*.s' matches all file names beginning with 'a' or 'b' and ending with '.s'; '?[zi−m]' matches all two-character file names ending with 'z' or the letters 'i' through 'm'.

If the argument with '*' or '?' also contains a '/', a slightly different procedure is used: instead of the current directory, the directory used is the one obtained by taking the argument up to the last '/' before a '*' or '?'. The matching process matches the remainder of the argument after this '/' against the files in the derived directory. For example: '/usr/dmr/a*.s' matches all files in directory '/usr/dmr' which begin with 'a' and end with '.s'.

In any event, a list of names is obtained which match the argument. This list is sorted into alphabetical order, and the resulting sequence of arguments replaces the single argument containing the '*', '[', or '?'. The same process is carried out for each argument (the resulting lists are *not* merged) and finally the command is called with the resulting list of arguments.

For example: directory /usr/dmr contains the files a1.s, a2.s, ..., a9.s. From any directory, the command

        as /usr/dmr/a?.s

calls *as* with arguments /usr/dmr/a1.s, /usr/dmr/a2.s, ... /usr/dmr/a9.s in that order.

Quoting. The character '\' causes the immediately following character to lose any special meaning it may have to the Shell; in this way '<', '>', and other characters meaningful to the Shell may be passed as part of arguments. A special case of this feature allows the continuation of commands onto more than one line: a new-line preceded by '\' is translated into a blank.

Sequences of characters enclosed in double (") or single (') quotes are also taken literally. For example:

        ls pr —h "My directory"

causes a directory listing to be produced by *ls,* and passed on to *pr* to be printed with the heading 'My directory'. Quotes permit the inclusion of blanks in the heading, which is a single argument to *pr.*

Argument passing. When the Shell is invoked as a command, it has additional string processing capabilities. Recall that the form in which the Shell is invoked is

        sh [name [arg1 ... [arg9]]]

The *name* is the name of a file which will be read and interpreted. If not given, this subinstance of the Shell will continue to read the standard input file.

In command lines in the file (not in command input), character sequences of the form '$n', where *n* is a digit, are replaced by the *n*th argument to the invocation of the Shell (argn). '$0' is replaced by *name.*

End of file. An end-of-file in the Shell's input causes it to exit. A side effect of this fact means that the way to log out from UNIX is to type an EOT.

Special commands. The following commands are treated specially by the Shell.

chdir is done without spawning a new process by executing *sys chdir* (II).

login is done by executing /bin/login without creating a new process.

wait is done without spawning a new process by executing *sys wait* (II).

shift is done by manipulating the arguments to the Shell.

':' is simply ignored.

Command file errors; interrupts. Any Shell-detected error, or an interrupt signal, during the execution of a command file causes the Shell to cease execution of that file.

Process that are created with a '&' ignore interrupts. Also if such a process has not redirected its input with a '<', its input is automatically redirected to the zero length file /dev/null.

FILES

        /etc/glob, which interprets '*', '?', and '['.
        /dev/null as a source of end-of-file.

SEE ALSO

        'The UNIX Time-sharing System', which gives the theory of operation of the Shell.
        chdir (I), login (I), wait (I), shift (I)

BUGS

        There is no way to redirect the diagnostic output.

NAME

  shift — adjust Shell arguments

SYNOPSIS

  shift

DESCRIPTION

  Shift is used in Shell command files to shift the argument list left by 1, so that old $2 can now be referred to by $1 and so forth. *Shift* is useful to iterate over several arguments to a command file. For example, the command file

```
: loop
if $1x = x exit
pr −3 $1
shift
goto loop
```

prints each of its arguments in 3-column format.

Shift is executed within the Shell.

SEE ALSO

  sh (I)

BUGS

NAME

size — size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. If no file is specified, **a.out** is used.

BUGS

NAME
     sleep — suspend execution for an interval

SYNOPSIS
     sleep time

DESCRIPTION
     Sleep will suspend execution for *time* seconds. It is used to execute a command in a certain amount of time as in:

          (sleep 105; command)&

     Or to execute a command every so often as in this shell command file:

          : loop
          command
          sleep 37
          goto loop

SEE ALSO
     sleep(II)

BUGS
     Time must be less than 65536 seconds.

NAME
    sort — sort or merge files

SYNOPSIS
    sort [—abdnrtx] [+pos [—pos]] . . . [—mo] [name] . . .

DESCRIPTION
    Sort sorts all the named files together and writes the result on the standard output. The name '—' means the standard input. The standard input is also used if no input file names are given. Thus sort may be used as a filter.

    The default sort key is an entire line. Default ordering is lexicographic in ASCII collating sequence, except that lower-case letters are considered the same as the corresponding upper-case letters. Non-ASCII bytes are ignored. The ordering is affected by the flags —abdnrt, one or more of which may appear:

    a   Do not map lower case letters.

    b   Leading blanks (spaces and tabs) are not included in fields.

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

    n   An initial numeric string, consisting of optional minus sign, digits and optionally included decimal point, is sorted by arithmetic value.

    r   Reverse the sense of comparisons.

    tx   Tab character between fields is x.

    Selected parts of the line, specified by +pos and —pos, may be used as sort keys. Pos has the form m.n, where m specifies a number of fields to skip, and n a number of characters to skip further into the next field. A missing .n is taken to be 0. +pos denotes the beginning of the key; —pos denotes the first position after the key (end of line by default). The ordering rule may be overridden for a particular key by appending one or more of the flags abdnr to +pos.

    When no tab character has been specified, a field consists of nonblanks and any preceding blanks. Under the —b flag, leading blanks are excluded from a field. When a tab character has been specified, a field is a string ending with a tab character.

    When keys are specified, later keys are compared only when all earlier ones compare equal. Lines that compare equal are ordered with all bytes significant.

    These flag arguments are also understood:

    —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, except under the merge flag —m.

FILES
    /usr/tmp/stm???

NAME

spell — find spelling errors

SYNOPSIS

spell file ...

DESCRIPTION

Spell attacks the same problem as *typo (I)*, but from the opposite direction. It extracts words from the input files and looks them up in *Webster's Seventh Collegiate Dictionary;* any words which appear neither in the dictionary nor in a list of about 2000 words frequently occurring in Bell Laboratories documents are listed on the output file *sp.out.* Words which are reasonable transformations of dictionary entries (e.g. a dictionary entry plus *s*) are so marked; words which could not be found even when transformed are marked with asterisks.

The process takes on the order of 5 to 10 minutes. There is a limit of nine input files.

FILES

/usr/lib/w2006, /usr/dict/words, sp.out; spjnkq[123] are temporaries.

SEE ALSO

typo (I)

BUGS

There should be no limit on the number of input files.

More suffixes, and perhaps some prefixes, should be added.

It should be usable as a filter.

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.

BUGS

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 the same as use of the —s option of *ld*.

FILES
  /tmp/stm?   temporary file

SEE ALSO
  ld(I), as(I)

BUGS

NAME
        stty — set typewriter options

SYNOPSIS
        stty option ...

DESCRIPTION
        Stty will set certain I/O options on the current output typewriter. The option strings are selected
        from the following set:

even	allow even parity
—even	disallow even parity
odd	allow odd parity
—odd	disallow odd parity
raw	raw mode input (no erase, kill, interrupt, quit, EOT; parity bit passed back)
—raw	negate raw mode
—nl	allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl	accept only new-line 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 in output
tabs	preserve tabs
delay	calculate cr, tab, and form-feed delays
—delay	no cr/tab/ff delays
tdelay	calculate tab delays
—tdelay	no tab delays

SEE ALSO
        stty (II)

BUGS
        There should be 'package' options such as **execuport, 33,** or **terminet.**

NAME

　　sum — sum file

SYNOPSIS

　　sum name ...

DESCRIPTION

　　Sum sums the contents of the bytes (mod 2^{16}) of one or more files and prints the answer in octal. A separate sum is printed for each file specified, along with the number of whole or partial 512-byte blocks read.

　　In practice, *sum* is often used to verify that all of a special file can be read without error.

BUGS

NAME
      tee — pipe fitting

SYNOPSIS
      tee [name ...]

DESCRIPTION
      Tee transcribes the standard input to the standard output and makes copies in the named files.

BUGS

NAME

      time — time a command

SYNOPSIS

      time command

DESCRIPTION

      The given command is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command.

      The execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

      The times are printed on the diagnostic output stream.

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
     tp — manipulate DECtape and magtape

SYNOPSIS
     tp [key] [name ...]

DESCRIPTION
     Tp saves and restores selected portions of the file system hierarchy on DECtape or mag tape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed.

     The function portion of the key is specified by one of the following letters:

     r   The indicated files and directories, together with all subdirectories, are dumped onto the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so './abc' can never be the same as '/usr/dmr/abc' even if '/usr/dmr' is the current directory. If no file argument is given, '.' is the default.

     u   updates the tape. u is the same as r, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. u is the default command if none is given.

     d   deletes the named files and directories from the tape. At least one file argument must be given. This function is not permitted on magtapes.

     x   extracts the named files from the tape to the file system. The owner, mode, and date-modified are restored to what they were when the file was dumped. If no file argument is given, the entire contents of the tape are extracted.

     t   lists the names of all files stored on the tape which are the same as or are hierarchically below the file arguments. If no file argument is given, the entire contents of the tape is listed.

     The following characters may be used in addition to the letter which selects the function desired.

     m   Specifies magtape as opposed to DECtape.

     0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, 'x' is default; for magtape '0' is the default.

     v   Normally *tp* does its work silently. The v (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the t function, v gives more information about the tape entries than just the name.

     c   means a fresh dump is being created; the tape directory will be zeroed before beginning. Usable only with r and u. This option is assumed with magtape since it is impossible to selectively overwrite magtape.

     f   causes new entries on tape to be 'fake' in that no data is present for these entries. Such fake entries cannot be extracted. Usable only with r and u.

     i   Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

     w   causes *tp* to pause before treating each file, type the indicative letter and the file name (as with v) and await the user's response. Response y means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response x means 'exit'; the *tp* command terminates immediately. In the x function, files previously asked about have been extracted already. With r, u, and d no change has been made to the tape.

FILES
     /dev/tap?
     /dev/mt?

DIAGNOSTICS
   Several; the non-obvious one is 'Phase error', which means the file changed after it was selected
   for dumping but before it was dumped.

BUGS

NAME

tr — transliterate

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*. If *string2* is short, it is padded with corresponding characters from *string1*. Any combination of the options **—cds** may be used. **—c** complements the set of characters in *string1* with respect to the universe of characters whose ascii codes are 001 through 377 octal. **—d** deletes all input characters in *string1*. **—s** squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[*a—b*] stands for the string of characters whose ascii codes run from character *a* to character *b*.

[*a**n], where *n* is an integer or empty, stands for *n*-fold repetition of character *a*. *n* is taken to be octal or decimal according as its first digit is or is not zero. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character '\' may be used as in *sh* to remove special meaning from any character in a string. In addition, '\' followed by 1, 2 or 3 octal digits stands for the character whose ascii code is given by those digits.

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the Shell; 012 is the ascii code for newline.

    tr —cs "[A—Z][a—z]" "[\012*]" <file1 >file2

SEE ALSO

sh (I), ed (I), ascii (VII)

BUGS

Won't handle ascii NUL in *string1* or *string2;* always deletes NUL from input.

NAME
       troff — format text

SYNOPSIS
       troff [+*n*] [−*n*] [−n*n*] [−m*x*] [−t] [−f] [−w] [−i] [−a] [−p*n*] files

DESCRIPTION
       Troff formats text for a Graphic Systems phototypesetter according to control lines embedded in
       the text files. It reads the standard input if no file arguments are given. The non-file option ar-
       guments are interpreted as follows:

       +*n*   Commence typesetting at the first page numbered *n* or larger.

       −*n*   Stop after page *n*.

       −n*n*   First generated (not necessarily printed) page is given the number *n;* simulates ".pn *n*".

       −m*x*   Prepends a standard macro file; simulates ".so /usr/lib/tmac.*x*".

       −t   Place output on standard output instead of the phototypesetter.

       −f   Refrain from feeding out paper and stopping the phototypesetter at the end.

       −w   Wait until phototypsetter is available, if currently busy.

       −i   Read from standard input after the files have been exhausted.

       −a   Send a printable approximation of the results to the standard output.

       −p*n*   Print all characters with point-size *n* while retaining all prescribed spacings and motions.

FILES
       /usr/lib/suftab   suffix hyphenation tables
       /tmp/rtm?   temporary
       /usr/lib/tmac.x   standard macro files

SEE ALSO
       TROFF User's Manual (internal memorandum).
       TROFF Made Trivial (internal memorandum).
       nroff (I), eqn (I) catsim (VI)

BUGS

NAME

tss — interface to MH-TSS

SYNOPSIS

tss

DESCRIPTION

Tss will call the Honeywell 6070 on the 201 data phone. It will then go into direct access with MH-TSS. Output generated by MH-TSS is typed on the standard output and input requested by MH-TSS is read from the standard input with UNIX typing conventions.

An interrupt signal is transmitted as a 'break' to MH-TSS.

Input lines beginning with '!' are interpreted as UNIX commands. Input lines beginning with '~' are interpreted as commands to the interface routine.

~⟨file	insert input from named UNIX file
~⟩file	deliver tss output to named UNIX file
~p	pop the output file
~q	disconnect from tss (quit)
~r file	receive from HIS routine csr/daccopy
~s file	send file to HIS routine csr/daccopy

Ascii files may be most efficiently transmitted using the HIS routine csr/daccopy in this fashion. Bold face text comes from MH-TSS. *Aftname* is the 6070 file to be dealt with; *file* is the UNIX file.

SYSTEM? csr/daccopy (s) *aftname*
Send Encoded File ~s *file*

SYSTEM? csr/daccopy (r) *aftname*
Receive Encoded File ~r *file*

FILES

/dev/dn0, /dev/dp0, /etc/msh

DIAGNOSTICS

Most often, 'Transmission error on last message.'

BUGS

When problems occur, and they often do, *tss* exits rather abruptly.

NAME

      tty — get typewriter name

SYNOPSIS

      tty

DESCRIPTION

      Tty gives the name of the user's typewriter in the form 'tty*n*' for *n* a digit or letter. The actual path name is then '/dev/tty*n*'.

DIAGNOSTICS

      'not a tty' if the standard input file is not a typewriter.

BUGS

NAME

    typo — find possible typos

SYNOPSIS

    typo [−1] [−n] file ...

DESCRIPTION

    Typo hunts through a document for unusual words, typographic errors, and *hapax legomena* and prints them on the standard output.

    The words used in the document are printed out in decreasing order of peculiarity along with an index of peculiarity. An index of 10 or more is considered peculiar. Printing of certain very common English words is suppressed.

    The statistics for judging words are taken from the document itself, with some help from known statistics of English. The −n option suppresses the help from English and should be used if the document is written in, for example, Urdu.

    The −1 option causes the final output to appear in a single column instead of three columns. The normal header and pagination is also suppressed.

    Roff (I) and nroff (I) control lines are ignored. Upper case is mapped into lower case. Quote marks, vertical bars, hyphens, and ampersands are stripped from within words. Words hyphenated across lines are put back together.

FILES

    /tmp/ttmp??
    /usr/lib/salt
    /usr/lib/w2006

BUGS

    Because of the mapping into lower case and the stripping of special characters, words may be hard to locate in the original text.

    The escape sequences of troff (I) are not correctly recognized.

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(I). 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 (I), comm (I)

BUGS

NAME

wait — await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & have completed, and report on abnormal terminations.

Because *sys wait* must be executed in the parent process, the Shell itself executes *wait,* without creating a new process.

SEE ALSO

sh (I)

BUGS

After executing *wait* you are committed to waiting until termination, because interrupts and quits are ignored by all processes concerned. The only out, if the process does not terminate, is to *kill* it from another terminal or to hang up.

NAME
wc — word count

SYNOPSIS
wc [—rlwapc] [name ...]

DESCRIPTION
Wc counts lines and words in the named files, or in the standard input if no name appears. A word is a maximal string of ascii graphics delimited by spaces, tabs or newlines. Other characters are always ignored.

Any of the following options may appear in any order. When any option other than —r appears, only the specified information is reported in the order in which the options occur. Otherwise, printing is as with option —lw.

—r Ignore all *roff, nroff* and *troff* control lines, i.e. lines beginning with '.' or '"',

—l Print count of lines.

—w Print count of words.

—a Print count of alphanumeric strings, with underscore taken as alphanumeric.

—p Print count of punctuation strings, i.e. all strings of printing characters other than alphanumerics.

—c Print count of *roff* control lines, regardless of —r.

DIAGNOSTICS
'Cannot open' for unopenable file.

BUGS

NAME

    who — who is on the system

SYNOPSIS

    who [who-file]

DESCRIPTION

    Who, without an argument, lists the name, typewriter channel, and login time for each current UNIX user.

    Without an argument, *who* examines the /tmp/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file will be /tmp/wtmp, which contains a record of all the logins since it was created. Then *who* will list logins, logouts, and crashes since the creation of the wtmp file.

    Each login is listed with user name, typewriter 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.

FILES

    /tmp/utmp

SEE ALSO

    login (I), init (VIII)

BUGS

NAME

write — write to another user

SYNOPSIS

write user [ttyno]

DESCRIPTION

Write copies lines from your typewriter to that of another user. When first called, it sends the message

message from yourname...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the typewriter 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 *ttyno* argument may be used to indicate the last character of the appropriate typewriter name.

Permission to write may be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular *roff* and *pr*, disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, *write* calls the mini-shell *msh* 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

/tmp/utmp	to find user
/etc/msh	to execute '!'

SEE ALSO

mesg(I), who(I)

BUGS

INTRODUCTION TO SYSTEM CALLS

Section II of this manual lists all the entries into the system. In most cases two calling sequences are specified, one of which is usable from assembly language, and the other from C. Most of these calls have an error return. From assembly language an erroneous call is always indicated by turning on the c-bit of the condition codes. The presence of an error is most easily tested by the instructions *bes* and *bec* ("branch on error set (or clear)"). These are synonyms for the *bcs* and *bcc* instructions.

From C, an error condition is indicated by an otherwise impossible returned value. Almost always this is −1; the individual sections specify the details.

In both cases an error number is also available. In assembly language, this number is returned in r0 on erroneous calls. From C, the external variable *errno* is set to the error number. *Errno* is not cleared on succesful calls, so it should be tested only after an error has occurred. There is a table of messages associated with each error, and a routine for printing the message. See *perror (III)*.

The possible error numbers are not recited with each writeup in section II, since many errors are possible for most of the calls. Here is a list of the error numbers, their names inside the system (for the benefit of system-readers), and the messages available using *perror*. A short explanation is also provided.

0   —   (unused)

1   EPERM   Not owner and not super-user
     Typically this error indicates an attempt to modify a file in some way forbidden except to its owner. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2   ENOENT   No such file or directory
     This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3   ESRCH   No such process
     The process whose number was given to *signal* does not exist, or is already dead.

4   —   (unused)

5   EIO   I/O error
     Some physical I/O error occurred during a *read* or *write*. This error may in some cases occur on a call following the one to which it actually applies.

6   ENXIO   No such device or address
     I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not dialled in or no disk pack is loaded on a drive.

7   E2BIG   Arg list too long
     An argument list longer than 512 bytes (counting the null at the end of each argument) is presented to *exec*.

8   ENOEXEC   Exec format error
     A request is made to execute a file which, although it has the appropriate permissions, does not start with one of the magic numbers 407 or 410.

9   EBADF   Bad file number
     Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file which is open only for writing (resp. reading).

10   ECHILD   No children
Wait and the process has no living or unwaited-for children.

11   EAGAIN   No more processes
In a *fork*, the system's process table is full and no more processes can for the moment be created.

12   ENOMEM   Not enough core
During an *exec* or *break*, a program asks for more core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments is such as to require more than the existing 8 segmentation registers.

13   EACCES   Permission denied
An attempt was made to access a file in a way forbidden by the protection system.

14   —   (unused)

15   ENOTBLK   Block device required
A plain file was mentioned where a block device was required, e.g. in *mount*.

16   EBUSY   Mount device busy
An attempt was made to dismount a device on which there is an open file or some process's current directory.

17   EEXIST   File exists
In existing file was mentioned in a context in which it should not have, e.g. *link*.

18   EXDEV   Cross-device link
A link to a file on another device was attempted.

19   ENODEV   No such device
An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20   ENOTDIR   Not a directory
A non-directory was specified where a directory is required, for example in a path name or as an argument to *chdir*.

21   EISDIR   Is a directory
An attempt to write on a directory.

22   EINVAL   Invalid argument
Some invalid argument: currently, dismounting a non-mounted device, mentioning an unknown signal in *signal*, and giving an unknown request in *stty* to the TIU special file.

23   ENFILE   File table overflow
The system's table of open files is full, and temporarily no more *opens* can be accepted.

24   EMFILE   Too many open files
Only 10 files can be open per process; this error occurs when the eleventh is opened.

25   ENOTTY   Not a typewriter
The file mentioned in *stty* or *gtty* is not a typewriter or one of the other devices to which these calls apply.

26   ETXTBSY   Text file busy
An attempt to execute a pure-procedure program which is currently open for writing (or reading!).

27   EFBIG   File too large
      An attempt to make a file larger than the maximum of 2048 blocks.

28   ENOSPC   No space left on device
      During a *write* to an ordinary file, there is no free space left on the device.

29   ESPIPE   Seek on pipe
      A *seek* was issued to a pipe., This error should also be issued for other non-seekable devices.

30   EROFS   Read-only file system
      An attempt to modify a file or directory was made on a device mounted read-only.

NAME

        break — set program break

SYNOPSIS

        (break = 17.)
        sys break; addr

        char *brk(addr)

        char *sbrk(incr)

DESCRIPTION

        Break sets the system's idea of the lowest location not used by the program (called the break) to *addr* (rounded up to the next multiple of 64 bytes). Locations not less than *addr* and below the stack pointer are not in the address space and will thus cause a memory violation if accessed.

        From C, *brk* will set the break to *addr*. The old break is returned.

        In the alternate entry *sbrk, incr* more bytes are added to the program's data space and a pointer to the start of the new area is returned.

        When a program begins execution via *exec* the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use *break*.

SEE ALSO

        exec (II), alloc (III)

DIAGNOSTICS

        The c-bit is set if the program requests more memory than the system limit or if more than 8 segmentation registers would be required to implement the break. From C, −1 is returned for these errors.

