UNIX PROGRAMMER'S MANUAL

Third Edition

K. Thompson

D. M. Ritchie

February, 1973

February 1973

PREFACE
to the Third Edition

In the months since the last appearance of this manual, many changes have occurred both in the system itself and in the way it is used.

Perhaps most obviously, there have been additions, deletions, and modifications to the system and its software. It is these changes, of course, that caused the appearance of this revised manual.

Second, the number of people spending an appreciable amount of time writing UNIX software has increased. Credit is due to L. L. Cherry, M. D. McIlroy, L. E. McMahon, R. Morris, J. F. Ossanna, and E. N. Pinson for their contributions.

Finally, the number of UNIX installations has grown to 16, with more 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.

In particular, any system which uses a PDP-11/20 processor will not include all the software described herein, nor will the software behave the same way. The second, or even the first, edition of this manual is likely to be more appropriate.

Besides additions, deletions, and modifications to the writeups in each section, this manual differs from its predecessors in two ways: all the commands used for system maintenance and not intended for normal users have been moved to a new section VIII; and there is a new "How to Get Started" chapter that gives some elementary facts and many pointers to other sections.

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 Inevitably, this means that many sections will soon be out of date. (The rate of change of the system is so great that a dismayingly large number of early sections had to be modified while the rest were being written. The unbounded effort required to stay up-to-date is best indicated by the fact that several of the programs described were written specifically to aid in preparation of this manual!)

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 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.

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 special files section IV discusses the characteristics of each system "file" which actually refers to an I/O device. Unlike previous editions, 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 have a common format.

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:

Underlined 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 diagnostics that may be produced. This section tends to be as terse as the diagnostics themselves.

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

Previous edition of this manual had an owner section, which has been dropped from this edition because the "owners" of many routines became fairly hard to pin down. The major contributors to UNIX, (cast in order of appearance) together with their login names and most notable contributions, are

ken	K. Thompson	(UNIX, many commands)
dmr	D. M. Ritchie	(many commands, as, ld, C)
jfo	J. F. Ossanna	(roff, nroff)
doug	M. D. McIlroy	(tmg, m6)
rhm	R. Morris	(dc, much of library)
lem	L. E. McMahon	(cref)
llc	L. L. Cherry	(form, fed, salloc)
csr	C. S. Roberts	(tss)
enp	E. N. Pinson	(proof)

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 roff.

The assistance of R. Morris is gratefully acknowledged.

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 on the one hand, and IBM 2741-type terminals on the other.

To use UNIX, you must have a valid UNIX 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.

TTY 37 terminal

UNIX will type out "login: "; 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).

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 MH-TSS requires half-duplex). When a connection with UNIX is established, a few garbage characters are typed (the login message at the wrong speed). Depress the "break" key; this is a speed-independent signal to UNIX that a 300-baud terminal is in use. UNIX will type "login: " at the correct speed; you type your user name, followed by the "return" key. Henceforth, the "return", "new line", or "linefeed" keys will give exactly the same results. Each line must be terminated with one of these keys; no one is listening to you until the return is received.

Selectric terminals

From an IBM 2741 or the Anderson-Jacobson Selectric terminal, no message will appear. After the data connection is established, press the "return" key. UNIX should type "login: " as described above. If the greeting does not

appear after a few seconds, unlock the keyboard by switching the terminal to local and back to remote, and type "return". If necessary, hang up and try again; something has gone wrong.

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 a UNIX program, the Shell, will type a "%" to you. (The Shell is described below under "How to run a program".

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

Logging out

There are three ways to log out:

You can simply hang up the phone. Hanging up is safe if you are at command level, that is, if the Shell has just typed its prompt signal "%". It is also safe if you are in interactive system programs, for example the editor. It is unsafe if you are executing a non-interactive program, or one of your own programs, which either does not read the typewriter or ignores the end-of-file indications which will result from hanging up. The reason is that UNIX, unlike most systems, does not terminate a program simply because it has been hung-up upon.

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 (login (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, as described above in Logging in.

UNIX typewriter I/O is full-duplex (except for Selectric terminals). 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 stops echoing input characters, and starts echoing "#" no matter what you typed. The last character which was echoed correctly will be received correctly by the program to which you were talking; subsequent characters have been thrown away.

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 TerminNet 300 terminals, will set the tab stops correctly (tabs(VII)).

Section dc(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 reference 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 file "filex" in directory "lem"; "lem" is itself a sub-directory 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) and stat(I). See mkdir(I) for making directories; rmdir(I) for destroying them.

For a fuller discussion of the file system, see MM-71-1273