NAME

    chdir — change working directory

SYNOPSIS

    (chdir = 12.)

    sys chdir; dirname

    chdir(dirname)

    char *dirname;

DESCRIPTION

    Dirname is the address of the pathname of a directory, terminated by a null byte. *Chdir* causes this directory to become the current working directory.

SEE ALSO

    chdir(I)

DIAGNOSTICS

    The error bit (c-bit) is set if the given name is not that of a directory or is not readable. From C, a −1 returned value indicates an error, 0 indicates success.

NAME
        chmod — change mode of file

SYNOPSIS
        (chmod = 15.)
        sys chmod; name; mode

        chmod(name, mode)
        char *name;

DESCRIPTION
        The file whose name is given as the null-terminated string pointed to by *name* has its mode
        changed to *mode*. Modes are constructed by ORing together some combination of the following:

                4000 set user ID on execution
                2000 set group ID on execution
                0400 read by owner
                0200 write by owner
                0100 execute (search on directory) by owner
                0070 read, write, execute (search) by group
                0007 read, write, execute (search) by others

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

SEE ALSO
        chmod (I)

DIAGNOSTIC
        Error bit (c-bit) set if *name* cannot be found or if current user is neither the owner of the file nor
        the super-user. From C, a −1 returned value indicates an error, 0 indicates success.

NAME
        chown — change owner

SYNOPSIS
        (chmod = 16.)
        sys chown; name; owner

        chown(name, owner)
        char *name;

DESCRIPTION
        The file whose name is given by the null-terminated string pointed to by *name* has its owner
        changed to *owner* (a numerical user ID). Only the present owner of a file (or the super-user) may
        donate the file to another user. Changing the owner of a file removes the set-user-ID protection
        bit unless it is done by the super user.

SEE ALSO
        chown (I), passwd (V)

DIAGNOSTICS
        The error bit (c-bit) is set on illegal owner changes. From C a −1 returned value indicates error,
        0 indicates success.

NAME
        close — close a file

SYNOPSIS
        (close = 6.)
        (file descriptor in r0)
        sys close

        close(fildes)

DESCRIPTION
        Given a file descriptor such as returned from an *open, creat,* or *pipe* call, *close* closes the associat-
        ed file. A close of all files is automatic on *exit,* but since processes are limited to 15 simultane-
        ously open files, *close* is necessary for programs which deal with many files.

SEE ALSO
        creat (II), open (II), pipe (II)

DIAGNOSTICS
        The error bit (c-bit) is set for an unknown file descriptor. From C a −1 indicates an error, 0 indi-
        cates success.

NAME

creat — create a new file

SYNOPSIS

(creat = 8.)
sys creat; name; mode
(file descriptor in r0)

creat(name, mode)
char *name;

DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file called *name*, given as the address of a null-terminated string. If the file did not exist, it is given mode *mode*. See chmod(II) for the construction of the *mode* argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to 0 length.

The file is also opened for writing, and its file descriptor is returned (in r0).

The *mode* given is arbitrary; it need not allow writing. This feature is used by programs which deal with temporary files of fixed names. The creation is done with a mode that forbids writing. Then if a second instance of the program attempts a *creat*, an error is returned and the program knows that the name is unusable for the moment.

SEE ALSO

write (II), close (II), stat (II)

DIAGNOSTICS

The error bit (c-bit) may be set if: a needed directory is not searchable; the file does not exist and the directory in which it is to be created is not writable; the file does exist and is unwritable; the file is a directory; there are already too many files open.

From C, a −1 return indicates an error.

NAME
      csw — read console switches

SYNOPSIS
      (csw = 38.; not in assembler)
      sys   csw

      getcsw( )

DESCRIPTION
      The setting of the console switches is returned (in r0).

NAME
    dup — duplicate an open file descriptor

SYNOPSIS
    (dup = 41.; not in assembler)
    (file descriptor in r0)
    sys dup

    dup(fildes)
    int fildes;

DESCRIPTION
    Given a file descriptor returned from an *open, pipe,* or *creat* call, *dup* will allocate another file
    descriptor synonymous with the original. The new file descriptor is returned in r0.

    Dup is used more to reassign the value of file descriptors than to genuinely duplicate a file
    descriptor. Since the algorithm to allocate file descriptors returns the lowest available value,
    combinations of *dup* and *close* can be used to manipulate file descriptors in a general way. This
    is handy for manipulating standard input and/or standard output.

SEE ALSO
    creat (II), open (II), close (II), pipe (II)

DIAGNOSTICS
    The error bit (c-bit) is set if: the given file descriptor is invalid; there are already too many open
    files. From C, a −1 returned value indicates an error.

NAME

       exec — execute a file

SYNOPSIS

       (exec = 11.

       sys exec; name; args

       ...

       name: <...\0>

       ...

       args: arg1; arg2; ...; 0

       arg1: <...\0>

       arg2: <...\0>

         ...

       execl(name, arg1, arg2, ..., argn, 0)

       char *name, *arg1, *arg2, ..., *argn;

       execv(name, argv)

       char *name;

       char *argv[ ];

DESCRIPTION

       Exec overlays the calling process with the named file, then transfers to the beginning of the core image of the file. There can be no return from the file; the calling core image is lost.

       Files remain open across *exec* calls. Ignored signals remain ignored across *exec*, but signals that are caught are reset to their default values.

       Each user has a *real* user ID and group ID and an *effective* user ID and group ID (The real ID identifies the person using the system; the effective ID determines his access privileges.) *Exec* changes the effective user and group ID to the owner of the executed file if the file has the "set-user-ID" or "set-group-ID" modes. The real user ID is not affected.

       The form of this call differs somewhat depending on whether it is called from assembly language or C; see below for the C version.

       The first argument to *exec* is a pointer to the name of the file to be executed. The second is the address of a null-terminated list of pointers to arguments to be passed to the file. Conventionally, the first argument is the name of the file. Each pointer addresses a string terminated by a null byte.

       Once the called file starts execution, the arguments are available as follows. The stack pointer points to a word containing the number of arguments. Just above this number is a list of pointers to the argument strings. The arguments are placed as high as possible in core.

```
sp→     nargs
        arg1
        ...
        argn
        −1

arg1:   <arg1\0>
        ...
argn:   <argn\0>
```

       From C, two interfaces are available. *execl* is useful when a known file with known arguments is being called; the arguments to *execl* are the character strings constituting the file and the arguments; as in the basic call, the first argument is conventionally the same as the file name (or its last component). A 0 argument must end the argument list.

       The *execv* version is useful when the number of arguments is unknown in advance; the arguments to *execv* are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

When a C program is executed, it is called as follows:

```
main(argc, argv)
int argc;
char *argv[ ];
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file.

Argv is not directly usable in another *execv*, since *argv[argc]* is −1 and not 0.

SEE ALSO
fork (II)

DIAGNOSTICS
If the file cannot be found, if it is not executable, if it does not have a valid header (407 or 410 octal as first word), if maximum memory is exceeded, or if the arguments require more than 512 bytes a return from *exec* constitutes the diagnostic; the error bit (c-bit) is set. From C the returned value is −1.

BUGS
Only 512 characters of arguments are allowed.

NAME
        exit — terminate process

SYNOPSIS
        (exit = 1.)
        (status in r0)
        sys exit

        exit(status)
        int status;

DESCRIPTION
        Exit is the normal means of terminating a process. *Exit* closes all the process's files and notifies
        the parent process if it is executing a *wait*. The low byte of r0 (resp. the argument to *exit*) is
        available as status to the parent process.

        This call can never return.

SEE ALSO
        wait (II)

DIAGNOSTICS
        None.

NAME

     fork — spawn new process

SYNOPSIS

     (fork = 2.)
     sys fork
     (new process return)
     (old process return)

     fork( )

DESCRIPTION

     Fork is the only way new processes are created. The new process's core image is a copy of that of the caller of *fork*. The only distinction is the return location and the fact that r0 in the old (parent) process contains the process ID of the new (child) process. This process ID is used by *wait*.

     The two returning processes share all open files that existed before the call. In particular, this is the way that standard input and output files are passed and also how pipes are set up.

     From C, the returned value is 0 in the child process, non-zero in the parent process; however, a return of −1 indicates inability to create a new process.

SEE ALSO

     wait (II), exec (II)

DIAGNOSTICS

     The error bit (c-bit) is set in the old process if a new process could not be created because of lack of process space. From C, a return of −1 (not just negative) indicates an error.

NAME
       fstat — get status of open file

SYNOPSIS
       (fstat = 28.)
       (file descriptor in r0)
       sys fstat; buf

       fstat(fildes, buf)
       struct inode buf;

DESCRIPTION
       This call is identical to *stat,* except that it operates on open files instead of files given by name.
       It is most often used to get the status of the standard input and output files, whose names are
       unknown.

SEE ALSO
       stat(II)

DIAGNOSTICS
       The error bit (c-bit) is set if the file descriptor is unknown; from C, a −1 return indicates an er-
       ror, 0 indicates success.

NAME

getgid — get group identifications

SYNOPSIS

(getgid = 47.; not in assembler)

sys getgid

getgid( )

DESCRIPTION

Getgid returns a word, the low byte of which contains the real group ID of the current process. The high byte contains the effective group ID of the current process. The real group ID identifies the group of the person who is logged in, in contradistinction to the effective group ID, which determines his access permission at the moment. It is thus useful to programs which operate using the "set group ID" mode, to find out who invoked them.

SEE ALSO

setgid (II)

DIAGNOSTICS

NAME

     getuid — get user identifications

SYNOPSIS

     (getuid = 24.)

     sys getuid

     getuid( )

DESCRIPTION

     Getuid returns a word, the low byte of which contains the real user ID of the current process. The high byte contains the effective user ID of the current process. The real user ID identifies the person who is logged in, in contradistinction to the effective user ID, which determines his access permission at the moment. It is thus useful to programs which operate using the "set user ID" mode, to find out who invoked them.

SEE ALSO

     setuid (II)

DIAGNOSTICS

     —

NAME
       gtty — get typewriter status

SYNOPSIS
       (gtty = 32.)
       (file descriptor in r0)
       sys gtty; arg
       ...
       arg: .=.+6

       gtty(fildes, arg)
       int arg[3];

DESCRIPTION
       Gtty stores in the three words addressed by *arg* the status of the typewriter whose file descriptor
       is given in r0 (resp. given as the first argument). The format is the same as that passed by *stty*.

SEE ALSO
       stty (II)

DIAGNOSTICS
       Error bit (c-bit) is set if the file descriptor does not refer to a typewriter. From C, a −1 value is
       returned for an error, 0, for a successful call.

NAME

indir — indirect system call

SYNOPSIS

(indir = 0.; not in assembler)
sys indir; syscall

DESCRIPTION

The system call at the location *syscall* is executed. Execution resumes after the *indir* call.

The main purpose of *indir* is to allow a program to store arguments in system calls and execute them out of line in the data segment. This preserves the purity of the text segment.

If *indir* is executed indirectly, it is a no-op.

SEE ALSO

—

DIAGNOSTICS

—

NAME
>   kill — send signal to a process

SYNOPSIS
>   (kill = 37.; not in assembler)
>   (process number in r0)
>   **sys kill; sig**
>
>   **kill(pid, sig);**

DESCRIPTION
>   *Kill* sends the signal *sig* to the process specified by the process number in r0. See signal (II) for a list of signals.
>
>   The sending and receiving processes must have the same controlling typewriter, otherwise this call is restricted to the super-user.

SEE ALSO
>   signal (II), kill (I)

DIAGNOSTICS
>   The error bit (c-bit) is set if the process does not have the same controlling typewriter and the user is not super-user, or if the process does not exist.

BUGS
>   Equality between the controlling typewriters of the sending and receiving process is neither a necessary nor sufficient condition for allowing the sending of a signal. The correct condition is equality of user IDs.

- 1 -

NAME

    link — link to a file

SYNOPSIS

    (link = 9.)
    sys link; name1; name2

    link(name1, name2)
    char *name1, *name2;

DESCRIPTION

    A link to *name1* is created; the link has the name *name2*. Either name may be an arbitrary path name.

SEE ALSO

    link(I), unlink(II)

DIAGNOSTICS

    The error bit (c-bit) is set when *name1* cannot be found; when *name2* already exists; when the directory of *name2* cannot be written; when an attempt is made to link to a directory by a user other than the super-user; when an attempt is made to link to a file on another file system. From C, a —1 return indicates an error, a 0 return indicates success.

NAME
       mknod — make a directory or a special file

SYNOPSIS
       (mknod = 14.; not in assembler)
       sys mknod; name; mode; addr

       mknod(name, mode, addr)
       char *name;

DESCRIPTION
       Mknod creates a new file whose name is the null-terminated string pointed to by *name*. The
       mode of the new file (including directory and special file bits) is initialized from *mode*. The first
       physical address of the file is initialized from *addr*. Note that in the case of a directory, *addr*
       should be zero. In the case of a special file, *addr* specifies which special file.

       Mknod may be invoked only by the super-user.

SEE ALSO
       mkdir (I), mknod (VIII), fs (V)

DIAGNOSTICS
       Error bit (c-bit) is set if the file already exists or if the user is not the super-user. From C, a −1
       value indicates an error.

NAME

mount — mount file system

SYNOPSIS

(mount = 21.)
sys mount; special; name; rwflag

mount(special, name, rwflag)
char *special, *name;

DESCRIPTION

Mount announces to the system that a removable file system has been mounted on the block-structured special file *special;* from now on, references to file *name* will refer to the root file on the newly mounted file system. *Special* and *name* are pointers to null-terminated strings containing the appropriate path names.

Name must exist already. Its old contents are inaccessible while the file system is mounted.

The *rwflag* argument determines whether the file system can be written on; if it is 0 writing is allowed, if non-zero no writing is done. Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

SEE ALSO

mount (VIII), umount (II)

DIAGNOSTICS

Error bit (c-bit) set if: *special* is inaccessible or not an appropriate file; *name* does not exist; *special* is already mounted; there are already too many file systems mounted.

NAME

nice — set program priority

SYNOPSIS

(nice = 34.)
(priority in r0)
sys nice

nice(priority)

DESCRIPTION

The currently executing process is set into the priority specified by *priority*. If *priority* is positive, the priority of the process is below default; if negative the process must be the super-user and its priority is raised. The valid range of *priority* is between 20 and −220. The value of 16 is recommended to users who wish to execute long-running programs without flak from the administration.

The effect of this call is passed to a child process by the *fork* system call. The effect can be cancelled by another call to *nice* with a *priority* of 0.

SEE ALSO

nice(I)

DIAGNOSTICS

The error bit (c-bit) is set if the user requests a *priority* outside the range of 0 to 20 and is not the super-user.

NAME
        open — open for reading or writing

SYNOPSIS
        (open = 5.)
        sys open; name; mode

        open(name, mode)
        char *name;

DESCRIPTION
        Open opens the file *name* for reading (if *mode* is 0), writing (if *mode* is 1) or for both reading and writing (if *mode* is 2). *Name* is the address of a string of ASCII characters representing a path name, terminated by a null character.

        The returned file descriptor should be saved for subsequent calls to *read, write,* and *close.*

SEE ALSO
        creat (II), read (II), write (II), close (II)

DIAGNOSTICS
        The error bit (c-bit) is set if the file does not exist, if one of the necessary directories does not exist or is unreadable, if the file is not readable (resp. writable), or if too many files are open. From C, a −1 value is returned on an error.

NAME
         pipe — create a pipe

SYNOPSIS
         (pipe = 42.)
         sys pipe
         (read file descriptor in r0)
         (write file descriptor in r1)

         pipe(fildes)
         int fildes[2];

DESCRIPTION
         The *pipe* system call creates an I/O mechanism called a pipe. The file descriptors returned can be
         used in read and write operations. When the pipe is written using the descriptor returned in r1
         (resp. fildes[1]), up to 4096 bytes of data are buffered before the writing process is suspended. A
         read using the descriptor returned in r0 (resp. fildes[0]) will pick up the data.

         It is assumed that after the pipe has been set up, two (or more) cooperating processes (created by
         subsequent *fork* calls) will pass data through the pipe with *read* and *write* calls.

         The Shell has a syntax to set up a linear array of processes connected by pipes.

         Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed)
         return an end-of-file. Write calls under similar conditions are ignored.

SEE ALSO
         sh (I), read (II), write (II), fork (II)

DIAGNOSTICS
         The error bit (c-bit) is set if too many files are already open. From C, a −1 returned value indi-
         cates an error.

- 1 -

NAME
        profil — execution time profile

SYNOPSIS
        (profil = 44.; not in assembler)
        sys   profil; buff; bufsiz; offset; scale

        profil(buff, bufsiz, offset, scale)
        char buff[ ];
        int bufsiz, offset, scale;

DESCRIPTION
        Buff points to an area of core whose length (in bytes) is given by *bufsiz*. After this call, the user's
        program counter (pc) is examined each clock tick (60th second); *offset* is subtracted from it, and
        the result multiplied by *scale*. If the resulting number corresponds to a word inside *buff*, that
        word is incremented.

        The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left:
        177777(8) gives a 1-1 mapping of pc's to words in *buff*; 77777(8) maps each pair of instruction
        words together. 2(8) maps all instructions onto the beginning of *buff* (producing a non-
        interrupting core clock).

        Profiling is turned off by giving a *scale* of 0 or 1. It is rendered ineffective by giving a *bufsiz* of 0.
        Profiling is also turned off when an *exec* is executed but remains on in child and parent both
        after a *fork*.

SEE ALSO
        monitor (III), prof (I)

DIAGNOSTICS

NAME
        read — read from file

SYNOPSIS
        (read = 3.)
        (file descriptor in r0)
        sys read; buffer; nbytes

        read(fildes, buffer, nbytes)
        char *buffer;

DESCRIPTION
        A file descriptor is a word returned from a successful *open*, or *pipe* call. *Buffer* is the location of *nbytes* contiguous bytes into which the input will be placed. It is not guaranteed that all *nbytes* bytes will be read; for example if the file refers to a typewriter at most one line will be returned. In any event the number of characters read is returned (in r0).

        If the returned value is 0, then end-of-file has been reached.

SEE ALSO
        open (II), pipe (II)

DIAGNOSTICS
        As mentioned, 0 is returned when the end of the file has been reached. If the read was otherwise unsuccessful the error bit (c-bit) is set. Many conditions can generate an error: physical I/O errors, bad buffer address, preposterous *nbytes*, file descriptor not that of an input file. From C, a —1 return indicates the error.

NAME

       seek — move read/write pointer

SYNOPSIS

       (seek = 19.)
       (file descriptor in r0)
       sys seek; offset; ptrname

       seek(fildes, offset, ptrname)

DESCRIPTION

       The file descriptor refers to a file open for reading or writing. The read (resp. write) pointer for the file is set as follows:

          if *ptrname* is 0, the pointer is set to *offset*.

          if *ptrname* is 1, the pointer is set to its current location plus *offset*.

          if *ptrname* is 2, the pointer is set to the size of the file plus *offset*.

          if *ptrname* is 3, 4 or 5, the meaning is as above for 0, 1 and 2 except that the offset is multiplied by 512.

       If *ptrname* is 0 or 3, *offset* is unsigned, otherwise it is signed.

SEE ALSO

       open(II), creat(II)

DIAGNOSTICS

       The error bit (c-bit) is set for an undefined file descriptor. From C, a −1 return indicates an error.

NAME
    setgid — set process's group ID

SYNOPSIS
    (setgid = 46.)
    (group ID in r0)
    sys setgid

    setgid(gid)

DESCRIPTION
    The group ID of the current process is set to the argument. Both the effective and the real group
    ID are set. This call is only permitted to the super-user or if the argument is the real group ID.

SEE ALSO
    getgid(II)

DIAGNOSTICS
    Error bit (c-bit) is set as indicated; from C, a −1 value is returned.

NAME

      setuid — set process's user ID

SYNOPSIS

      (setuid = 23.)
      (user ID in r0)
      sys setuid

      setuid(uid)

DESCRIPTION

      The user ID of the current process is set to the argument. Both the effective and the real user ID
      are set. This call is only permitted to the super-user or if the argument is the real user ID.

SEE ALSO

      getuid(II)

DIAGNOSTICS

      Error bit (c-bit) is set as indicated; from C, a −1 value is returned.

NAME
       signal — catch or ignore signals

SYNOPSIS
       (signal = 48.)
       sys signal; sig; label
       (old value in r0)

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

DESCRIPTION
       When the signal defined by *sig* is sent to the current process, it is to be treated according to *label*
       (resp. *func.)* The following is the list of signals:

1	hangup
2	interrupt
3*	quit
4*	illegal instruction
5*	trace trap
6*	IOT instruction
7*	EMT instruction
8*	floating point exception
9	kill (cannot be caught or ignored)
10*	bus error
11*	segmentation violation
12*	bad argument to sys call

       If *label* is 0, the default system action applies to the signal. This is processes termination with or
       without a core dump. If *label* is odd, the signal is ignored. Any other even *label* specifies an ad-
       dress in the process where an interrupt is simulated. An RTI instruction will return from the in-
       terrupt. As a signal is caught, it is reset to 0. Thus if it is desired to catch every such signal, the
       catching routine must issue another *signal* call.

       In C, if *func* is 0 or 1, the action is as described above. If *func* is even, it is assumed to be the
       address of a function entry point. When the signal occurs, the function will be called. A return
       from the function will simulate the RTI.

       The starred signals in the list above cause a core image if not caught or ignored.

       In assembly language, the old value of the signal is returned in r0. In C, that value is returned.

       After a *fork*, the child inherits all signals. The *exec* call resets all caught signals to default action.

SEE ALSO
       kill (I), kill (II)

DIAGNOSTICS
       The error bit (c-bit) is set if the given signal is out of range. In C, a −1 indicates an error; 0 indi-
       cates success.

NAME
        sleep — stop execution for interval

SYNOPSIS
        (sleep = 35.; not in assembler)
        (seconds in r0)
        sys sleep

        sleep(seconds)

DESCRIPTION
        The current process is suspended from execution for the number of seconds specified by the argument.

SEE ALSO
        sleep (I)

DIAGNOSTICS
        —

NAME
     stat — get file status

SYNOPSIS
     (stat = 18.)
     sys stat; name; buf

     stat(name, buf)
     char *name;
     struct inode *buf;

DESCRIPTION
     Name points to a null-terminated string naming a file; *buf* is the address of a 36(10) byte buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be readable. After *stat, buf* has the following structure (starting offset given in bytes):

     struct {
          char   minor;   /* +0: minor device of i-node */
          char   major;   /* +1: major device */
          int   inumber   /* +2 */
          int   flags;   /* +4: see below */
          char   nlinks;   /* +6: number of links to file */
          char   uid;   /* +7: user ID of owner */
          char   gid;   /* +8: group ID of owner */
          char   size0;   /* +9: high byte of 24-bit size */
          int   size1;   /* +10: low word of 24-bit size */
          int   addr[8];   /* +12: block numbers or device number */
          int   actime[2];   /* +28: time of last access */
          int   modtime[2];   /* +32: time of last modification */
     };

     The flags are as follows:

          100000   i-node is allocated
          060000   2-bit file type:
               000000   plain file
               040000   directory
               020000   character-type special file
               060000   block-type special file.
          010000   large file
          004000   set user-ID on execution
          002000   set group-ID on execution
          000400   read (owner)
          000200   write (owner)
          000100   execute (owner)
          000070   read, write, execute (group)
          000007   read, write, execute (others)

SEE ALSO
     ls (I), fstat (II), fs (V)

DIAGNOSTICS
     Error bit (c-bit) is set if the file cannot be found. From C, a −1 return indicates an error.

NAME

stime — set time

SYNOPSIS

(stime = 25.)
(time in r0-r1)
sys stime

stime(tbuf)
int tbuf[2];

DESCRIPTION

Stime sets the system's idea of the time and date. Time is measured in seconds from 0000 GMT Jan 1 1970. Only the super-user may use this call.

SEE ALSO

date (I), time (II), ctime (III)

DIAGNOSTICS

Error bit (c-bit) set if user is not the super-user.

NAME
       stty — set mode of typewriter

SYNOPSIS
       (stty = 31.)
       (file descriptor in r0)
       sys stty; arg

       ...

       arg: speed; 0; mode

       stty(fildes, arg)
       int arg[3];

DESCRIPTION
       Stty sets mode bits and character speeds for the typewriter whose file descriptor is passed in r0
       (resp. is the first argument to the call). First, the system delays until the typewriter is quiescent.
       Then the speed and general handling of the input side of the typewriter is set from the low byte
       of the first word of the *arg*, and the speed of the output side is set from the high byte of the first
       word of the *arg*. The speeds are selected from the following table, which corresponds to the
       speeds supported by the DH-11 interface. If DC-11, DL-11 or KL-11 interfaces are used, impos-
       sible speed changes are ignored.

0	(turn off device)
1	50 baud
2	75 baud
3	110 baud
4	134.5 baud
5	150 baud
6	200 baud
7	300 baud
8	600 baud
9	1200 baud
10	1800 baud
11	2400 baud
12	4800 baud
13	9600 baud
14	External A
15	External B

In the current configuration, only 150 and 300 baud are really supported, in that the code conver-
sion and line control required for 2741's (134.5 baud) must be implemented by the user's pro-
gram, and the half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied.

The second word of the *arg* is currently unused and is available for expansion.

The third word of the *arg* sets the *mode*. It contains several bits which determine the system's
treatment of the typewriter:

10000	~~no delays after tabs (e.g. TN-300)~~
200	even parity allowed on input (e. g. for M37s)
100	odd parity allowed on input
040	raw mode: wake up on all characters
020	map CR into LF; echo LF or CR as CR-LF
010	echo (full duplex)
004	map upper case to lower on input (e. g. M33)
002	echo and print tabs as spaces
001	~~inhibit all function delays (e. g. CRTs)~~ *hangup on last close*

Characters with the wrong parity, as determined by bits 200 and 100, are ignored.

In raw mode, every character is passed back immediately to the program. No erase or kill processing is done; the end-of-file character (EOT), the interrupt character (DELETE) and the quit character (FS) are not treated specially.

Mode 020 causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (used for GE TermiNet 300's and other terminals without the newline function).

SEE ALSO
stty (I), gtty (II)

DIAGNOSTICS
The error bit (c-bit) is set if the file descriptor does not refer to a typewriter. From C, a negative value indicates an error.

NAME

      sync — update super-block

SYNOPSIS

      (sync = 36.; not in assembler)

      sys sync

DESCRIPTION

      Sync causes all information in core memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

      It should be used by programs which examine a file system, for example *check, df,* etc. It is mandatory before a boot.

SEE ALSO

      sync (VIII)

DIAGNOSTICS

      —

NAME

　　time — get date and time

SYNOPSIS

　　(time = 13.)
　　sys time

　　time(tvec)
　　int tvec[2];

DESCRIPTION

　　Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. From *as,* the high order word is in the r0 register and the low order is in r1. From C, the user-supplied vector is filled in.

SEE ALSO

　　date(I), stime(II), ctime(III)

DIAGNOSTICS

NAME

times — get process times

SYNOPSIS

(times = 43.; not in assembler)
sys times; buffer

times(buffer)
struct tbuffer *buffer;

DESCRIPTION

Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/60 seconds.

After the call, the buffer will appear as follows:

```
struct tbuffer {
        int     proc_user_time;
        int     proc_system_time;
        int     child_user_time[2];
        int     child_system_time[2];
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO

time(I)

DIAGNOSTICS

—

BUGS

The process times should be 32 bits as well.

NAME

umount — dismount file system

SYNOPSIS

(umount = 22.)
sys umount; special

DESCRIPTION

Umount announces to the system that special file *special* is no longer to contain a removable file system. The file associated with the special file reverts to its ordinary interpretation (see *mount*).

SEE ALSO

umount (VIII), mount (II)

DIAGNOSTICS

Error bit (c-bit) set if no file system was mounted on the special file or if there are still active files on the mounted file system.

NAME

unlink — remove directory entry

SYNOPSIS

(unlink = 10.)
sys unlink; name

unlink(name)
char *name;

DESCRIPTION

Name points to a null-terminated string. *Unlink* removes the entry for the file pointed to by *name* from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

SEE ALSO

rm (I), rmdir (I), link (II)

DIAGNOSTICS

The error bit (c-bit) is set to indicate that the file does not exist or that its directory cannot be written. Write permission is not required on the file itself. It is also illegal to unlink a directory (except for the super-user). From C, a −1 return indicates an error.

NAME
    wait — wait for process to die

SYNOPSIS
    (wait = 7.)
    sys wait

    wait(status)
    int *status;

What does wait(1) return?

Where

DESCRIPTION
    Wait causes its caller to delay until one of its child processes terminates. If any child has died
    since the last *wait,* return is immediate; if there are no children, return is immediate with the er-
    ror bit set (resp. with a value of −1 returned). In the case of several children several *wait* calls
    are needed to learn of all the deaths.

    If no error is indicated on return, the r1 high byte (resp. the high byte stored into *status*) con-
    tains the low byte of the child process r0 (resp. the argument of *exit*) when it terminated. The
    r1 (resp. *status*) low byte contains the termination status of the process. See signal (II) for a list
    of termination statuses (signals); 0 status indicates normal termination. If the 0200 bit of the ter-
    mination status is set, a core image of the process was produced by the system.

    If the parent process terminates without waiting on its children, the initialization process (process
    ID = 1) inherits the children.

SEE ALSO
    exit (II), fork (II), signal (II), *kill (I)*

DIAGNOSTICS
    The error bit (c-bit) is set if there are no children not previously waited for. From C, a returned
    value of −1 indicates an error.

NAME
        write — write on a file

SYNOPSIS
        (write = 4.)
        (file descriptor in r0)
        sys write; buffer; nbytes

        write(fildes, buffer, nbytes)
        char *buffer;

DESCRIPTION
        A file descriptor is a word returned from a successful *open, creat* or *pipe* call.

        Buffer is the address of *nbytes* contiguous bytes which are written on the output file. The number
        of characters actually written is returned (in r0). It should be regarded as an error if this is not
        the same as requested.

        Writes which are multiples of 512 characters long and begin on a 512-byte boundary are more
        efficient than any others.

SEE ALSO
        creat(II), open(II), pipe(II)

DIAGNOSTICS
        The error bit (c-bit) is set on an error: bad descriptor, buffer address, or count; physical I/O errors.
        From C, a returned value of —1 indicates an error.

NAME
     alloc — core allocator

SYNOPSIS
     char *alloc(size)

     free(ptr)
     char *ptr;

DESCRIPTION
     Alloc and *free* provide a simple general-purpose core management package. *Alloc* is given a size in bytes; it returns a pointer to an area at least that size which is even and hence can hold an object of any type. The argument to *free* is a pointer to an area previously allocated by *alloc;* this space is made available for further allocation.

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

     The routine uses a first-fit algorithm which coalesces blocks being freed with other blocks already free. It calls *sbrk* (see *break (II)*) to get more core from the system when there is no suitable space already free, and writes "Out of space" on the standard output, then exits, if that fails.

     The external variable *slop* (which is 2 if not set) is a number such that if *n* bytes are requested, and if the first free block of size at least *n* is no larger than *n+slop*, then the whole block will be allocated instead of being split up. Larger values of *slop* tend to reduce fragmentation at the expense of unused space in the allocated blocks.

DIAGNOSTICS
     "Out of space" if it needs core and can't get it.

BUGS

NAME

     atan − arc tangent function

SYNOPSIS

     jsr   r5,atan[2]

     double atan(x)
     double x;

     double atan2(x, y)
     double x, y;

DESCRIPTION

     The *atan* entry returns the arc tangent of fr0 in fr0; from C, the arc tangent of x is returned. The range is $-\pi/2$ to $\pi/2$. The *atan2* entry returns the arc tangent of fr0/fr1 in fr0; from C, the arc tangent of x/y is returned. The range is $-\pi$ to π.

DIAGNOSTIC

     There is no error return.

BUGS

NAME
    atof — ascii to floating

SYNOPSIS
    double atof(nptr)
    char *nptr;

DESCRIPTION
    Atof converts a string to a floating number. *Nptr* should point to a string containing the number; the first unrecognized character ends the number.

    The only numbers recognized are: an optional minus sign followed by a string of digits optionally containing one decimal point, then followed optionally by the letter e followed by a signed integer.

DIAGNOSTICS
    There are none; overflow results in a very large number and garbage characters terminate the scan.

BUGS
    The routine should accept initial +, initial blanks, and E for e. Overflow should be signalled.

NAME
        crypt — password encoding

SYNOPSIS
        mov $key,r0
        jsr   pc,crypt

        char *crypt(key)
        char *key;

DESCRIPTION
        On entry, r0 should point to a string of characters terminated by an ASCII NULL. The routine performs an operation on the key which is difficult to invert (i.e. encrypts it) and leaves the resulting eight bytes of ASCII alphanumerics in a global cell called "word".

        From C, the *key* argument is a string and the value returned is a pointer to the eight-character encrypted password.

        Login uses this result as a password.

SEE ALSO
        passwd(I), passwd(V), login(I)

NAME

ctime — convert date and time to ASCII .

SYNOPSIS

char *ctime(tvec)
int tvec[2];

[from Fortran]
double precision ctime
... = ctime(dummy)

int *localtime(tvec)
int tvec[2];

int *gmtime(tvec)
int tvec[2];

DESCRIPTION

Ctime converts a time in the vector *tvec* such as returned by time (II) into ASCII and returns a pointer to a character string in the form

Sun Sep 16 01:03:52 1973\n\0

All the fields have constant width.

Once the time has been placed into *t* and *t+2*, this routine is callable from assembly language as follows:

```
mov     $t,—(sp)
jsr     pc,_ctime
tst     (sp)+
```

and a pointer to the string is available in r0.

The *localtime* and *gmtime* entries return pointers to integer vectors containing the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time UNIX uses. The value is a pointer to an array whose components are

0   seconds
1   minutes
2   hours
3   day of the month (1-31)
4   month (0-11)
5   year — 1900
6   day of the week (Sunday = 0)
7   day of the year (0-365)
8   Daylight Saving Time flag if non-zero

The external variable *timezone* contains the difference, in seconds, between GMT and local standard time (in EST, is 5*60*60); the external variable *daylight* is non-zero iff the standard U.S.A. Daylight Saving Time conversion should be applied between the last Sundays in April and October. The external variable *nixonflg* if non-zero supersedes *daylight* and causes daylight time all year round.

A routine named *ctime* is also available from Fortran. Actually it more resembles the *time* (II) system entry in that it returns the number of seconds since the epoch 0000 GMT Jan. 1, 1970 (as a floating-point number).

SEE ALSO

time(II)

BUGS

NAME

ecvt — output conversion

SYNOPSIS

```
jsr     pc,ecvt

jsr     pc,fcvt
```

char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
...

DESCRIPTION

Ecvt is called with a floating point number in fr0.

On exit, the number has been converted into a string of ascii digits in a buffer pointed to by r0. The number of digits produced is controlled by a global variable *_ndigits*.

Moreover, the position of the decimal point is contained in r2: r2=0 means the d.p. is at the left hand end of the string of digits; r2>0 means the d.p. is within or to the right of the string.

The sign of the number is indicated by r1 (0 for +; 1 for −).

The low order digit has suffered decimal rounding (i. e. may have been carried into).

From C, the *value* is converted and a pointer to a null-terminated string of *ndigit* digits is returned. The position of the decimal point is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero.

Fcvt is identical to *ecvt*, except that the correct digit has had decimal rounding for F-style output of the number of digits specified by *ndigits*.

SEE ALSO

printf(III)

BUGS

NAME
        exp — exponential function

SYNOPSIS
        jsr   r5,exp

        double exp(x)
        double x;

DESCRIPTION
        The exponential of fr0 is returned in fr0. From C, the exponential of x is returned.

DIAGNOSTICS
        If the result is not representable, the c-bit is set and the largest positive number is returned. From C, no diagnostic is available.

        Zero is returned if the result would underflow.

BUGS

NAME
        floor — floor and ceiling functions

SYNOPSIS
        double floor(x)
        double x;

        double ceil(x)
        double x;

DESCRIPTION
        The floor function returns the largest integer (as a double precision number) not greater than x.

        The ceil function returns the smallest integer not less than x.

BUGS

NAME

    fptrap — floating point interpreter

SYNOPSIS

    sys   signal; 4; fptrap

DESCRIPTION

    Fptrap is a simulator of the 11/45 FP11-B floating point unit. It works by intercepting illegal in-
    struction faults and examining the offending operation codes for possible floating point.

FILES

    found in /lib/libu.a; a fake version is in /lib/liba.a

DIAGNOSTICS

    A break point trap is given when a real illegal instruction trap occurs.

SEE ALSO

    signal(II)

BUGS

    Rounding mode is not interpreted. It's slow.

NAME

gamma — log gamma function

SYNOPSIS

    jsr   r5,gamma

    double gamma(x)
    double x;

DESCRIPTION

If x is passed (in fr0) *gamma* returns $\ln|\Gamma(x)|$ (in fr0). The sign of $\Gamma(x)$ is returned in the external integer *signgam*. The following C program might be used to calculate Γ:

```
y = gamma(x);
if (y > 88.)
        error( );
y = exp(y);
if(signgam)
        y = −y;
```

DIAGNOSTICS

The c-bit is set on negative integral arguments and the maximum value is returned. There is no error return for C programs.

BUGS

No error return from C.

NAME
     getarg — get command arguments from Fortran

SYNOPSIS
     call getarg (i, iarray [, isize])

     ... = iargc(dummy)

DESCRIPTION
     The *getarg* entry fills in *iarray* (which is considered to be *integer*) with the Hollerith string representing the *i* th argument to the command in which it it is called. If no *isize* argument is specified, at least one blank is placed after the argument, and the last word affected is blank padded. The user should make sure that the array is big enough.

     If the *isize* argument is given, the argument will be followed by blanks to fill up *isize* words, but even if the argument is long no more than that many words will be filled in.

     The blank-padded array is suitable for use as an argument to setfil (III).

     The *iargc* entry returns the number of arguments to the command, counting the first (file-name) argument.

SEE ALSO
     exec (II), setfil (III)

BUGS

NAME

    getc — buffered input

SYNOPSIS

```
·mov    $filename,r0
jsr     r5,fopen; iobuf

fopen(filename, iobuf)
char *filename;
struct buf *iobuf;

jsr     r5,getc; iobuf
(character in r0)

getc(iobuf)
struct buf *iobuf;

jsr     r5,getw; iobuf
(word in r0)

getw(iobuf)
struct buf *iobuf;
```

DESCRIPTION

These routines provide a buffered input facility. *Iobuf* is the address of a 518(10) byte buffer area whose contents are maintained by these routines. Its format is:

```
ioptr:   .=.+2         / file descriptor
         .=.+2         / characters left in buffer
         .=.+2         / ptr to next character
         .=.+512.      / the buffer
```

Or in C,

```
struct buf {
        int fildes;
        int nleft;
        char *nextp;
        char buffer[512];
};
```

Fopen may be called initially to open the file. On return, the error bit (c-bit) is set if the open failed. If *fopen* is never called, *get* will read from the standard input file. From C, the value is negative if the open failed.

Getc returns the next byte from the file in r0. The error bit is set on end of file or a read error. From C, the character is returned; it is −1 on end-of-file or error.

Getw returns the next word in r0. *Getc* and *getw* may be used alternately; there are no odd/even problems. *Getw* is may be called from C; −1 is returned on end-of-file or error, but of course is also a legitimate value.

Iobuf must be provided by the user; it must be on a word boundary.

To reuse the same buffer for another file, it is sufficient to close the original file and call *fopen* again.

SEE ALSO

    open(II), read(II), putc(III)

DIAGNOSTICS

    c-bit set on EOF or error;
    from C, negative return indicates error or EOF.

NAME
        getchar — read character

SYNOPSIS
        getchar( )

DESCRIPTION
        Getchar provides the simplest means of reading characters from the standard input for C programs. It returns successive characters until end-of-file, when it returns "\0".

        Associated with this routine is an external variable called *fin*, which is a structure containing a buffer such as described under *getc* (III).

        Generally speaking, *getchar* should be used only for the simplest applications; *getc* is better when there are multiple input files.

SEE ALSO
        getc (III)

DIAGNOSTICS
        Null character returned on EOF or error.

BUGS
        —1 should be returned on EOF; null is a legitimate character.

NAME
        getpw — get name from UID

SYNOPSIS
        getpw(uid, buf)
        char *buf;

DESCRIPTION
        Getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding
        line; it returns non-zero if *uid* could not be found. The line is null-terminated.

FILES
        /etc/passwd

SEE ALSO
        passwd(V)

DIAGNOSTICS
        non-zero return on error.

BUGS

NAME

hmul — high-order product

SYNOPSIS

hmul(x, y)

DESCRIPTION

Hmul returns the high-order 16 bits of the product of x and y. (The binary multiplication operator generates the low-order·16 bits of a product.)

NAME

      hypot — calculate hypotenuse

SYNOPSIS

      jsr   r5,hypot

DESCRIPTION

      The square root of $fr0 \times fr0 + fr1 \times fr1$ is returned in fr0. The calculation is done in such a way that overflow will not occur unless the answer is not representable in floating point.

DIAGNOSTICS

      The c-bit is set if the result cannot be represented.

BUGS

NAME

ierror — catch Fortran errors

SYNOPSIS

if (ierror (*errno*) .ne. 0) goto *label*

DESCRIPTION

Ierror provides a way of detecting errors during the running of a Fortran program. Its argument is a run-time error number such as enumerated in *fc* (I).

When *ierror* is called, it returns a 0 value; thus the **goto** statement in the synopsis is not executed. However, the routine stores inside itself the call point and invocation level. If and when the indicated error occurs, a **return** is simulated from *ierror* with a non-zero value; thus the **goto** (or other statement) is executed. It is a ghastly error to call *ierror* from a subroutine which has already returned when the error occurs.

This routine is essentially tailored to catching end-of-file situations. Typically it is called just before the start of the loop which reads the input file, and the **goto** jumps to a graceful termination of the program.

There is a limit of 5 on the number of different error numbers which can be caught.

SEE ALSO

fc (I)

BUGS

There is no way to ignore errors.

NAME

ldiv — long division

SYNOPSIS

ldiv(hidividend, lodividend, divisor)

lrem(hidividend, lodividend, divisor)

DESCRIPTION

The concatenation of the signed 16-bit *hidividend* and the unsigned 16-bit *lodividend* is divided by *divisor*. The 16-bit signed quotient is returned by *ldiv* and the 16-bit signed remainder is returned by *lrem*. Divide check and erroneous results will occur unless the magnitude of the divisor is greater than that of the high-order dividend.

An integer division of an unsigned dividend by a signed divisor may be accomplished by

quo = ldiv(0, dividend, divisor);

and similarly for the remainder operation.

Often both the quotient and the remainder are wanted. Therefore *ldiv* leaves a remainder in the external cell *ldivr*.

BUGS

No divide check check.

NAME
> locv — long output conversion

SYNOPSIS
> **char *locv(hi, lo)**
> **int hi, lo;**

DESCRIPTION
> *Locv* converts a signed double-precision integer, whose parts are passed as arguments, to the equivalent ASCII character string and returns a pointer to that string.

BUGS

NAME

      log — natural logarithm

SYNOPSIS

      jsr   r5,log

      double log(x)
      double x;

DESCRIPTION

      The natural logarithm of fr0 is returned in fr0. From C, the natural logarithm of x is returned.

DIAGNOSTICS

      The error bit (c-bit) is set if the input argument is less than or equal to zero and the result is a negative number very large in magnitude. From C, there is no error indication.

NAME

monitor — prepare execution profile

SYNOPSIS

monitor(lowpc, highpc, buffer, bufsize)
int lowpc( ), highpc( ), buffer[ ], bufsize;

DESCRIPTION

Monitor is an interface to the system's profile entry (II). *Lowpc* and *highpc* are the names of two functions; *buffer* is the address of a (user supplied) array of *bufsize* integers. *Monitor* arranges for the system to sample the user's program counter periodically and record the execution histogram in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc*. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

extern etext;

...
monitor(2, &etext, buf, bufsize);

Etext is a loader-defined symbol which lies just above all the program text.

To stop execution monitoring and write the results on the file *mon.out*, use

monitor(0);

Then, when the program exits, prof (I) can be used to examine the results.

It is seldom necessary to call this routine directly; the **—p** option of *cc* is simpler if one is satisfied with its default profile range and resolution.

FILES

mon.out

SEE ALSO

prof (I), profil (II), cc (I)

NAME
     nargs — argument count

SYNOPSIS
     nargs( )

DESCRIPTION
     Nargs returns the number of actual parameters supplied by the caller of the routine which calls *nargs*.

     The argument count is accurate only when none of the actual parameters is *float* or *double*. Such parameters count as four arguments instead of one.

BUGS
     As indicated.

NAME

nlist — get entries from name list

SYNOPSIS

jsrr5,nlist; file; list

...

file: <file name\0>; .even
list: <name1xxx>; type1; value1
        <name2xxx>; type2; value2

...

0

nlist(filename, nl)
char *filename;
struct {
      char   name[8];
      int   type;
      int   value;
} nl[ ];

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of a list of 8-character names (null padded) each followed by two words. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are placed in the two words following the name. If the name is not found, the type entry is set to −1.

This subroutine is useful for examining the system name list kept in the file **/unix**. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out (V)

DIAGNOSTICS

All type entries are set to −1 if the file cannot be found or if it is not a valid namelist.

BUGS

NAME
       perror — system error messages

SYNOPSIS
       perror(s)
       char *s;

       int sys_nerr;
       char *sys_errlist[];

       int errno;

DESCRIPTION
       Perror produces a short error message describing the last error encountered during a call to the
       system from a C program. First the argument string *s* is printed, then a colon, then the message
       and a new-line. Most usefully, the argument string is the name of the program which incurred
       the error. The error number is taken from the external variable *errno*, which is set when errors
       occur but not cleared when non-erroneous calls are made.

       To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided;
       errno can be used as an index in this table to get the message string without the newline.
       Sys_nerr is the largest message number provided for in the table; it should be checked because
       new error codes may be added to the system before they are added to the table.

SEE ALSO
       Introduction to System Calls

BUGS

NAME
        pow — floating exponentiation

SYNOPSIS
        movf   x,fr0
        movf   y,fr1
        jsr   pc,pow

        double pow(x,y)
        double x, y;

DESCRIPTION
        Pow returns the value of x^y (in fr0). *Pow(0, y)* is 0 for any *y*. *Pow(−x, y)* returns a result only if *y* is an integer.

SEE ALSO
        exp(III), log(III)

DIAGNOSTICS
        The carry bit is set on return in case of overflow, *pow(0, 0)*, or *pow(−x, y)* for non-integral *y*. From C there is no diagnostic.

BUGS

NAME
        printf — formatted print

SYNOPSIS
        printf(format, arg$_1$, ...);
        char *format;

DESCRIPTION
        Printf converts, formats, and prints its arguments after the first under control of the first argu-
        ment. The first argument is a character string which contains two types of objects: plain charac-
        ters, which are simply copied to the output stream, and conversion specifications, each of which
        causes conversion and printing of the next successive argument to *printf*.

        Each conversion specification is introduced by the character %. Following the %, there may be

        — an optional minus sign "—" which specifies *left adjustment* of the converted argument in
          the indicated field;

        — an optional digit string specifying a *field width;* if the converted argument has fewer char-
          acters than the field width it will be blank-padded on the left (or right, if the left-
          adjustment indicator has been given) to make up the field width;

        — an optional period "." which serves to separate the field width from the next digit string;

        — an optional digit string *(precision)* which specifies the number of digits to appear after the
          decimal point, for e- and f-conversion, or the maximum number of characters to be print-
          ed from a string;

        — a character which indicates the type of conversion to be applied.

        The conversion characters and their meanings are

        d
        o
        x   The (integral) argument is converted to decimal, octal, or hexadecimal notation respective-
            ly.

        f   The argument is converted to decimal notation in the style "[—]ddd.ddd" where the
            number of d's after the decimal point is equal to the precision specification for the argu-
            ment. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no di-
            gits and no decimal point are printed. The argument should be *float* or *double*.

        e   The argument is converted in the style "[—]d.ddde±dd" where there is one digit before
            the decimal point and the number after is equal to the precision specification for the
            argument; when the precision is missing, 6 digits are produced. The argument should be a
            float or *double* quantity.

        c   The argument character or character-pair is printed if non-null.

        s   The argument is taken to be a string (character pointer) and characters from the string are
            printed until a null character or until the number of characters indicated by the precision
            specification is reached; however if the precision is 0 or missing all characters up to a null
            are printed.

        l   The argument is taken to be an unsigned integer which is converted to decimal and print-
            ed (the result will be in the range 0 to 65535).

        If no recognizable character appears after the %, that character is printed; thus % may be printed
        by use of the string %%. In no case does a non-existent or small field width cause truncation of
        a field; padding takes place only if the specified field width exceeds the actual width. Characters
        generated by *printf* are printed by calling *putchar.*

SEE ALSO
        putchar (III)

- 1 -

NAME
            putc — buffered output

SYNOPSIS
            mov $filename,r0
            jsr   r5,fcreat; iobuf

            fcreat(file, iobuf)
            char *file;
            struct buf *iobuf;

            (get byte in r0)
            jsr   r5,putc; iobuf

            putc(c, iobuf)
            int c;
            struct buf *iobuf;

            (get word in r0)
            jsr   r5,putw; iobuf

            putw(w, iobuf);
            int w;
            struct buf *iobuf;

            jsr   r5,flush; iobuf

            fflush(iobuf)
            struct buf *iobuf;

DESCRIPTION
            Fcreat creates the given file (mode 666) and sets up the buffer *iobuf* (size 518 bytes); *putc* and *putw*
            write a byte or word respectively onto the file; *flush* forces the contents of the buffer to be writ-
            ten, but does not close the file. The format of the buffer is:

            iobuf: .=.+2   / file descriptor
                       .=.+2   / characters unused in buffer
                       .=.+2   / ptr to next free character
                       .=.+512.   / buffer

            Or in C,

            struct buf {
                       int fildes;
                       int nunused;
                       char *nxtfree;
                       char buff[512];
            };

            Fcreat sets the error bit (c-bit) if the file creation failed (from C, returns −1); none of the other
            routines returns error information.

            Before terminating, a program should call *flush* to force out the last of the output *(fflush* from C).

            The user must supply *iobuf,* which should begin on a word boundary.

            To write a new file using the same buffer, it suffices to call *[f]flush,* close the file, and call *fcreat*
            again.

SEE ALSO
            creat(II), write(II), getc(III)

DIAGNOSTICS
            error bit possible on *fcreat* call.

NAME
putchar — write character

SYNOPSIS
putchar(ch)

flush( )

DESCRIPTION
Putchar writes out its argument and returns it unchanged. Only the low-order byte is written, and only if it is non-null. Unless other arrangements have been made, *putchar* writes in unbuffered fashion on the standard output file.

Associated with this routine is an external variable *fout* which has the structure of a buffer discussed under putc (III). If the file descriptor part of this structure (first word) is greater than 2, output via *putchar* is buffered. To achieve buffered output one may say, for example,

            fout = dup(1);   or
            fout = creat(...);

In such a case *flush* must be called before the program terminates in order to flush out the buffered output. *Flush* may be called at any time.

SEE ALSO
putc(III)

BUGS
The *fout* notion is kludgy.

NAME
      qsort — quicker sort

SYNOPSIS
      (base of data in r1)
      (end+1 of data in r2)
      (element width in r3)
      jsr pc,qsort

      qsort(base, nel, width, compar)
      char *base;
      int (*compar)( );

DESCRIPTION
      Qsort is an implementation of the quicker-sort algorithm. The assembly-language version is designed to sort equal length elements. Registers r1 and r2 delimit the region of core containing the array of byte strings to be sorted: r1 points to the start of the first string, r2 to the first location above the last string. Register r3 contains the length of each string. r2−r1 should be a multiple of r3. On return, r0, r1, r2, r3 are destroyed.

      The C version has somewhat different arguments and the user must supply a comparison routine. The first argument is to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return a negative integer if the first element is to be considered less than the second, a positive integer if the second element is smaller than the first, and 0 if the elements are equal.

SEE ALSO
      sort (I)

BUGS

NAME
    rand — random number generator

SYNOPSIS
    (seed in r0)
    jsr   pc,srand   /to initialize

    jsr   pc,rand /to get a random number

    srand(seed)
    int seed;

    rand( )

DESCRIPTION
    Rand uses a multiplicative congruential random number generator to return successive pseudo-random numbers (in r0) in the range from 1 to $2^{15}-1$.

    The generator is reinitialized by calling *srand* with 1 as argument (in r0). It can be set to a random starting point by calling *srand* with whatever you like as argument, for example the low-order word of the time.

BUGS
    The low-order bits are not very random.

NAME

      reset — execute non-local goto

SYNOPSIS

      setexit( )

      reset( )

DESCRIPTION

These routines are useful for dealing with errors discovered in a low-level subroutine of a program.

Setexit is typically called just at the start of the main loop of a processing program. It stores certain parameters such as the call point and the stack level.

Reset is typically called after diagnosing an error in some subprocedure called from the main loop. When *reset* is called, it pops the stack appropriately and generates a non-local return from the last call to *setexit*.

It is erroneous, and generally disastrous, to call *reset* unless *setexit* has been called in a routine which is an ancestor of *reset*.

BUGS

NAME
        setfil — specify Fortran file name

SYNOPSIS
        call setfil (unit, hollerith-string)

DESCRIPTION
        Setfil provides a primitive way to associate an integer I/O *unit* number with a file named by the *hollerith-string*. The end of the file name is indicated by a blank. Subsequent I/O on this unit number will refer to the file whose name is specified by the string.

        Setfil should be called only before any I/O has been done on the *unit*, or just after doing a **rewind** or **endfile**. It is ineffective for unit numbers 5 and 6.

SEE ALSO
        fc (I)

BUGS
        The exclusion of units 5 and 6 is unwarranted.

NAME

    sin — sine, cosine

SYNOPSIS

    jsr   r5,sin (cos)

    double sin(x)
    double x;

    double cos(x)
    double x;

DESCRIPTION

    The sine (cosine) of fr0 (resp. x), measured in radians, is returned (in fr0).

    The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

BUGS

NAME
    sqrt — square root function

SYNOPSIS
    jsr   r5,sqrt

    double sqrt(x)
    double x;

DESCRIPTION
    The square root of fr0 (resp. x) is returned (in fr0).

DIAGNOSTICS
    The c-bit is set on negative arguments and 0 is returned. There is no error return for C programs.

BUGS
    No error return from C.

NAME

ttyn — return name of current typewriter

SYNOPSIS

jsr   pc,ttyn

ttyn(file)

DESCRIPTION

Ttyn hunts up the last character of the name of the typewriter which is the standard input (from *as*) or is specified by the argument *file* descriptor (from C). If *n* is returned, the typewriter name is then "/dev/tty*n*".

x is returned if the indicated file does not correspond to a typewriter.
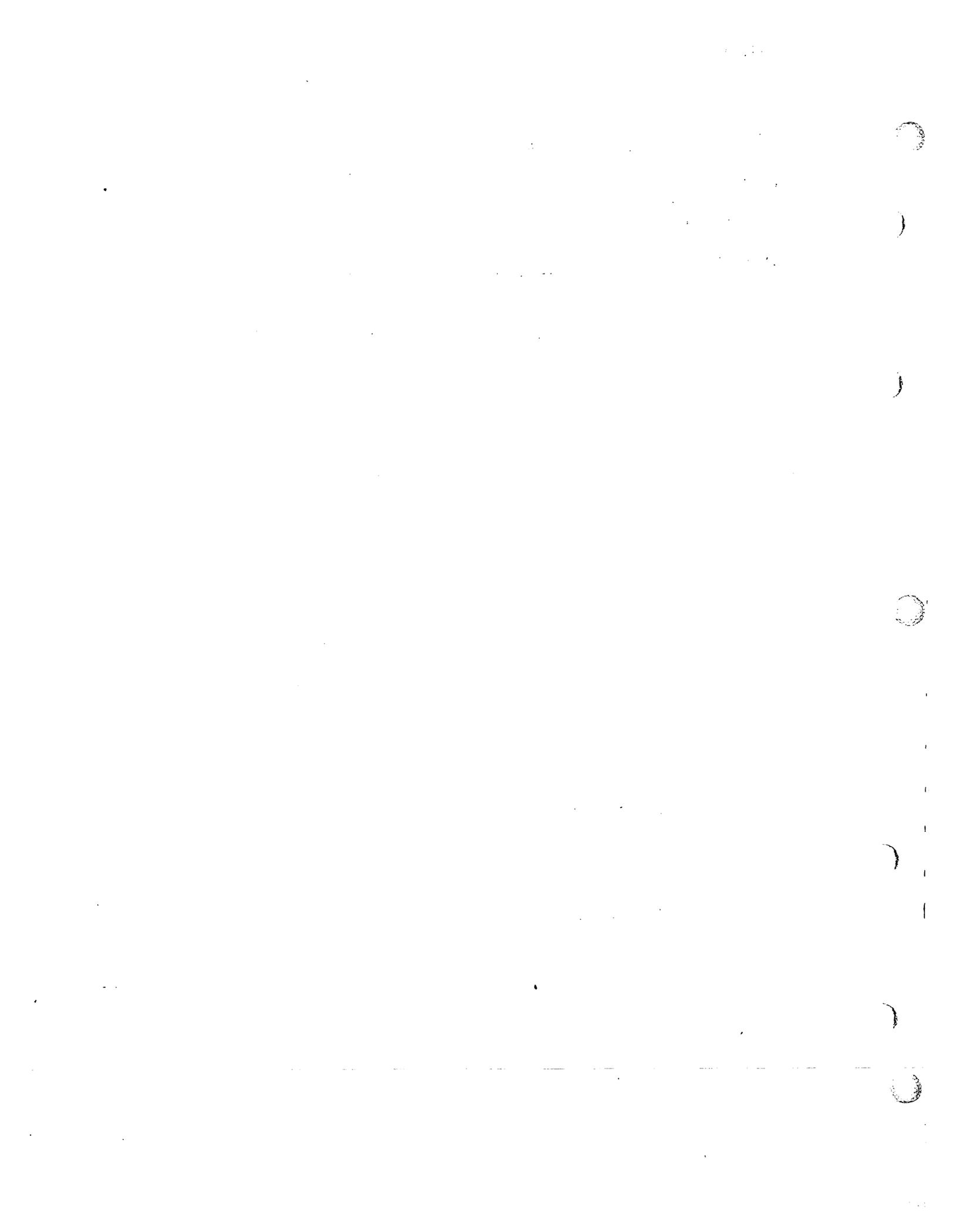
NAME
          vt — display (vt01) interface

SYNOPSIS
          openvt( )

          erase( )

          label(s)
          char s[ ];

          line(x,y)

          circle(x,y,r)

          arc(x,y,x0,y0,x1,y1)

          dot(x,y,dx,n,pattern)
          int pattern[ ];

          move(x,y)

DESCRIPTION
          C interface routines to perform similarly named functions described in vt(IV). *Openvt* must be
          used before any of the others to open the storage scope for writing.

FILES
          /dev/vt0, found in /lib/libp.a

SEE ALSO
          vt (IV)

BUGS

NAME

    cat — phototypesetter interface

DESCRIPTION

    Cat provides the interface to a Graphic Systems C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding will not be described here.

    Only one process may have this file open at a time. It is write-only.

FILES

    /dev/cat

SEE ALSO

    troff (I), Graphic Systems specification (available on request)

BUGS

NAME

        dc — DC-11 communications interface

DESCRIPTION

        The discussion of typewriter I/O given in tty (IV) applies to these devices.

        The DC-11 typewriter interface operates at any of four speeds, independently settable for input and output. The speed is selected by the same encoding used by the DH (IV) device (enumerated in stty (II)); impossible speed changes are ignored.

FILES

        /dev/tty[01234567abcd]   113B Dataphones (not currently connected— see dh (IV))

SEE ALSO

        tty (IV), stty (II), dh (IV)

BUGS

NAME
    dh — DH-11 communications multiplexer

DESCRIPTION
    Each line attached to the DH-11 communications multiplexer behaves as described in tty (IV).
    Input and output for each line may independently be set to run at any of 16 speeds; see stty (II)
    for the encoding.

FILES
    /dev/tty[f-u]

SEE ALSO
    tty (IV), stty (II)

BUGS

NAME

        dn — dn11 ACU interface

DESCRIPTION

        The *dn?* files are write-only. The permissible codes are:

            0-9 dial 0-9
            :   dial *
            ;   dial #
            —   4 second delay for second dial tone
            =   end-of-number

        The entire telephone number must be presented in a single *write* system call.

        It is recommended that an end-of-number code be given even though not all ACU's actually require it.

FILES

        /dev/dn0   connected to 801 with dp0
        /dev/dn1   not currently connected
        /dev/dn2   not currently connected

SEE ALSO

        dp (IV)

BUGS

NAME

    dp — dp11 201 data-phone interface

DESCRIPTION

    The *dp0* file is a 201 data-phone interface. *Read* and *write* calls to dp0 are limited to a maximum of 512 bytes. Each write call is sent as a single record. Seven bits from each byte are written along with an eighth odd parity bit. The sync must be user supplied. Each read call returns characters received from a single record. Seven bits are returned unaltered; the eighth bit is set if the byte was not received in odd parity. A 10 second time out is set and a zero-byte record is returned if nothing is received in that time.

FILES

    /dev/dp0

SEE ALSO

    dn (IV), gerts (III)

BUGS

NAME
    kl — KL-11 or DL-11 asynchronous interface

DESCRIPTION
    The discussion of typewriter I/O given in tty (IV) applies to these devices.

    Since they run at a constant speed, attempts to change the speed via stty (II) are ignored.

    The on-line console typewriter is interfaced using a KL-11 or DL-11. By appropriate switch settings during a reboot, UNIX will come up as a single-user system with I/O on the console typewriter.

FILES
    /dev/tty
    /dev/tty8   synonym for /dev/tty
    /dev/tty9   second console (not currently connected)

SEE ALSO
    tty (IV), init (VIII)

BUGS
    Modem control for the DL-11E is not implemented.

NAME

lp — line printer

DESCRIPTION

Lp provides the interface to any of the standard DEC line printers. When it is opened or closed, a suitable number of page ejects are generated. Bytes written are printed.

An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

{	+←
}	+→
`	+'
		+|
~	+~

The driver correctly interprets carriage returns, backspaces, tabs, and form feeds. A sequence of newlines which extends over the end of a page is turned into a form feed. Lines longer than 80 characters are truncated. (This number is a parameter in the driver.)

FILES

/dev/lp

SEE ALSO

lpr (I)

BUGS

Half-ASCII mode and the maximum line length should be settable by a call analogous to stty (II).

NAME

        mem — core memory

DESCRIPTION

        Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system using the debugger.

        A memory address is an 18-bit quantity which is used directly as a UNIBUS address. References to non-existent locations cause errors to be returned.

        Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

        The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed. In particular, the I/O area of *kmem* is located beginning at 160000 (octal) rather than at 760000. The 1K region beginning at 140000 (octal) is the system's data for the current process.

        The file *null* returns end-of-file on *read* and ignores write.

FILES

        /dev/mem, /dev/kmem, /dev/null

NAME

       pc — PC-11 paper tape reader/punch

DESCRIPTION

       Ppt refers to the PC-11 paper tape reader or punch, depending on whether it is read or written.

       When *ppt* is opened for writing, a 100-character leader is punched. Thereafter each byte written is punched on the tape. No editing of the characters is performed. When the file is closed, a 100-character trailer is punched.

       When *ppt* is opened for reading, the process waits until tape is placed in the reader and the reader is on-line. Then requests to read cause the characters read to be passed back to the program, again without any editing. This means that several null leader characters will usually appear at the beginning of the file. Likewise several nulls are likely to appear at the end. End-of-file is generated when the tape runs out.

       Seek calls for this file are meaningless.

FILES

       /dev/ppt

BUGS

       If both the reader and the punch are open simultaneously, the trailer is sometimes not punched. Sometimes the reader goes into a dead state in which it cannot be opened.

NAME
        rf — RF11/RS11 fixed-head disk file

DESCRIPTION
        This file refers to the concatenation of all RS-11 disks.

        Each disk contains 1024 256-word blocks. The length of the combined RF file is
        1024×(minor+1) blocks. That is minor device zero is 1024 blocks long; minor device one is 2048,
        etc.

        The *rf0* file accesses the disk via the system's normal buffering mechanism and may be read and
        written without regard to physical disk records. There is also a "raw" interface which provides
        for direct transmission between the disk and the user's read or write buffer. A single read or
        write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient
        when many words are transmitted. The name of the raw RF file is *rrf0*. The same minor device
        considerations hold for the raw interface as for the normal interface.

        In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512
        bytes (a disk block). Likewise *seek* calls should specify a multiple of 512 bytes.

FILES
        /dev/rf0, /dev/rrf0

BUGS
        The 512-byte restrictions are not physically necessary, but are still imposed.

NAME

    rk — RK-11/RK03 (or RK05) disk

DESCRIPTION

    Rk? refers to an entire RK03 disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 4871.

    Drive numbers (minor devices) of eight and greater are treated specially. Drive $8+x$ is the $x+1$ way interleaving of devices rk0 to rk$x$. Thus blocks on rk10 are distributed alternately among rk0, rk1, and rk2.

    The *rk* files discussed above access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RK files begin with *rrk* and end with a number which selects the same disk as the corresponding *rk* file.

    In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *seek* calls should specify a multiple of 512 bytes.

FILES

    /dev/rk?, /dev/rrk?

BUGS

    Care should be taken in using the interleaved files. First, the same drive should not be accessed simultaneously using the ordinary name and as part of an interleaved file, because the same physical blocks have in effect two different names; this fools the system's buffering strategy. Second, the combined files cannot be used for swapping or raw I/O.

NAME

    rp — RP-11/RP03 moving-head disk

DESCRIPTION

    The files *rp0 ... rp7* refer to sections of RP disk drive 0. The files *rp8 ... rp15* refer to drive 1 etc. This is done since the size of a full RP drive is 81200 blocks and internally the system is only capable of addressing 65536 blocks. Also since the disk is so large, this allows it to be broken up into more manageable pieces.

    The origin and size of the pseudo-disks on each drive are as follows:

disk	start	length
0	0	40600
1	40600	40600
2	0	9200
3	72000	9200
4	0	65535
5	15600	65535
6-7	unassigned	

    It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. Here is a suggestion for two useful configurations: If the root of the file system is on some other device and the RP used as a mounted device, then *rp0* and *rp1,* which divide the disk into two equal size portions, is a good idea. Other things being equal, it is advantageous to have two equal-sized portions since one can always be copied onto the other, which is occasionally useful.

    If the RP is the only disk and has to contain the root and the swap area, the root can be put on *rp2* and a mountable file system on *rp5.* Then the swap space can be put in the unused blocks 9200 to 15600 of *rp0* (or, equivalently, *rp4).* This arrangement puts the root file system, the swap area, and the i-list of the mounted file system relatively near each other and thus tends to minimize head movement.

    The *rp* access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with *rrp* and end with a number which selects the same disk section as the corresponding *rp* file.

    In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise *seek* calls should specify a multiple of 512 bytes.

FILES

    /dev/rp?, /dev/rrp?

BUGS

NAME
          tc − TC-11/TU56 DECtape

DESCRIPTION
          The files *tap0 ... tap7* refer to the TC-11/TU56 DECtape drives 0 to 7.

          The 256-word blocks on a standard DECtape are numbered 0 to 577.

FILES
          /dev/tap?

SEE ALSO
          tp (I)

BUGS
          Since reading is synchronous, only one block is picked up per tape reverse.

NAME
        tiu — Spider interface

DESCRIPTION
        Spider is a fast digital switching network. *Tiu* is a directory which contains files each referring to
        a Spider control or data channel. The file /dev/tiu/dn refers to data channel n, likewise
        /dev/tiu/cn refers to control channel n.

        The precise nature of the UNIX interface has not been defined yet.

FILES
        /dev/tiu/d?, /dev/tiu/c?

BUGS

NAME

tm — TM-11/TU-10 magtape interface

DESCRIPTION

The files *mt0, ..., mt7* refer to the DEC TU10/TM11 magtape. When opened for reading or writing, the tape is rewound. When closed, it is rewound; if it was open for writing, an end-of-file is written first.

A standard tape consists of a series of 512 byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The *mt* files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named *rmt0, ..., rmt7*. Each *read* or *write* call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An error is returned when a tape mark is read, but another read will fetch the first record of the new tape file.

FILES

/dev/mt?, /dev/rmt?

SEE ALSO

tp (I)

BUGS

If any non-data error is encountered, it refuses to do anything more until closed. In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark.

NAME
       tty — general typewriter interface

DESCRIPTION
       All of the low-speed asynchronous communications ports use the same general interface, no matter what hardware is involved. This section discusses the common features of the interface; the KL, DC, and DH writeups (IV) describe peculiarities of the individual devices.

       When a typewriter file is opened, it causes the process to wait until a connection is established. In practice user's programs seldom open these files; they are opened by *init* and become a user's input and output file. The very first typewriter file open in a process becomes the *control type-writer* for that process. The control typewriter plays a special role in handling quit or interrupt signals, as discussed below. The control typewriter is inherited by a child process during a *fork*.

       A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters which have not yet been read by some program. Currently this limit is 256 characters. When the input limit is reached all the saved characters are thrown away without notice.

       When first opened, the interface mode is 300 baud; either parity accepted; 10 bits/character (one stop bit); and newline action character. The system delays transmission after sending certain function characters. Delays for horizontal tab, newline, and form feed are calculated for the Teletype Model 37; the delay for carriage return is calculated for the GE TermiNet 300. Most of these operating states can be changed by using the system call stty(II). In particular, provided the hardware permits, the speed of received and transmitted characters can be changed. In addition, the following software modes can be invoked: acceptance of even parity, odd parity, or both; a raw mode in which all characters may be read one at a time; a carriage return (CR) mode in which CR is mapped into newline on input and either CR or line feed (LF) cause echoing of the sequence LF-CR; mapping of upper case letters into lower case; suppression of echoing; suppression of delays after function characters; and the printing of tabs as spaces. See getty (VIII) for the way that terminal speed and type are detected.

       Normally, typewriter input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

       During input, erase and kill processing is normally done. The character '#' erases the last character typed, except that it will not erase beyond the beginning of a line or an EOT. The character '@' kills the entire line up to the point where it was typed, but not beyond an EOT. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Either '@' or '#' may be entered literally by preceding it by '\'; the erase or kill character remains, but the '\' disappears.

       In upper-case mode, all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by '\'. In addition, the following escape sequences are generated on output and accepted on input:

               for   use
                `   \'
                | \!
                ~   \^
                {   \(
                }   \)

       In raw mode, the program reading is awakened on each character. No erase or kill processing is done; and the EOT, quit and interrupt characters are not treated specially. The input parity bit is passed back to the reader, but parity is still generated for output characters.

- 2 -

The ASCII EOT character may be used to generate an end of file from a typewriter. When an EOT is received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line. Thus if there are no characters waiting, which is to say the EOT occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file signal. The EOT is passed back unchanged in raw mode.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a *hangup* signal is sent to all processes with the typewriter as control typewriter. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs which read a typewriter and test for end-of-file on their input can terminate appropriately when hung up on.

Two characters have a special meaning when typed. The ASCII DEL character (sometimes called 'rubout') is not passed to a program but generates an *interrupt* signal which is sent to all processes with the associated control typewriter. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to reveive a simulated trap to an agreed-upon location. See signal (II).

The ASCII character FS generates the *quit* signal. Its treatment is identical to the interrupt signal except that unless a receiving process has made other arrangements it will not only be terminated but a core image file will be generated. See signal (II). If you find it hard to type this character, try control-\ or control-shift-L.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is always generated on output. The EOT character is not transmitted (except in raw mode) to prevent terminals which respond to it from hanging up.

SEE ALSO

dc (IV), kl (IV), dh (IV), getty (VIII), stty (I, II), gtty (I, II), signal (II)

BUGS

Half-duplex terminals are not supported. On raw-mode output, parity should be transmitted as specified in the characters written.

NAME

vs — voice synthesizer interface

DESCRIPTION

Bytes written on *vs* drive a Federal Screw Works Votrax® voice synthesizer. The upper two bits encode an inflection, the other 6 specify a phoneme. The code is given in section vs (VII).

Touch-Tone® signals sent by a caller will be picked up during a *read* as the ASCII characters {0123456789#*}.

FILES

/dev/vs

SEE ALSO

speak (VI), vs (VII)

BUGS

NAME
    vt — 11/20 (vt01) interface

DESCRIPTION
    The file *vt0* provides the interface to a PDP 11/20 which runs a VT01A-controlled Tektronix 611 storage display. The inter-computer interface is a pair of DR-11C word interfaces.

    Although the display has essentially only two commands, namely "erase screen" and "display point", the 11/20 program will draw points, lines, and arcs, and print text on the screen. The 11/20 can also type information on the attached 33 TTY.

    This special file operates in two basic modes. If the first byte written of the file cannot be interpreted as one of the codes discussed below, the rest of the transmitted information is assumed to ASCII and written on the screen. The screen has 33 lines (1/2 a standard page). The file simulates a 37 TTY: the control characters NL, CR, BS, and TAB are interpreted correctly. It also interprets the usual escape sequences for forward and reverse half-line motion and for full-line reverse. Greek is not available yet. Normally, when the screen is full (i.e. the 34th line is started) the screen is erased before starting a new page. To allow perusal of the displayed text, it is usual to assert bit 0 of the console switches. This causes the program to pause before erasing until this bit is lowered.

    If the first byte written is recognizable, the display runs in graphic mode. In this case bytes written on the file are interpreted as display commands. Each command consists of a single byte usually followed by parameter bytes. Often the parameter bytes represent points in the plotting area. Each point coordinate consists of 2 bytes interpreted as a 2's complement 16-bit number. The plotting area itself measures (±03777)×(±03777) (numbers in octal); that is, 12 bits of precision. Attempts to plot points outside the screen limits are ignored.

    The graphic commands follow.

        order (1); 1 parameter byte
            The parameter indicates a subcommand, possibly followed by subparameter bytes, as follows:

        erase (1)
            The screen is erased. The program will wait until bit 0 of the console switches is down.

        label (3); several subparameter bytes
            The following bytes up to a null byte are printed as ASCII text on the screen. The origin of the text is the last previous point plotted; or the upper left hand of the screen if there were none.

        point (2); 4 parameter bytes
            The 4 parameter bytes are taken as a pair of coordinates representing a point to be plotted.

        line (3); 8 parameter bytes
            The parameter bytes are taken as 2 pairs of coordinates representing the ends of a line segment which is plotted. Only the portion lying within the screen is displayed.

        frame (4); 1 parameter byte
            The parameter byte is taken as a number of sixtieths of a second; an externally-available lead is asserted for that time. Typically the lead is connected to an automatic camera which advances its film and opens the shutter for the specified time.

        circle (5); 6 parameter bytes
            The parameter bytes are taken as a coordinate pair representing the origin, and a word representing the radius of a circle. That portion of the circle which lies within the screen is plotted.

        arc (6); 12 parameter bytes
            The first 4 parameter bytes are taken to be a coordinate-pair representing the center of a circle. The next 4 represent a coordinate-pair specifying a point on this circle.

The last 4 should represent another point on the circle. An arc is drawn counter-clockwise from the first circle point to the second. If the two points are the same, the whole circle is drawn. For the second point, only the smaller in magnitude of its two coordinates is significant; the other is used only to find the quadrant of the end of the arc. In any event only points within the screen limits are plotted.

dot-line (7); at least 6 parameter bytes

The first 4 parameter bytes are taken as a coordinate-pair representing the origin of a dot-line. The next byte is taken as a signed x-increment. The next byte is an unsigned word-count, with '0' meaning '256'. The indicated number of words is picked up. For each bit in each word a point is plotted which is visible if the bit is '1', invisible if not. High-order bits are plotted first. Each successive point (or non-point) is offset rightward by the given x-increment.

Asserting bit 3 of the console switches causes the display processor to throw away everything written on it. This sometimes helps if the display seems to be hung up.

FILES

/dev/vt0

BUGS

NAME

a.out — assembler and link editor output

DESCRIPTION

A.out is the output file of the assembler *as* and the link editor *ld.* Both programs make *a.out* executable if there were no errors and no unresolved external references.

This file has four sections: a header, the program and data text, a symbol table, and relocation bits (in that order). The last two may be empty if the program was loaded with the "—s" option of *ld* or if the symbols and relocation have been removed by *strip*.

The header always contains 8 words:

1 A magic number (407 or 410(8))
2 The size of the program text segment
3 The size of the initialized portion of the data segment
4 The size of the uninitialized (bss) portion of the data segment
5 The size of the symbol table
6 The entry location (always 0 at present)
7 Unused
8 A flag indicating relocation bits have been suppressed

The sizes of each segment are in bytes but are even. The size of the header is not included in any of the other sizes.

When a file produced by the assembler or loader is loaded into core for execution, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0 in the core image; the header is not loaded. If the magic number (word 0) is 407, it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. If the magic number is 410, the data segment begins at the first 0 mod 8K byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment.

The stack will occupy the highest possible locations in the core image: from 177776(8) and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by the *break* system call.

The start of the text segment in the file is 20(8); the start of the data segment is $20+S_t$ (the size of the text) the start of the relocation information is $20+S_t+S_d$; the start of the symbol table is $20+2(S_t+S_d)$ if the relocation information is present, $20+S_t+S_d$ if not.

The symbol table consists of 6-word entries. The first four words contain the ASCII name of the symbol, null-padded. The next word is a flag indicating the type of symbol. The following values are possible:

00 undefined symbol
01 absolute symbol
02 text segment symbol
03 data segment symbol
37 file name symbol (produced by ld)
04 bss segment symbol
40 undefined external (.globl) symbol
41 absolute external symbol
42 text segment external symbol
43 data segment external symbol
44 bss segment external symbol

Values other than those given above may occur if the user has defined some of his own instructions.

The last word of a symbol table entry contains the value of the symbol.

If the symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader *ld* as the name of a common region whose size is indicated by the value of the symbol.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in core when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation bits for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the "suppress relocation" flag in the header is on.

Bits 3-1 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

    00 indicates the reference is absolute
    02 indicates the reference is to the text segment
    04 indicates the reference is to initialized data
    06 indicates the reference is to bss (uninitialized data)
    10 indicates the reference is to an undefined external symbol.

Bit 0 of the relocation word indicates if *on* that the reference is relative to the pc (e.g. "clr x"); if *off*, that the reference is to the actual symbol (e.g., "clr *$x").

The remainder of the relocation word (bits 15-4) contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

SEE ALSO

as(I), ld(I), strip(I), nm(I)

NAME

archive (library) file format

DESCRIPTION

The archive command *ar* is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor *ld*.

A file produced by *ar* has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number is 177555(8) (it was chosen to be unlikely to occur anywhere else). The header of each file is 16 bytes long:

0-7	file name, null padded on the right
8-11	modification time of the file
12	user ID of file owner
13	file mode
14-15	file size

If the file is an odd number of bytes long, it is padded with a null byte, but the size in the header is correct.

Notice there is no provision for empty areas in an archive file.

SEE ALSO

ar, ld (I)

BUGS

Names are only 8 characters, not 14. More important, there isn't enough room to store the proper mode, so *ar* always extracts in mode 666.

NAME
core — format of core image file

DESCRIPTION
UNIX writes out a core image of a terminated process when any of various errors occur. See *signal (II)* for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called "core" and is written in the process's working directory (provided it can be; normal access controls apply).

The first 1024 bytes of the core image are a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The remainder represents the actual contents of the user's core area when the core image was written. At the moment, if the text segment is write-protected and shared, it is not dumped; otherwise the entire address space is dumped.

The actual format of the information in the first 1024 bytes is complicated. A guru will have to be consulted if enlightenment is required. In general the debugger *db (I)* should be used to deal with core images.

SEE ALSO
db (I), signal (II)

NAME

format of directories

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry. Directory entries are 16 bytes long. The first word is the i-number of the file represented by the entry, if non-zero; if zero, the entry is empty.

Bytes 2-15 represent the (14-character) file name, null padded on the right. These bytes are not cleared for empty slots.

By convention, the first two entries in each directory are for "." and "..". The first is an entry for the directory itself. The second is for the parent directory. The meaning of ".." is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. Therefore in both cases ".." has the same meaning as ".".

SEE ALSO

file system (V).

NAME

dump — incremental dump tape format

DESCRIPTION

The *dump* and *restor* commands are used to write and read incremental dump magnetic tapes.

The dump tape consists of blocks of 512-bytes each. The first block has the following structure.

```
struct {
        int     isize;
        int     fsize;
        int     date[2];
        int     ddate[2];
        int     tsize;
};
```

Isize, and *fsize* are the corresponding values from the super block of the dumped file system. (See file system (V).) *Date* is the date of the dump. *Ddate* is the incremental dump date. The incremental dump contains all files modified between *ddate* and *date*. *Tsize* is the number of blocks per reel. This block checksums to the octal value 31415.

Next there are enough whole tape blocks to contain one word per file of the dumped file system. This is *isize* divided by 16 rounded to the next higher integer. The first word corresponds to i-node 1, the second to i-node 2, and so forth. If a word is zero, then the corresponding file was not dumped. A non-zero value for the word indicates that the file was dumped and the value is one more than the number of blocks it contains.

The rest of the tape contains for each dumped file a header block and the data blocks from the file. The header contains an exact copy of the i-node (see file system (V)) and also checksums to 031415. The number of data blocks per file is directly specified by the control word for the file and indirectly specified by the size in the i-node. If these numbers differ, the file was dumped with a 'phase error'.

SEE ALSO

dump (VIII), restor (VIII), file system(V)

NAME

  fs — format of file system volume

DESCRIPTION

  Every file system storage volume (e.g. RF disk, RK disk, RP disk, DECtape reel) has a common format for certain vital information. Every such volume is divided into a certain number of 256 word (512 byte) blocks. Block 0 is unused and is available to contain a bootstrap program, pack label, or other information. ,

  Block 1 is the *super block*. Starting from its first word, the format of a super-block is

```
struct {
        int     isize;
        int     fsize;
        int     nfree;
        int     free[100];
        int     ninode;
        int     inode[100];
        char    flock;
        char    ilock;
        char    fmod;
        int     time[2];
};
```

  Isize is the number of blocks devoted to the i-list, which starts just after the super-block, in block 2. *Fsize* is the first block not potentially available for allocation to a file. This number is unused by the system, but is used by programs like *check (I)* to test for bad block numbers. The free list for each volume is maintained as follows. The *free* array contains, in *free[1], ... , free[nfree−1]*, up to 99 numbers of free blocks. *Free[0]* is the block number of the head of a chain of blocks constituting the free list. The first word in each free-chain block is the number (up to 100) of free-block numbers listed in the next 100 words of this chain member. The first of these 100 blocks is the link to the next member of the chain. To allocate a block: decrement *nfree,* and the new block is *free[nfree].* If the new block number is 0, there are no blocks left, so give an error. If *nfree* became 0, read in the block named by the new block number, replace *nfree* by its first word, and copy the block numbers in the next 100 words into the *free* array. To free a block, check if *nfree* is 100; if so, copy *nfree* and the *free* array into it, write it out, and set *nfree* to 0. In any event set *free[nfree]* to the freed block's number and increment *nfree.*

  Ninode is the number of free i-numbers in the *inode* array. To allocate an i-node: if *ninode* is greater than 0, decrement it and return *inode[ninode].* If it was 0, read the i-list and place the numbers of all free inodes (up to 100) into the *inode* array, then try again. To free an i-node, provided *ninode* is less than 100, place its number into *inode[ninode]* and increment *ninode.* If *ninode* is already 100, don't bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

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

  Time is the last time the super-block of the file system was changed, and is a double-precision representation of the number of seconds that have elapsed since 0000 Jan. 1 1970 (GMT). During a reboot, the *time* of the super-block for the root file system is used to set the system's idea of the time.

  I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 32 bytes long, so 16 of them fit into a block. Therefore, i-node *i* is located in block $(i + 31) / 16$, and begins $32 \cdot ((i + 31) \pmod{16})$ bytes from its start. I-node 1 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node is as follows.

```
struct {
        int     flags;          /* +0: see below */
        char    nlinks;         /* +2: number of links to file */
        char    uid;            /* +3: user ID of owner */
        char    gid;            /* +4: group ID of owner */
        char    size0;          /* +5: high byte of 24-bit size */
        int     size1;          /* +6: low word of 24-bit size */
        int     addr[8];        /* +8: block numbers or device number */
        int     actime[2];      /* +24: time of last access */
        int     modtime[2];     /* +28: time of last modification */
};
```

The flags are as follows:

```
100000    i-node is allocated
060000    2-bit file type:
     000000    plain file
     040000    directory
     020000    character-type special file
     060000    block-type special file.
010000    large file
004000    set user-ID on execution
002000    set group-ID on execution
000400    read (owner)
000200    write (owner)
000100    execute (owner)
000070    read, write, execute (group)
000007    read, write, execute (others)
```

Special files are recognized by their flags and not by i-number. A block-type special file is basically one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files the high byte of the first address word specifies the type of device; the low byte specifies one of several devices of that type. The device type numbers of block and character special files overlap.

The address words of ordinary files and directories contain the numbers of the blocks in the file (if it is small) or the numbers of indirect blocks (if the file is large).

Byte number n of a file is accessed as follows. N is divided by 512 to find its logical block number (say b) in the file. If the file is small (flag 010000 is 0), then b must be less than 8, and the physical block number is $addr[b]$.

If the file is large, b is divided by 256 to yield i, and $addr[i]$ is the physical block number of the indirect block. The remainder from the division yields the word in the indirect block which contains the number of the block for the sought-for byte.

For block b in a file to exist, it is not necessary that all blocks less than b exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

SEE ALSO
        check (VIII)

NAME

     mtab — mounted file system table

DESCRIPTION

     Mtab resides in directory */etc* and contains a table of devices mounted by the *mount* command. *Umount* removes entries.

     Each entry is 64 bytes long; the first 32 are the null-padded name of the place where the special file is mounted; the second 32 are the null-padded name of the special file. The special file has all its directories stripped away; that is, everything through the last "/" is thrown away.

     This table is present only so people can look at it. It does not matter to *mount* if there are duplicated entries nor to *umount* if a name cannot be found.

FILES

     /etc/mtab

SEE ALSO

     mount (VIII), umount (VIII)

BUGS

NAME

passwd — password file

DESCRIPTION

Passwd contains for each user the following information:

name (login name, contains no upper case)
encrypted password
numerical user ID
numerical group ID (for now, always 1)
GCOS job number and box number
initial working directory
program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The job and box numbers are separated by a comma. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

SEE ALSO

login (I), crypt (III), passwd (I)

NAME
       speak.m — voice synthesizer vocabulary

SYNOPSIS
       struct {
               int n;
               struct {
                       int key;   .
                       int phon;
               } entry[vocab.n];
               int m;
               char strings[vocab.m];
       } vocab

DESCRIPTION
       This type of file is created and used only by *speak*. The keys are the words, word fragments and
       letters of the vocabulary.

       The *i*th key is stored as a null-terminated string at &*vocab.strings[entry[i].key]*. Its phonetic string
       is similarly stored at &*vocab.strings[entry[i].phon]*. The keys are ordered lexicographically. The
       contents of certain parts of *vocab* are invariant: *vocab.entry[0].key* = *vocab.entry.[0].phon* =
       vocab.strings[0] = 0.

       Each key is maintained exactly as entered into *speak*, so that fragments are recognized by an ini-
       tial '%' and letters by an initial '*'.

       Each phoneme of a phonetic string is maintained in *vs* code. A '%' in a phonetic string is
       represented as octal 001, and all following characters are kept literally.

SEE ALSO
       speak (VI), vs (VII)

BUGS
       The coding 001 for '%' precludes the use of phoneme *3-1*, which is no particular loss on the
       model 5 Votrax, but will pinch on the model 6. 0200 would be a safe choice.

NAME

tp — DEC/mag tape formats

DESCRIPTION

The command *tp* dumps and extracts files to and DECtape and magtape. The formats of these tapes are the same except that magtapes have larger directories.

Block zero contains a copy of a stand-alone bootstrap program. See boot procedures (VIII).

Blocks 1 through 24 for DECtape (1 through 62 for magtape) contain a directory of the tape. There are 192 (resp. 496) entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

path name	32 bytes
mode	2 bytes
uid	1 byte
gid	1 byte
unused	1 byte
size	3 bytes
time modified	4 bytes
tape address	2 bytes
unused	16 bytes
check sum	2 bytes

The path name entry is the path name of the file when put on the tape. If the pathname starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null byte. Mode, uid, gid, size and time modified are the same as described under i-nodes (file system (V)). The tape address is the tape block number of the start of the contents of the file. Every file starts on a block boundary. The file occupies (size+511)/512 blocks of continuous tape. The checksum entry has a value such that the sum of the 32 words of the directory entry is zero.

Blocks 25 (resp. 63) on are available for file storage.

A fake entry (see tp(I)) has a size of zero.

SEE ALSO

file system(V), tp(I)

NAME

ttys — typewriter initialization data

DESCRIPTION

The *ttys* file is read by the *init* program and specifies which typewriter special files are to have a process created for them which will allow people to log in. It consists of lines of 3 characters each.

The first character is either '0' or '1'; the former causes the line to be ignored, the latter causes it to be effective. The second character is the last character in the name of a typewriter; e.g. *x* refers to the file '/dev/tty*x*'. The third character is the offset in a table contained in *init* which selects an initialization program for the line; currently it must be '0' and the only such program is */etc/getty*.

FILES

found in /etc

SEE ALSO

init (VIII)

NAME

      utmp — user information

DESCRIPTION

      This file allows one to discover information about who is currently using UNIX. The file is binary; each entry is 16(10) bytes long. The first eight bytes contain a user's login name or are null if the table slot is unused. The low order byte of the next word contains the last character of a typewriter name. The next two words contain the user's login time. The last word is unused.

      This file resides in directory /tmp.

SEE ALSO

      init (VIII) and login (I), which maintain the file; who (I), which interprets it.

NAME

wtmp — user login history

DESCRIPTION

This file records all logins and logouts. Its format is exactly like utmp (V) except that a null user name indicates a logout on the associated typewriter. Furthermore, the typewriter name '~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with typewriter names '|' and '}' indicate the system-maintained time just before and just after a *date* command has changed the system's idea of the time.

Wtmp is maintained by login (I) and init (VII). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by ac (VIII).

This file resides in directory */usr/adm*.

SEE ALSO

login (I), init (VIII), ac (VIII), who (I)
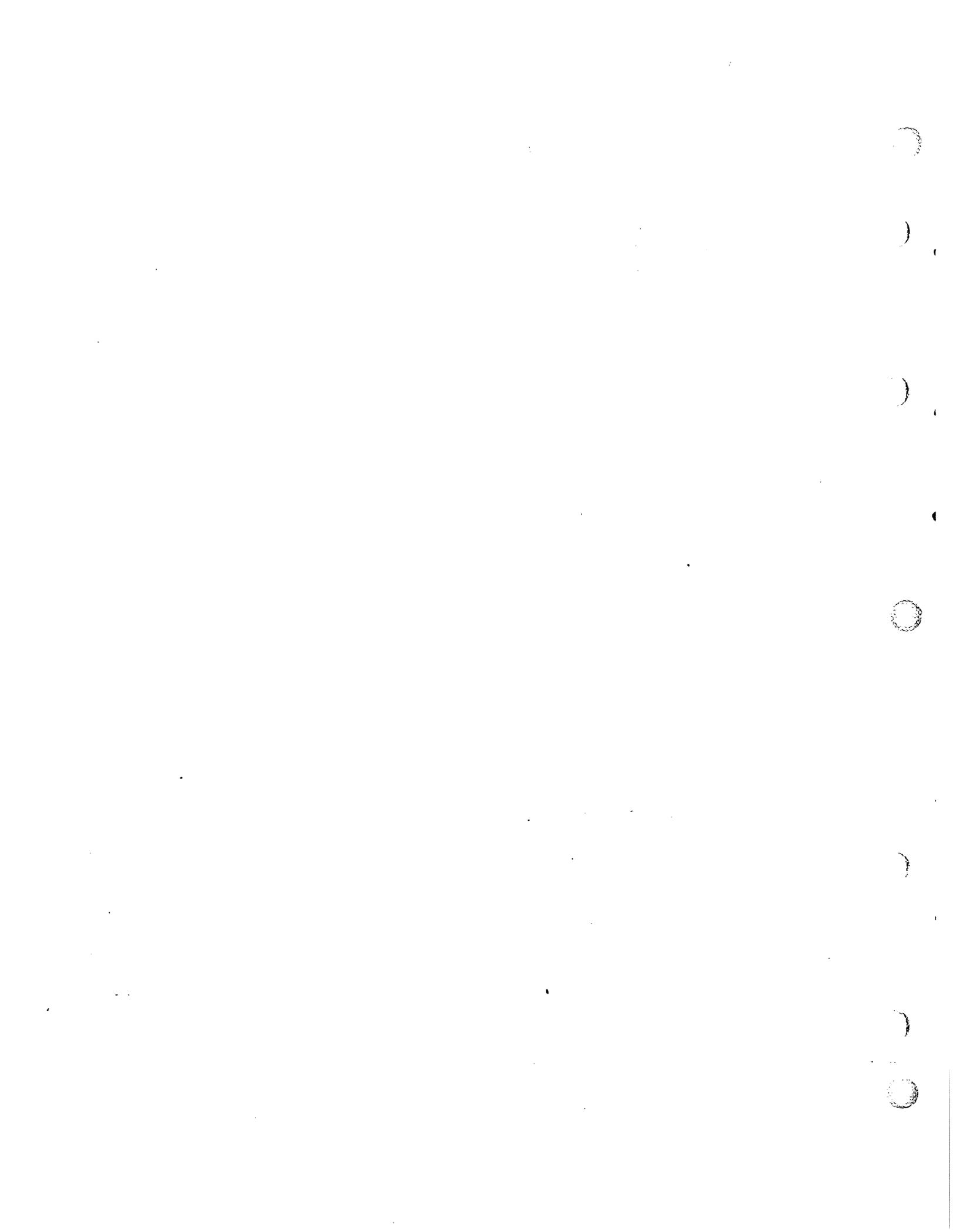
NAME

    apl − APL interpreter

SYNOPSIS

    apl

DESCRIPTION

Apl is an interpreter for the language APL described in the reference. The interpreter maintains its workspace on disk rather than in core. This has two consequences: there is the potential of a million byte workspace; it takes a week to access that much data.

Not Implemented (never)
1. Lamination (except for scalar,scalar)
2. 0 div 0 is a domain error.
3. 0 mod x is a domain error.
4. No function definition − use ')edit fname' to enter the system editor; type "w" when done editing to write the function out in a place where apl can pick it up. Type "w file" to save it.
5. Indexing is off in character vectors containing overstrikes.

Under Implemented (later)
1. Negative numbers raised to fractional powers are handled incorrectly.
2. No trace or SI.
3. Incomplete set of I-beams and system calls.

Over Implemented (over zealous)
1. Ravel[i] − obvious extension of cat.
2. Grade up and grade down extend to matrices.
3. Arbitrary overstriking is allowed in characters.

FILES

    /usr/lib/apl/*   programs
    alloc.d workspace
    apl_ed   editor intermediate

SEE ALSO

    IBM GH20-0906-1 "APL User's Manual"
    /usr/pub/apl ASCII APL character set

BUGS

NAME
    azel − obtain satellite predictions

SYNOPSIS
    azel [−d] [−l] satellite1 [−d] [−l] satellite2 ...

DESCRIPTION
    Azel predicts, in convenient form, the apparent trajectories of Earth satellites whose orbital elements are given in the argument files. If a given satellite name cannot be read, an attempt is made to find it in a directory of satellites maintained by the programs's author. The −d option causes *azel* to ask for a date and read line 1 data (see below) from the standard input. The −l option causes *azel* to ask for the observers latitude, west-longitude, and height above sea level.

    For each satellite given the program types its full name, the date, and a sequence of lines each containing a time, an azimuth, an elevation, a distance, and a visual magnitude. Each such line indicates that: at the indicated time, the satellite may be seen from Murray Hill (or provided location) at the indicated azimuth and elevation, and that its distance and apparent magnitude are as given. Predictions are printed only when the sky is dark (sun more than 5 degrees below the horizon) and when the satellite is not eclipsed by the earth's shadow. Satellites which have not been seen and verified will not have had their visual magnitude level set correctly.

    All times input and output by *azel* are GMT (Universal Time).

    The satellites for which elements are maintained are:

    sla,b,e,f,k   Skylab A through Skylab K. Skylabs A and B are the laboratory and its rocket respectively; the remainder are various other objects attendant upon its launch and subsequent activities. A, B, and probably K have been sighted and verified.

    cop   Copernicus I. Never verified.

    oao   Orbiting Astronomical Observatory. Seen and verified.

    pag   Pageos I. Seen and verified; fairly dim (typically 2nd-3rd magnitude), but elements are extremely accurate.

    exp19   Explorer 19; seen and verified, but quite dim (4th-5th magnitude) and fast-moving.

    c103b, c156b, c184b, c206b, c220b, c461b, c500b
                  Various of the USSR Cosmos series; none seen.

    7276a   Unnamed (satellite # 72-76A); not seen.

    The element files used by *azel* contain five lines. The first line gives a year, month number, day, hour, and minute at which the program begins its consideration of the satellite, followed by a number of minutes and an interval in minutes. If the year, month, and day are 0, they are taken to be the current date (taken to change at 6 A.M. local time). The output report starts at the indicated epoch and prints the position of the satellite for the indicated number of minutes at times separated by the indicated interval. This line is ended by two numbers which specify options to the program governing the completeness of the report; they are ordinarily both "1". The first option flag suppresses output when the sky is not dark; the second supresses output when the satellite is eclipsed by the earth's shadow. The next line of an element file is the full name of the satellite. The next three are the elements themselves (including certain derivatives of the elements).

FILES
    /usr/jfo/el/* − orbital element files

SEE ALSO
    sky (VI)

AUTHOR
    J. F. Ossanna

NAME
      bas — basic

SYNOPSIS
      bas [file]

DESCRIPTION
      Bas is a dialect of Basic. If a file argument is provided, the file is used for input before the console is read. *Bas* accepts lines of the form:

          statement
          integer statement

      Integer numbered statements (known as internal statements) are stored for later execution. They are stored in sorted ascending order. Non-numbered statements are immediately executed. The result of an immediate expression statement (that does not have '=' as its highest operator) is printed.

      Statements have the following syntax:

      expression
          The expression is executed for its side effects (assignment or function call) or for printing as described above.

      done
          Return to system level.

      draw expression expression expression
          A line is drawn on the Tektronix 611 display '/dev/vt0' from the current display position to the XY co-ordinates specified by the first two expressions. The scale is zero to one in both X and Y directions. If the third expression is zero, the line is invisible. The current display position is set to the end point.

      display list
          The list of expressions and strings is concatenated and displayed (i.e. printed) on the 611 starting at the current display position. The current display position is not changed.

      dump
          The name and current value of every variable is printed.

      erase
          The 611 screen is erased.

      for name = expression expression statement
      for name = expression expression
          ...
      next
          The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression.

      goto expression
          The expression is evaluated, truncated to an integer and execution goes to the corresponding integer numbered statment. If executed from immediate mode, the internal statements are compiled first.

      if expression statement
          The statement is executed if the expression evaluates to non-zero.

      list [expression [expression]]
          is used to print out the stored internal statements. If no arguments are given, all internal statements are printed. If one argument is given, only that internal statement is listed. If two arguments are given, all internal statements inclusively between the arguments are printed.

print list
>   The list of expressions and strings are concatenated and printed. (A string is delimited by "
>   characters.)

prompt list
>   *Prompt* is the same as *print* except that no newline character is printed.

return [expression]
>   The expression is evaluated and the result is passed back as the value of a function call. If
>   no expression is given, zero is returned.

run
>   The internal statements are compiled. The symbol table is re-initialized. The random
>   number generator is reset. Control is passed to the lowest numbered internal statement.

save [expression [expression]]
>   *Save* is like *list* except that the output is written on the *file* argument. If no argument is
>   given on the command, **b.out** is used.

Expressions have the following syntax:

name
>   A name is used to specify a variable. Names are composed of a letter followed by letters
>   and digits. The first four characters of a name are significant.

number
>   A number is used to represent a constant value. A number is written in Fortran style, and
>   contains digits, an optional decimal point, and possibly a scale factor consisting of an e fol-
>   lowed by a possibly signed exponent.

(expression)
>   Parentheses are used to alter normal order of evaluation.

_ expression
>   The result is the negation of the expression.

expression operator expression
>   Common functions of two arguments are abbreviated by the two arguments separated by
>   an operator denoting the function. A complete list of operators is given below.

expression ([expression [, expression] ...])
>   Functions of an arbitrary number of arguments can be called by an expression followed by
>   the arguments in parentheses separated by commas. The expression evaluates to the line
>   number of the entry of the function in the internally stored statements. This causes the
>   internal statements to be compiled. If the expression evaluates negative, a builtin function
>   is called. The list of builtin functions appears below.

name [expression [, expression] ...]
>   Each expression is truncated to an integer and used as a specifier for the name. The result
>   is syntactically identical to a name. a[1,2] is the same as a[1][2]. The truncated expressions
>   are restricted to values between 0 and 32767.

. The following is the list of operators:

=
>   = is the assignment operator. The left operand must be a name or an array element. The
>   result is the right operand. Assignment binds right to left, all other operators bind left to
>   right.

& |
>   & (logical and) has result zero if either of its arguments are zero. It has result one if both
>   its arguments are non-zero. | (logical or) has result zero if both of its arguments are zero. It
>   has result one if either of its arguments are non-zero.
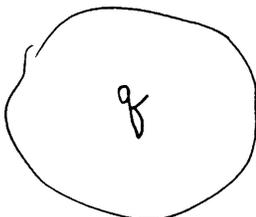
< <= > >= == <>
> The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, <> not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: a>b>c is the same as a>b&b>c.

+ −
> Add and subtract.

* /
> Multiply and divide.

^
> Exponentiation.

The following is a list of builtin functions:

arg(i)
> is the value of the *i*-th actual parameter on the current level of function call.

exp(x)
> is the exponential function of *x*.

log(x)
> is the natural logarithm of *x*.

sin(x)
> is the sine of *x* (radians).

cos(x)
> is the cosine of *x* (radians).

atn(x)
> is the arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.

rnd( )
> is a uniformly distributed random number between zero and one.

expr( )
> is the only form of program input. A line is read from the input and evaluated as an expression. The resultant value is returned.

int(x)
> returns *x* truncated to an integer.

FILES
> /tmp/btm?   temporary
> b.out   save file

DIAGNOSTICS
> Syntax errors cause the incorrect line to be typed with an underscore where the parse failed. All other diagnostics are self explanatory.

BUGS
> Has been known to give core images.

NAME

    bj — the game of black jack

SYNOPSIS

    /usr/games/bj

DESCRIPTION

    Bj is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

    The bet is $2 every hand.

        A player 'natural' (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a 'push' (no money exchange).

        If the dealer has an ace up, the player is allowed to make an 'insurance' bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.

        If the player is dealt two cards of the same value, he is allowed to 'double'. He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)

        If a dealt hand has a total of ten or eleven, the player may 'double down'. He may double the bet ($2 to $4) and receive exactly one more card on that hand.

        Under normal play, the player may 'hit' (draw a card) as long as his total is not over twenty-one. If the player 'busts' (goes over twenty-one), the dealer wins the bet.

        When the player 'stands' (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.

        If both player and dealer stand, the one with the largest total wins. A tie is a push.

    The machine deals and keeps score. The following questions will be asked at appropriate times. Each question is answered by y followed by a new line for 'yes', or just new line for 'no'.

    ?   (means, "do you want a hit?")
    Insurance?
    Double down?

    Every time the deck is shuffled, the dealer so states and the 'action' (total bet) and 'standing' (total won or lost) is printed. To exit, hit the interrupt key (DEL) and the action and standing will be printed.

BUGS

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. The *month* is a number between 1 and 12.
>   The calendar produced is that for England and her colonies.
>
>   Try September 1752.

BUGS
>   The year is always considered to start in January even though this is historically naive.

NAME
     catsim − phototypesetter simulator

SYNOPSIS
     catsim

DESCRIPTION
     Catsim will interpret its standard input as codes for the phototypesetter (cat). The output of *cat-sim* is output to the display (vt).

     About the only use of *catsim* is to save time and paper on the phototypesetter by the following command:

          troff −t file ... | catsim

FILES
     /dev/vt0

SEE ALSO
     troff (I), cat (IV), vt (IV)

BUGS
     Point sizes are not correct. The vt character set is restricted to one font of ASCII.

NAME
  chess — the game of chess

SYNOPSIS
  /usr/games/chess

DESCRIPTION
  Chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check and is not required; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

  Each move is echoed in the appropriate notation followed by the program's reply.

FILES
  /usr/lib/book   opening 'book'

DIAGNOSTICS
  The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

WARNING
  Over-use of this program will cause it to go away.

BUGS
  Pawns may be promoted only to queens.

NAME

col — filter reverse line feeds

SYNOPSIS

col

DESCRIPTION

Col reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ascii code ESC-7). *Col* is particularly useful for filtering multicolumn output made with the '.rt' command of *nroff.*

SEE ALSO

nroff (I)

BUGS

Can't back up more than 102 lines.

NAME

cubic — three dimensional tic-tac-toe

SYNOPSIS

/usr/games/cubic

DESCRIPTION

Cubic plays the game of three dimensional 4×4×4 tic-tac-toe. Moves are given by the three digits (each 1-4) specifying the coordinate of the square to be played.

WARNING

Too much playing of the game will cause it to disappear.

BUGS

NAME
> factor — discover prime factors of a number

SYNOPSIS
> **factor**

DESCRIPTION
> When *factor* is invoked, it types out 'Enter:' at you. If you type in a positive number less than 2^{56} (about 7.2×10^{16}) it will repeat the number back at you and then its prime factors each one printed the proper number of times. Then it says 'Enter:' again. To exit, feed it an EOT or a delete.
>
> Maximum time to factor is proportional to $\sqrt{n}$ and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 10^{13}.

DIAGNOSTICS
> 'Ouch.' for input out of range or for garbage input.

BUGS

NAME
> graf — draw graph on GSI terminal

SYNOPSIS
> **graf | nroff | gsi**

DESCRIPTION
> *Graf* is a preprocessor to *nroff* (q.v.) for producing plots imbedded in documents. The standard input is copied to the standard output except for plots, which are inserted whenever a line beginning ".GR" is found. The remainder of the line should be the arguments to *plog*, normally including a "<file" to point off to the data for the graph. *Graf* itself reads its standard input.
>
> *Graf* and *neqn* can be used together:
>
> neqn files | graf | nroff | gsi
>
> produces a memo with figures and equations and text all intermixed. There is no typesetter equivalent of *graf*, nor are there any plans for one.

SEE ALSO
> plog (VI), neqn (I), nroff (I), gsi (VI)

BUGS
> Same as *plog* (VI). Axes and labels are not scaled down correctly for smaller graphs. It should recognize .so commands.

NAME

gsi — interpret funny characters on GSI terminal

SYNOPSIS

gsi

DESCRIPTION

Gsi interprets commands specific to the GSI terminal. It converts half line forward and reverse motions into the right vertical motion, It also attempts to draw Greek letters and other special symbols of the Model 37 extended character set. (These are normally preceded by shift-out and followed by shift-in.) *Gsi* is most often used to print equations neatly, in the sequence

neqn file ... | nroff | gsi

Gsi also interprets the plot control characters ACK and BEL. This makes it useful in the sequence

graf | nroff | gsi

FILES

SEE ALSO

neqn (I), graf (VI), greek (VII)

BUGS

Some funny characters can't be correctly printed in column 1 because you can't move to the left from there.

NAME
      hyphen — find hyphenated words

SYNOPSIS
      hyphen file ...

DESCRIPTION
      It finds all of the words in a document which are hyphenated across lines and prints them back at you in a convenient format.

      If no arguments are given, the standard input is used. Thus *hyphen* may be used as a filter.

BUGS
      Yes, it gets confused, but with no ill effects other than spurious extra output.

NAME

        ibm — submit off-line job to HO IBM 370

SYNOPSIS

        ibm [**—j**] file ...

DESCRIPTION

        Ibm arranges to have the 201 data phone daemon submit a job to the IBM 370 at Holmdel via
        the Murray Hill H6070. Normally the job is submitted with enough "JCL" (the IBM version of
        the shell) to return the output to your box at Murray Hill. You can supply your own if you
        dare— the **—j** option suppresses all JCL.

        If there are no arguments, the standard input is read and submitted. Thus *ibm* may be used as a
        filter.

FILES

        /usr/dpd/*   spool area
        /etc/passwd   personal ident cards
        /etc/dpd   daemon

SEE ALSO

        dpd (VIII), passwd (V)

BUGS

        Stuff is sent on 6-bit cards, so lower case vanishes, as do some of the special characters.

NAME
       lc − LIL compiler

SYNOPSIS
       lc [−c] [−p] [−P] file ...

DESCRIPTION
       Lc is the UNIX LIL compiler. It accepts three types of arguments:

       Arguments whose names end with '.l' are taken to be LIL 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 '.l'. The '.o' file is normally deleted, however, if a single LIL program is compiled and loaded all at one go.

       The following flags are interpreted by *lc*.

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

       −p   If loading takes place, replace the standard startup routine by one which automatically calls the *monitor* subroutine (III) 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* (I).

       −P   Run only the string preprocessor on the named LIL programs, and leave the output on corresponding files suffixed '.i'.

       Other arguments are taken to be object programs or perhaps libraries of 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**.

FILES
       file.l   input file
       file.o   object file
       a.out   loaded output
       /tmp/ctm?   temporary
       /lib/l[01]   compiler
       /lib/crt0.o   runtime startoff
       /lib/mcrt0.o   runtime startoff for monitoring.
       /lib/libc.a   builtin functions, etc.

SEE ALSO
       "Programming in LIL: a tutorial," LIL Reference Manual,
       monitor (III), prof (I), cdb (I), ld (I).

DIAGNOSTICS
       The diagnostics produced by LIL itself are intended to be self-explanatory.

BUGS
       Creates temporary symbols of the form '..octal', which might conflict with user defined symbols.

NAME

m6 — general purpose macroprocessor

SYNOPSIS

m6 [name]

DESCRIPTION

M6 copies the standard input to the standard output, with substitutions for any macro calls that appear. When a file name argument is given, that file is read before the standard input.

The processor is as described in the reference with these exceptions:

#def,arg1,arg2,arg3: causes *arg1* to become a macro with defining text *arg2* and (optional) built-in serial number *arg3*.

#del,arg1: deletes the definition of macro *arg1*.

#end: is not implemented.

#list,arg1: sends the name of the macro designated by *arg1* to the current destination without recognition of any warning characters; *arg1* is 1 for the most recently defined macro, 2 for the next most recent, and so on. The name is taken to be empty when *arg1* doesn't make sense.

#warn,arg1,arg2: replaces the old warning character *arg1* by the new warning character *arg2*.

#quote,arg1: sends the definition text of macro *arg1* to the current destination without recognition of any warning characters.

#serial,arg1: delivers the built-in serial number associated with macro *arg1*.

#source,arg1: is not implemented.

#trace,arg1: with *arg1* = '1' causes a reconstruction of each later call to be placed on the standard output with a call level number; other values of *arg1* turn tracing off.

The built-in 'warn' may be used to replace inconvenient warning characters. The example below replaces '#' ':' '<' '>' by '[' ']' '{' '}'.

```
#warn,<#>,[:
[warn,<:>,]:
[warn,[substr,<<>>,1,1;,{]
[warn,[substr,{{>>,2,1;,}]
[now,[calls look like this]]
```

Every built-in function has a serial number, which specifies the action to be performed before the defining text is expanded. The serial numbers are: 1 gt, 2 eq, 3 ge, 4 lt, 5 ne, 6 le, 7 seq, 8 sne, 9 add, 10 sub, 11 mpy, 12 div, 13 exp, 20 if, 21 def, 22 copy, 23 warn, 24 size, 25 substr, 26 go, 27 gobk, 28 del, 29 dnl, 32 quote, 33 serial, 34 list, 35 trace. Serial number 0 specifies no built-in action.

SEE ALSO

A. D. Hall, M6 Reference Manual. Computer Science Technical Report #2, Bell Laboratories, 1969.

DIAGNOSTICS

Various table overflows and "impossible" conditions result in comment and dump. There are no diagnostics for poorly formed input.

AUTHOR

M. D. McIlroy

BUGS

Provision should be made to extend tables as needed, instead of wasting a big fixed core allocation. You get what the PDP11 gives you for arithmetic.

- 1 -

NAME
        maze — generate a maze problem

SYNOPSIS
        maze

DESCRIPTION
        Maze asks a few questions and then prints a maze.

BUGS
        Some mazes (especially small ones) have no solutions.

NAME

      moo — guessing game

SYNOPSIS

      /usr/games/moo

DESCRIPTION

      Moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A 'cow' is a correct digit in an incorrect position. A 'bull' is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).

BUGS

NAME
    npr — print file on Spider line-printer

SYNOPSIS
    npr file ...

DESCRIPTION
    Npr prints files on the line-printer in the Spider room, sending them over the Spider loop.

    If there are no arguments, the standard input is read and submitted. Thus *npr* may be used as a filter.

FILES
    /dev/tiu/d2   tiu to loop

BUGS

NAME
          plog — make a graph on the gsi terminal

SYNOPSIS
          plog [option] ...

DESCRIPTION
          Plog is almost the same as *plot* (q.v.) but the plot is written on the standard output using the control sequences for the GSI terminal. The following changes have been made:

          - The default for grids is no grid at all.
          - The 'a' option can be followed by two arguments; the second is the starting point for automatic abscissas.
          - There is a new option 'h' which must be followed by a numerical argument: it specifies the height desired for the plot.
          - There is a new option 'w' similar to 'h', except that the width is specified. If only one of 'h' and 'w' is given, the plot is made square of the indicated size. If neither is given, the plot is made six inches square.
          - There is a new option 'r' to be followed by a number which locates the plot that many inches to the right on the page.

SEE ALSO
          plot (VI)

BUGS
          Same as plot (VI). Drawing lines is not yet done exactly right. If you store the output in a file, before printing with cat you must turn off delays and turn off CR-NL echo (e.g. "stty -delay nl");

NAME
        plot — make a graph

SYNOPSIS
        plot [option] ...

DESCRIPTION
        Plot takes pairs of numbers from the standard input as abscissas and ordinates of a graph. The
        graph is plotted on the storage scope, /dev/vt0.

        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.

        c   Place character string given by next argument at each point.

        d   Omit connections between points. (Disconnect.)

        g*n*   Grid style:
                n=0, no grid
                n=1, axes only
                n=2, complete grid (default).

        s   Save screen, don't erase before plotting.

        x   Next 1 (or 2) arguments are lower (and upper) *x* limits.

        y   Next 1 (or 2) arguments are lower (and upper) *y* limits.

        Points are connected by straight line segments in the order they appear in input. If a specified
        lower limit exceeds the upper limit, or if the automatic increment is negative, the graph is plot-
        ted upside down. Automatic abscissas begin with the lower *x* limit, or with 0 if no limit is
        specified. Grid lines and automatically determined limits fall on round values, however round-
        ness may be subverted by giving an inappropriately rounded lower limit. Plotting symbols
        specified by c are placed so that a small initial letter, such as + o x, will fall approximately on
        the plotting point.

FILES
        /dev/vt0

SEE ALSO
        spline (VI), plog (VI)

BUGS
        A limit of 1000 points is enforced silently.

NAME

  ptx — permuted index

SYNOPSIS

  ptx [**−t**] input [output]

DESCRIPTION

  Ptx generates a permuted index from file *input* on file *output*. 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.

  Input should be edited to remove useless lines. The following words are suppressed: 'a', 'an', 'and', 'as', 'is', 'for', 'of', 'on', 'or', 'the', 'to', 'up'.

  The optional argument **−t** causes *ptx* to prepare its output for the phototypesetter.

  The index for this manual was generated using *ptx*.

FILES

  /bin/sort

NAME
     sfs — structured file scanner

SYNOPSIS
     sfs filename [—]

DESCRIPTION
     Sfs provides an interactive program for scanning and patching a structured file. If the second argument is supplied, the file is block addressed.

     Some features of *sfs* include.
     1. It provides interactive and preprogrammed operation.
     2. It provides expression evaluation (32 bit precision) and branching.
     3. It provides the ability to define a large set of hierarchical structure definitions.
     4. It provides the ability to locate, to dump, and to patch specific instances of structure in the file. Furthermore, in the dump and patch operations the external form of the structure is selected by the user.
     5. It provides the ability to escape to the UNIX command level to allow the use of other UNIX debugging aids.

SEE ALSO
     "SFS reference manual" (internal memorandum)

BUGS

NAME
      sky − obtain ephemerides

SYNOPSIS
      sky

DESCRIPTION
      Sky predicts the apparent locations of the Sun, the Moon, the planets out to Saturn, stars of mag-
      nitude at least 2.5, and certain other celestial objects. *Sky* reads the standard input to obtain a
      GMT time typed on one line with blanks separating year, month number, day, hour, and minute;
      if the year is missing the current year is used. If a blank line is typed the current time is used.
      The program prints the azimuth, elevation, and magnitude of objects which are above the hor-
      izon at the ephemeris location of Murray Hill at the indicated time.

      Placing a "1" input after the minute entry causes the program to print out the Greenwich Side-
      real Time at the indicated moment and to print for each body its right ascension and declination
      as well as its azimuth and elevation. Also, instead of the magnitude, the geocentric distance of
      the body, in units the program considers convenient, is printed. (For planets the unit is essen-
      tially A. U.)

      The magnitudes of Solar System bodies are not calculated and are given as 0. The effects of at-
      mospheric extinction are not included; the mean magnitudes of variable stars are marked with
      "*".

      For all bodies, the program takes into account precession and nutation of the equinox, annual
      (but not diurnal) aberration, diurnal parallax, and the proper motion of stars (but not annual
      parallax). In no case is refraction included.

      The program takes into account perturbations of the Earth due to the Moon, Venus, Mars, and
      Jupiter. The expected accuracies are: for the Sun and other stellar bodies a few tenths of seconds
      of arc; for the Moon (on which particular care is lavished) likewise a few tenths of seconds. For
      the Sun, Moon and stars the accuracy is sufficient to predict the circumstances of eclipses and oc-
      cultations to within a few seconds of time. The planets may be off by several minutes of arc.

FILES
      /usr/lib/startab, /usr/lib/moontab

SEE ALSO
      azel (VI)
      American Ephemeris and Nautical Almanac, for the appropriate years; also, the *Explanatory Supple-
      ment to the American Ephemeris and Nautical Almanac.*

AUTHOR
      R. Morris

BUGS

NAME
        sno — Snobol interpreter

SYNOPSIS
        sno [file]

DESCRIPTION
        Sno is a Snobol III (with slight differences) compiler and interpreter. *Sno* obtains input from the' concatenation of *file* and 'the standard input. All input through a statement containing the label 'end' is considered program and is compiled. The rest is available to 'syspit'.

        Sno differs from Snobol III in the following ways.

        There are no unanchored searches. To get the same effect:

        a ** b   unanchored search for b
        a *x* b = x c   unanchored assignment

        There is no back referencing.

        x = "abc"
        a *x* x   . is an unanchored search for 'abc'

        Function declaration is different. The function declaration is done at compile time by the use of the label 'define'. Thus there is no ability to define functions at run time and the use of the name 'define' is preempted. There is also no provision for automatic variables other than the parameters. For example:

        define   f( )

        or

        define   f(a,b,c)

        All labels except 'define' (even 'end') must have a non-empty statement.

        If 'start' is a label in the program, program execution will start there. If not, execution begins with the first executable statement. 'define' is not an executable statement.

        There are no builtin functions.

        Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators '/' and '*' must be set off by space.

        The right side of assignments must be non-empty.

        Either ' or " may be used for literal quotes.

        The pseudo-variable 'sysppt' is not available.

SEE ALSO
        Snobol III manual. (JACM; Vol. 11 No. 1; Jan 1964; pp 21)

BUGS

NAME
       speak — word to voice translator

SYNOPSIS
       speak [**—epsv**] [vocabulary [output]]

DESCRIPTION
       Speak turns a stream of words into utterances and outputs them to a voice synthesizer, or to a specified output file. It has facilities for maintaining a vocabulary. It receives, from the standard input

- working lines: text of words separated by blanks
- phonetic lines: strings of phonemes for one word preceded and separated by commas. The phonemes may be followed by comma-percent then a 'replacement part' — an ASCII string with no spaces. The phonetic code is given in bs (VII).
- empty lines
- command lines: beginning with !. The following command lines are recognized:

!r file	replace coded vocabulary from file
!w file	write coded vocabulary on file
!p	print phonetics for working word
!l	list vocabulary on standard output with phonetics
!c word	copy phonetics from working word to specified word
!d	print decomposition into substrings

Each working line replaces its predecessor. Its first word is the 'working word'. Each phonetic line replaces the phonetics stored for the working word. In particular, a phonetic line of comma only deletes the entry for the working word. Each working line, phonetic line or empty line causes the working line to be uttered. The process terminates at the end of input.

Unknown words are pronounced by rules, and failing that, are spelled. Spelling is done by taking each character of the word, prefixing it with "*", and looking it up. Unspellable words burp.

Speak is initialized with a coded vocabulary stored in file /usr/lib/speak.m. The vocabulary option substitutes a different file for /usr/lib/speak.m.

A set of single letter options may appear in any order preceded by —. Their meanings are:

- **—e** suppress English preprocessing
- **—p** suppress pronunciation by rule
- **—s** suppress spelling
- **—v** suppress voice output

The following input will reconstitute a coded vocabulary, 'speak.m', from an ascii listing, 'speak.v', that was created using !l. 'Null' names a nonexistent vocabulary file.

       cat speak.v — | speak —v null
       !w speak.m

FILES
       /usr/lib/speak.m

SEE ALSO
       M. D. McIlroy, "Synthetic English Speech by Rule," Computing Science Technical Report #14, Bell Laboratories, 1973
       vs (VII), vs (IV)

BUGS
       Excessively long words cause dumps.
       Space is not reclaimed from deleted entries; use !w and !r to effect reclamation.
       The first phoneme is sometimes dropped when !p is used after !d.

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 *plot* (I).

    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
    $$y_0'' = k y_1'', \quad y_n'' = k y_{n-1}''$$
    is set by the next argument. By default $k = 0$.

    n   Space output points so that approximately n points 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
    plot (I)

AUTHOR
    M. D. McIlroy.

BUGS
    A limit of 1000 input points is enforced silently.

NAME

    tmg — compiler-compiler

SYNOPSIS

    tmg name

DESCRIPTION

Tmg produces a translator for the language whose parsing and translation rules are described in file name.t. The new translator appears in a.out and may be used thus:

a.out input [output]

Except in rare cases input must be a randomly addressable file. If no output file is specified, the standard output file is assumed.

FILES

    /sys/tmg/tmgl.o the compiler-compiler
    /sys/tmg[abc]   libraries
    alloc.d   table storage

SEE ALSO

A Manual for the Tmg Compiler-writing Language, internal memorandum.

DIAGNOSTICS

Syntactic errors result in "???" followed by the offending line.
Situations such as space overflow with which the Tmg processor or a Tmg-produced processor can not cope result in a descriptive comment and a dump.

AUTHOR

    M. D. McIlroy

BUGS

9.2 footnote 1 is not enforced, causing trouble.
Restrictions (7.) against mixing bundling primitives should be lifted.
Certain hidden reserved words exist: gpar, classtab, trans.
Octal digits include 8=10 and 9=11.

NAME
      ttt — tic-tac-toe

SYNOPSIS
      /usr/games/ttt

DESCRIPTION
      Ttt is the X and O game popular in the first grade. This is a learning program that never makes
      the same mistake twice.

      Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

FILES
      /usr/games/ttt.k learning file

BUGS

NAME

      wump — hunt the wumpus

SYNOPSIS

      /usr/games/wump

DESCRIPTION

      Wump plays the game of "Hunt the Wumpus." A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

      The program asks various questions which you answer one per line; it will give a more detailed description if you want.

      This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

BUGS

      It will never replace Space War.

NAME

yacc — yet another compiler-compiler

SYNOPSIS

yacc [−v] [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 is *y.tab.c*, which must be compiled by the C compiler and loaded with any other routines required (perhaps a lexical analyzer) and the Yacc library:

       cc y.tab.c other.o −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.

SEE ALSO

"LR Parsing", by A. V. Aho and S. C. Johnson, to appear in Computing Surveys. "The YACC Compiler-compiler", internal memorandum.

AUTHOR

S. C. Johnson

FILES

y.output
y.tab.c
/lib/liby.a   runtime library for compiler

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.

BUGS

NAME
ascii — map of ASCII character set

SYNOPSIS
cat /usr/pub/ascii

DESCRIPTION
Ascii is a map of the ASCII character set, to be printed as needed. It contains:

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us
|040 sp |041  ! |042  " |043  # |044  $ |045  % |046  & |047  '
|050  ( |051  ) |052  * |053  + |054  , |055  — |056  . |057  /
|060  0 |061  1 |062  2 |063  3 |064  4 |065  5 |066  6 |067  7
|070  8 |071  9 |072  : |073  ; |074  < |075  = |076  > |077  ?
|100  @ |101  A |102  B |103  C |104  D |105  E |106  F |107  G
|110  H |111  I |112  J |113  K |114  L |115  M |116  N |117  O
|120  P |121  Q |122  R |123  S |124  T |125  U |126  V |127  W
|130  X |131  Y |132  Z |133  [ |134  \ |135  ] |136  ^ |137  _
|140  ` |141  a |142  b |143  c |144  d |145  e |146  f |147  g
|150  h |151  i |152  j |153  k |154  l |155  m |156  n |157  o
|160  p |161  q |162  r |163  s |164  t |165  u |166  v |167  w
|170  x |171  y |172  z |173  { |174  | |175  } |176  ~ |177 del
```

FILES
found in /usr/pub

NAME

    greek — graphics for extended ascii type-box

SYNOPSIS

    cat /usr/pub/greek

DESCRIPTION

    Greek gives the mapping from ascii to the "shift out" graphics in effect between SO and SI on model 37 Teletypes with a 128-character type-box. It contains:

alpha	α	A	beta	β	B	gamma	γ	\
GAMMA	Γ	G	delta	δ	D	DELTA	Δ	W
epsilon	ϵ	S	zeta	ζ	Q	eta	η	N
THETA	Θ	T	theta	θ	O	lambda	λ	L
LAMBDA	Λ	E	mu	μ	M	nu	ν	@
xi	ξ	X	pi	π	J	PI	Π	P
rho	ρ	K	sigma	σ	Y	SIGMA	Σ	R
tau	τ	I	phi	ϕ	U	PHI	Φ	F
psi	ψ	V	PSI	Ψ	H	omega	ω	C
OMEGA	Ω	Z	nabla	∇	[	not	$\neg$	_
partial	∂	]	integral	$\int$	^			

SEE ALSO

    ascii (VII)

NAME
     tabs — set tab stops

SYNOPSIS
     cat /usr/pub/tabs

DESCRIPTION
     When printed on a suitable terminal, this file will set tab stops every 8 columns. Suitable terminals include the Teletype model 37 and the GE TermiNet 300.

     These tab stop settings are desirable because UNIX assumes them in calculating delays.

NAME
        tmheader — TM cover sheet

SYNOPSIS
        ed /usr/pub/tmheader

DESCRIPTION
        /usr/pub/tmheader contains a prototype for making a *troff(1)* formatted cover sheet for a technical
        memorandum. Parameters to be filled in by the user are marked by self-explanatory names be-
        ginning with "—".

BUGS
        God help you on two-page abstracts. Try to write less.

NAME
        vs — voice synthesizer code

DESCRIPTION
        The octal codes below are understood by the Votrax® voice synthesizer. Inflection and phonemes are or-ed together. The mnemonics in the first column are used by *speak* (VI); the upper case mnemonics are used by the manufacturer.

0	300	4—strong inflection		u0	014	UH—but
1	200	3		u1	015	UH1—uncle
2	100	2		u2	016	UH2—stirrup
3	000	1—weak inflection		u3	034	UH3—app_le ab_le
				yu	027	U—use
a0	033	AH—contact		iu	010	U1—unite(,y1,iu,...)
a1	052	AH1—connect		ju	011	IU—new
aw	002	AW—law(,l,u2,aw)		b	061	B
au	054	AW1—fault		d	041	D
ae	021	AE—cat		f	042	F
ea	020	AE1—antenna		g	043	G
ai	037	A—name(,n,ai,y0,m)		h	044	H
aj	071	A1—namely		k	046	K
e0	004	EH—met enter		l	047	L
e1	076	EH1—seven		m	063	M
e2	077	EH2—seven		n	062	N
er	005	ER—weather		p	032	P
eu	073	OOH—Goethe cheveux		q	075	Q
eh	067	EHH—le cheveux		r	024	R
y0	023	EE—three		s	040	S
y1	026	Y—sixty		t	025	T
y2	035	Y1—yes		v	060	V
ay	036	AY—may		w	022	W
i0	030	I—six		z	055	Z
i1	064	I1—inept inside		sh	056	SH—show ship
i2	065	I2—static		zh	070	ZH—pleasure
iy	066	IY—cry(,k,r,a0,iy)		j	045	J—edge
ie	003	IE—zero		ch	057	CH—batch
ih	072	IH—station		th	006	TH—thin
o0	031	O—only no		dh	007	THV—then
o1	012	O1—hello		ng	053	NG—long ink
o2	013	O2—notice		−0	017	PA2—long pause
ou	051	OO1—good should		−1	001	PA1
oo	050	OO—look		−2	074	PA0—short pause

SEE ALSO
        speak (VI), vs (IV)
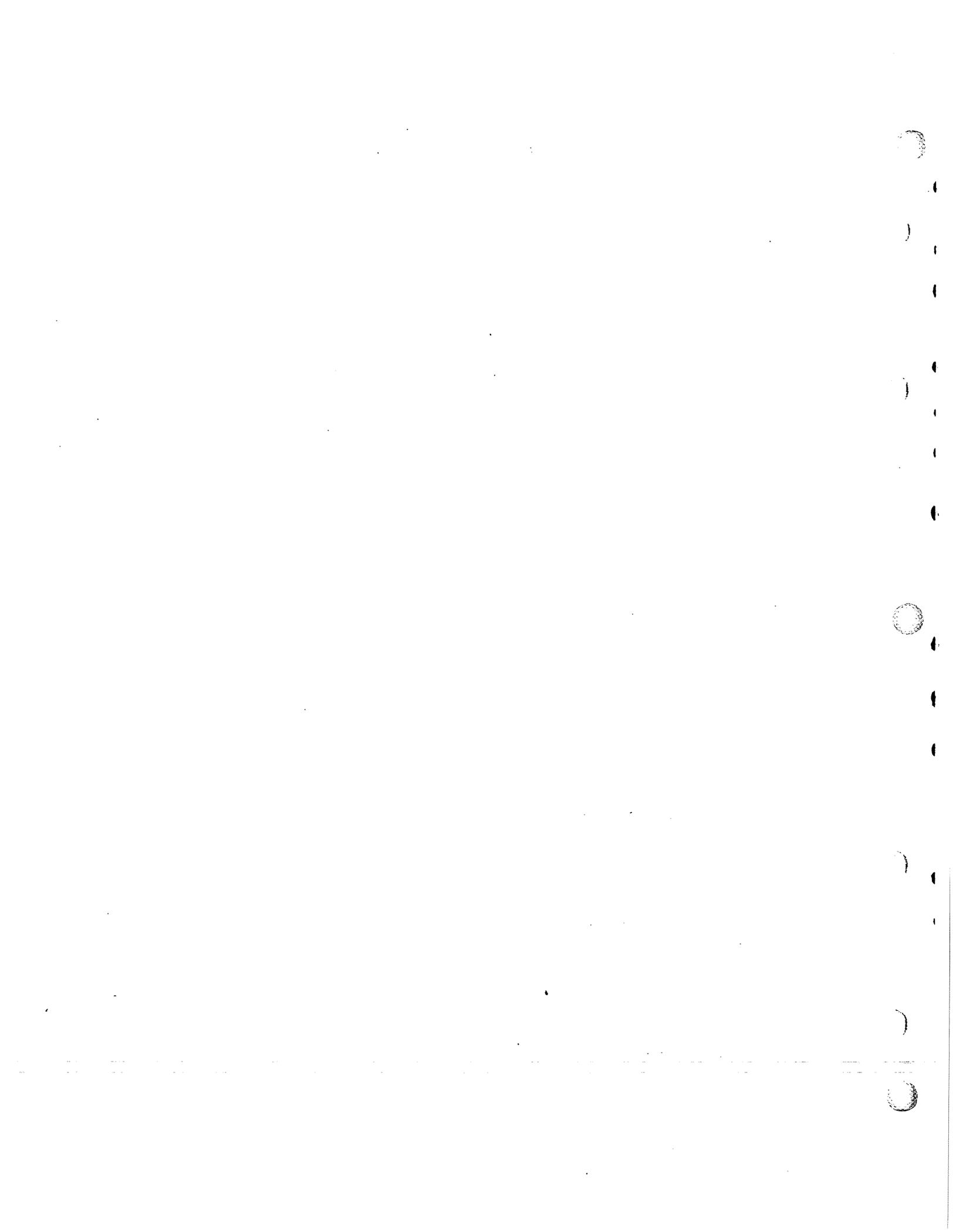
NAME
     20boot − install new 11/20 system

SYNOPSIS
     20boot

DESCRIPTION
     This shell command file copies the current version of the 11/20 program used to run the VT01 display onto the /dev/vt0 file. The 11/20 should have been started at its ROM location 773000.

FILES
     /dev/vt0, /usr/mdec/20.o (11/20 program)

SEE ALSO
     vt (IV)

NAME
        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. If no *wtmp* file is given, **/usr/adm/wtmp** is used.

        The accounting file */usr/adm/wtmp* is maintained by *init* and *login*. Neither of these programs
        creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it
        should be created with length 0. On the other hand if the file is left undisturbed it will grow
        without bound, so periodically any information desired should be collected and the file truncated.

FILES
        /usr/adm/wtmp

SEE ALSO
        init (VIII), login (I), wtmp (V).

BUGS

NAME
    boot procedures — UNIX startup

DESCRIPTION
    How to start UNIX.   UNIX is started by placing it in core starting at location zero and transferring to zero.   There are various ways to do this.   If UNIX is still intact after it has been running, the most obvious method, is simply to transfer to zero, but this is not recommended if the system has crashed.

The *tp* command places a bootstrap program on the otherwise unused block zero of the tape. The DECtape version of this program is called *tboot,* the magtape version *mboot.* If *tboot* or *mboot* is read into location zero and executed there, it will type '=' on the console, read in a *tp* entry name, load that entry into core, and transfer to zero.   Thus the next easiest way to run UNIX is to maintain the UNIX code on a tape using *tp.* Then when a boot is required, execute (somehow) a program which reads in and jumps to the first block of the tape.   In response to the '=' prompt, type the entry name of the system on the tape (we use plain 'unix').   It is strongly recommended that a current version of the system be maintained in this way, even if the first or third methods of booting the system are usually used.

The standard DEC ROM which loads DECtape is sufficient to read in *tboot,* but the magtape ROM loads block one, not zero.   If no suitable ROM is available, magtape and DECtape programs are presented below which may be manually placed in core and executed.

A third method of rebooting the system involves the otherwise unused block zero of each UNIX file system.   The single-block program *uboot* will read a UNIX pathname from the console, find the corresponding file on a device, load that file into core location zero, and transfer to it.   The current version of this boot program reads a single character (either **p** or **k** for RP or RK, both drive 0) to specify which device is to be searched.   *Uboot* operates under very severe space constraints.   It supplies no prompts, except that it echos a carriage return and line feed after the **p** or **k.**   No diagnostic is provided if the indicated file cannot be found, nor is there any means of correcting typographical errors in the file name except to start the program over.   *Uboot* can reside on any of the standard file systems or may be loaded from a *tp* tape as described above.

The standard DEC disk ROMs will load and execute *uboot* from block zero.

The switches.   The console switches play an important role in the use and especially the booting of UNIX.   During operation, the console switches are examined 60 times per second, and the contents of the address specified by the switches are displayed in the display register. (This is not true on the 11/40 since there is no display register on that machine.) If the switch address is even, the address is interpreted in kernel (system) space; if odd, the rounded-down address is interpreted in the current user space.

If any diagnostics are produced by the system, they are printed on the console only if the switches are non-zero.   Thus it is wise to have a non-zero value in the switches at all times.

During the startup of the system, the *init* program (VIII) reads the switches and will come up single-user if the switches are set to 173030.

It is unwise to have a non-existent address in the switches.   This causes a bus error in the system (displayed as 177777) at the rate of 60 times per second.   If there is a transfer of more than 16ms duration on a device with a data rate faster than the bus error timeout (about 10μs) then a permanent disk non-existent-memory error will occur.

ROM programs.   Here are some programs which are suitable for installing in read-only memories, or for manual keying into core if no ROM is present.   Each program is position-independent but should be placed well above location 0 so it will not be overwritten.   Each reads a block from the beginning of a device into core location zero.   The octal words constituting the program are listed on the left.

DECtape (drive 0) from endzone:

```
012700          mov     $tcba,r0
177346
010040          mov     r0,-(r0)        / use tc addr for wc
012710          mov     $3,(r0)         / read bn forward
000003
105710   1:     tstb    (r0)            / wait for ready
002376          bge     1b
112710          movb    $5,(r0)         / read (forward)
000005
000777          br                      / loop; now halt and start at 0
```

DECtape (drive 0) with search:

```
012700   1:     mov     $tcba,r0
177346
010040          mov     r0,-(r0)        / use tc addr for wc
012740          mov     $4003,-(r0)     / read bn reverse
004003
005710   2:     tst     (r0)
002376          bge     2b              / wait for error
005760          tst     -2(r0)          / loop if not end zone
177776
002365          bge     1b
012710          mov     $3,(r0)         / read bn forward
000003
105710   2:     tstb    (r0)            / wait for ready
002376          bge     2b
112710          movb    $5,(r0)         / read (forward)
000005
105710   2:     tstb    (r0)            / wait for ready
002376          bge     2b
005007          clr     pc              / transfer to zero
```

Caution: both of these DECtape programs will (literally) blow a fuse if 2 drives are dialed to zero.

Magtape from load point:

```
012700          mov     $mtcma,r0
172526
010040          mov     r0,-(r0)        / usr mt addr for wc
012740          mov     $60003,-(r0)    / read 9-track
060003
000777          br                      / loop; now halt and start at 0
```

RK (drive 0):

```
012700          mov     $rkda,r0
177412
005040          clr     -(r0)           / rkda cleared by start
010040          mov     r0,-(r0)
012740          mov     $5,-(r0)
000005
105710   1:     tstb    (r0)
002376          bge     1b
005007          clr     pc
```

RP (drive 0)

```
012700          mov     $rpmr,r0
176726
005040          clr     -(r0)
005040          clr     -(r0)
005040          clr     -(r0)
```

```
010040          mov     r0,—(r0)
012740          mov     $5,—(r0)
000005
105710    1:    tstb    (r0)
002376          bge     1b
005007          clr     pc
```

FILES

/unix — UNIX code
/usr/mdec/mboot — *tp* magtape bootstrap
/usr/mdec/tboot — *tp* DECtape bootstrap
/usr/mdec/uboot — file system bootstrap

SEE ALSO

tp (I), init (VIII)

NAME
    check — file system consistency check

SYNOPSIS
    check [—lsuib [numbers]] [filesystem]

DESCRIPTION
    Check examines a file system, builds a bit map of used blocks, and compares this bit map against
    the free list maintained on the file system. It also reads directories and compares the link-count
    in each i-node with the number of directory entries by which it is referenced. If the file system
    is not specified, a check of a default file system is performed. The normal output of *check* in-
    cludes a report of

    The number of blocks missing; i.e. not in any file nor in the free list,
    The number of special files,
    The total number of files,
    The number of large files,
    The number of directories,
    The number of indirect blocks,
    The number of blocks used in files,
    The highest-numbered block appearing in a file,
    The number of free blocks.

    The —l flag causes *check* to produce as part of its output report a list of the all the path names of
    files on the file system. The list is in i-number order; the first name for each file gives the i-
    number while subsequent names (i.e. links) have the i-number suppressed. The entries "." and
    ".." for directories are also suppressed. If the flag as given as —ll, the listing will include the ac-
    cessed and modified times for each file. The —l option supersedes —s.

    The —s flag causes *check* to ignore the actual free list and reconstruct a new one by rewriting the
    super-block of the file system. The file system should be dismounted while this is done; if this is
    not possible (for example if the root file system has to be salvaged) 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. Notice also that the words in the super-
    block which indicate the size of the free list and of the i-list are believed. If the super-block has
    been curdled these words will have to be patched. The —s flag causes the normal output reports
    to be suppressed.

    With the —u flag, *check* examines the directory structure for connectivity. A list of all i-node
    numbers that cannot be reached from the root is printed. This is exactly the list of i-nodes that
    should be cleared (see clri (VIII)) after a series of incremental restores. (See the bugs section of
    restor (VIII).) The —u option supersedes —s.

    The occurrence of i *n* times in a flag argument —ii...i causes *check* to store away the next *n* argu-
    ments which are taken to be i-numbers. When any of these i-numbers is encountered in a direc-
    tory a diagnostic is produced, as described below, which indicates among other things the entry
    name.

    Likewise, *n* appearances of b in a flag like —bb...b cause the next *n* arguments to be taken as
    block numbers which are remembered; whenever any of the named blocks turns up in a file, a
    diagnostic is produced.

FILES
    Currently, /dev/rp0 is the default file system.

SEE ALSO
    fs (V), clri (VIII), restor(VIII)

DIAGNOSTICS
    If a read error is encountered, the block number of the bad block is printed and *check* exits.
    "Bad freeblock" means that a block number outside the available space was encountered in the
    free list. "*n* dups in free" means that *n* blocks were found in the free list which duplicate blocks

either in some file or in the earlier part of the free list.

An important class of diagnostics is produced by a routine which is called for each block which is encountered in an i-node corresponding to an ordinary file or directory. These have the form

> b# complaint ; i= i# (class)

Here b# is the block number being considered; complaint is the diagnostic itself. It may be

blk if the block number was mentioned as an argument after −b;

bad if the block number has a value not inside the allocatable space on the device, as indicated by the devices's super-block;

dup if the block number has already been seen in a file;

din if the block is a member of a directory, and if an entry is found therein whose i-number is outside the range of the i-list on the device, as indicated by the i-list size specified by the super-block. Unfortunately this diagnostic does not indicate the offending entry name, but since the i-number of the directory itself is given (see below) the problem can be tracked down.

The i# in the form above is the i-number in which the named block was found. The class is an indicator of what type of block was involved in the difficulty:

sdir indicates that the block is a data block in a small file;

ldir indicates that the block is a data block in a large file (the indirect block number is not available);

idir indicates that the block is an indirect block (pointing to data blocks) in a large file;

free indicates that the block was mentioned after −b and is free;

urk indicates a malfunction in check.

When an i-number specified after −i is encountered while reading a directory, a report in the form

> i# ino; i= d# (class) name

where i# is the requested i-number. d# is the i-number of the directory, class is the class of the directory block as discussed above (virtually always "sdir") and name is the entry name. This diagnostic gives enough information to find a full path name for an i-number without using the -l option: use −b n to find an entry name and the i-number of the directory containing the reference to n, then recursively use −b on the i-number of the directory to find its name.

Another important class of file system diseases indicated by check is files for which the number of directory entries does not agree with the link-count field of the i-node. The diagnostic is hard to interpret. It has the form

> i# delta

Here i# is the i-number affected. Delta is an octal number accumulated in a byte, and thus can have the value 0 through 377(8). The easiest way (short of rewriting the routine) of explaining the significance of delta is to describe how it is computed.

If the associated i-node is allocated (that is, has the allocated bit on) add 100 to delta. If its link-count is non-zero, add another 100 plus the link-count. Each time a directory entry specifying the associated i-number is encountered, subtract 1 from delta. At the end, the i-number and delta are printed if delta is neither 0 nor 200. The first case indicates that the i-node was unallocated and no entries for it appear; the second that it was allocated and that the link-count and the number of directory entries agree.

Therefore (to explain the symptoms of the most common difficulties) delta = 377 (−1 in 8-bit, 2's complement octal) means that there is a directory entry for an unallocated i-node. This is somewhat serious and the entry should be be found and removed forthwith. Delta = 201 usually means that a normal, allocated i-node has no directory entry. This difficulty is much less serious. Whatever blocks there are in the file are unavailable, but no further damage will occur if nothing is done. A clri followed by a check −s will restore the lost space at leisure.

In general, values of *delta* equal to or somewhat above 0, 100, or 200 are relatively innocuous; just below these numbers there is danger of spreading infection.

BUGS

Check −*l* or −*u* on large file systems takes a great deal of core.

Since *check* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

NAME
     clri — clear i-node

SYNOPSIS
     clri i-number [filesystem]

DESCRIPTION
     Clri writes zeros on the 32 bytes occupied by the i-node numbered *i-number*. If the *file system* argument is given, the i-node resides on the given device, otherwise on a default file system. The file system argument must be a special file name referring to a device containing a file system. After *clri*, any blocks in the affected file will show up as "missing" in a *check* of of the file system.

     Read and write permission is required on the specified file system device. The i-node becomes allocatable.

     The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

BUGS
     Whatever the default file system is, it is likely to be wrong. Specify the file system explicitly.

     If the file is open, *clri* is likely to be ineffective.

NAME

     df — disk free

SYNOPSIS

     df [filesystem]

DESCRIPTION

     Df prints out the number of free blocks available on a file system. If the file system is unspecified, the free space on all of the normally mounted file systems is printed.

FILES

     /dev/rf?, /dev/rk?, /dev/rp?

SEE ALSO

     check (VIII)

BUGS

NAME
     dpd — spawn data phone daemon

SYNOPSIS
     /etc/dpd

DESCRIPTION
     Dpd is the 201 data phone' daemon. It is designed to submit jobs to the Honeywell 6070 computer via the GRTS interface.

     Dpd uses the directory *lusr/dpd*. The file *lock* in that directory is used to prevent two daemons from becoming active. After the daemon has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with **df.** Each such file is submitted as a job. Each line of a job file must begin with a key character to specify what to do with the remainder of the line.

     S   directs *dpd* to generate a unique snumb card. This card is generated by incrementing the first word of the file *lusr/dpd/snumb* and converting that to three-digit octal concatenated with the station ID.

     L   specifies that the remainder of the line is to be sent as a literal.

     B   specifies that the rest of the line is a file name. That file is to be sent as binary cards.

     F   is the same as B except a form feed is prepended to the file.

     U   specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.

     Any error encountered will cause the daemon to drop the call, wait up to 20 minutes and start over. This means that an improperly constructed *df* file may cause the same job to be submitted every 20 minutes.

     While waiting, the daemon checks to see that the *lock* file still exists. If it is gone, the daemon will exit.

FILES
     /dev/dn0, /dev/dp0, /usr/dpd/*

SEE ALSO
     opr (I)

NAME
  dump — incremental file system dump

SYNOPSIS
  dump [key [arguments] filesystem]

DESCRIPTION
  Dump will make an incremental file system dump on magtape of all files changed after a certain date. The argument *key,* specifies the date and other options about the dump. *Key* consists of characters from the set **iu0hds.**

  i   the dump date is taken from the file **/etc/ddate.**

  u   the date just prior to this dump is written on **/etc/ddate** upon successful completion of this dump.

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

  s   the size of the dump tape is specified in feet. The number of feet is taken from the next argument in *arguments.* It is assumed that there are 9 standard UNIX records per foot. When the specified size is reached, the dump will wait for reels to be changed. The default size is 1700 feet.

  If no arguments are given, the *key* is assumed to be **i** and the file system is assumed to be **/dev/rp0.**

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

        dump 0u /dev/rp0
        check —l /dev/rp0

  The *check* will come in handy in case it is necessary to restore individual files from this dump. Incremental dumps should then 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 tape and only the latest incremental tape.

FILES
  /dev/mt0   magtape
  /dev/rp0   default file system
  /etc/ddate

SEE ALSO
  restor (VIII), check (VIII), dump (V)

BUGS

NAME

     getty — set typewriter mode

SYNOPSIS

     /etc/getty

DESCRIPTION

     Getty is invoked by init (VIII) immediately after a typewriter is opened following a dial-up. The user's login name is read and the login (I) command is called with this name as an argument. While reading this name *getty* attempts to adapt the system to the speed and type of terminal being used.

     Getty initially sets the speed of the interface to 300 baud, specifies that raw mode is to be used (break on every character), that echo is to be suppressed, and either parity allowed. It types the "login:" message (which includes the characters which put the Terminet 300 terminal into full-duplex and return the GSI terminal to non-graphic mode. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the "break" ("interrupt") key. The speed is then changed to 150 baud and the "login:" is typed again, this time including the character sequence which puts a Teletype 37 into full-duplex. If a subsequent null character is received, the speed is changed back to 300 baud.

     The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see stty (II)).

     The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is nonempty, the system is told to map any future upper-case characters into the corresponding lower-case characters. Thus UNIX is usable from upper-case-only terminals.

     Finally, login is called with the user's name as argument.

SEE ALSO

     init (VIII), login (I), stty (II)

BUGS

NAME
       init — process control initialization

SYNOPSIS
       /etc/init

DESCRIPTION
       Init is invoked inside UNIX as the last step in the boot procedure. Generally its role is to create
       a process for each typewriter on which a user may log in.

       First, *init* checks to see if the console switches contain 173030. (This number is likely to vary
       between systems.) If so, the console typewriter **/dev/tty8** is opened for reading and writing and
       the Shell is invoked immediately. This feature is used to bring up a single-user system. When
       the system is brought up in this way, the *getty* and *login* routines mentioned below and described
       elsewhere are not used. If the Shell terminates, *init* starts over looking for the console switch
       setting.

       Otherwise, *init* invokes a Shell, with input taken from the file *letc/rc*. This command file per-
       forms housekeeping like removing temporary files, mounting file systems, and starting daemons.

       Then *init* reads the file *letc/ttys* and forks several times to create a process for each typewriter
       specified in the file. Each of these processes opens the appropriate typewriter for reading and
       writing. These channels thus receive file descriptors 0 and 1, the standard input and output.
       Opening the typewriter will usually involve a delay, since the *open* is not completed until some-
       one is dialed up and carrier established on the channel. Then the process executes the program
       specified by its line in *ttys;* the only program currently specifiable is *letc/getty* (q.v.). *Getty* reads
       the user's name and invokes *login* (q.v.) to log in the user and execute the Shell.

       Ultimately the Shell will terminate because of an end-of-file either typed explicitly or generated
       as a result of hanging up. The main path of *init*, which has been waiting for such an event,
       wakes up and removes the appropriate entry from the file *utmp*, which records current users, and
       makes an entry in *lusr/adm/wtmp*, which maintains a history of logins and logouts. Then the ap-
       propriate typewriter is reopened and *getty* is reinvoked.

       Init catches the *hangup* signal (signal #1) and interprets it to mean that the switches should be ex-
       amined as in a reboot: if they indicate a multi-user system, the *letc/ttys* file is read again. The
       Shell process on each line which used to be active in *ttys* but is no longer there is terminated; a
       new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is
       possible to drop or add phone lines without rebooting the system by changing the *ttys* file and
       sending a *hangup* signal to the *init* process: use "kill −1 1."

FILES
       /dev/tty?, /tmp/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO
       login (I), kill (I), sh (I), ttys (V), getty (VIII)

NAME
     lpd — line printer daemon

SYNOPSIS
     /etc/lpd

DESCRIPTION
     Lpd is the line printer daemon (spool area handler) invoked by *lpr*. It uses the directory *lusr/lpd*. The file *lock* in that directory is used to prevent two daemons from becoming active simultaneously. After the daemon has successfully set the lock, it scans the directory for files beginning with "df." Lines in each *df* file specify files to be printed in the same way as is done by the data-phone daemon dpd (VIII).

FILES
     /usr/lpd/*   spool area
     /dev/lp   printer

SEE ALSO
     dpd (III), lpr (VI)

BUGS

NAME
       mount — mount file system

SYNOPSIS
       /etc/mount special file [**−r**]

DESCRIPTION
       Mount announces to the system that a removable file system is present on the device correspond-
       ing to special file *special* (which must refer to a disk or possibly DECtape). The *file* must exist
       already; it becomes the name of the root of the newly mounted file system.

       Mount maintains a table of mounted devices; if invoked without an argument it prints the table.

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

SEE ALSO
       mount (II), mtab (V), umount (VIII)

BUGS
       Mounting file systems full of garbage will crash the system.

NAME
msh — mini-shell

SYNOPSIS
/etc/msh

DESCRIPTION
Msh is a heavily simplified version of the Shell. It reads one line from the standard input file, interprets it as a command, and calls the command.

The mini-shell supports few of the advanced features of the Shell; none of the following characters is special:

> < $ \ ; & | ^

However, "*", "[", and "?" are recognized and *glob* is called. The main use of *msh* is to provide a command-executing facility for various interactive sub-systems.

SEE ALSO
sh (I), glob (VIII)

NAME
　　　　sync — update the super block

SYNOPSIS
　　　　sync

DESCRIPTION
　　　　Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. See sync (II) for details.

SEE ALSO
　　　　sync (II)

BUGS

NAME

umount — dismount file system

SYNOPSIS

/etc/umount special

DESCRIPTION

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

SEE ALSO

mount (VIII), umount (II), mtab (V)

FILES

/etc/mtab   mounted device table

DIAGNOSTICS

It complains if the special file is not mounted or if it is busy. The file system is busy if there is an open file on it or if someone has his current directory there.

BUGS

NAME

update — periodically update the super block

SYNOPSIS

update

DESCRIPTION

Update is a program that executes the *sync* primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file. See sync (II) for details.

SEE ALSO

sync (II), init (VIII)

BUGS

With *update* running, if the CPU is halted just as the *sync* is executed, a file system can be damaged. This is partially due to DEC hardware that writes zeros when NPR requests fail. A fix would be to have *sync* temporarily increment the system time by at least 30 seconds to trigger the execution of *update*. This would give 30 seconds grace to halt the CPU.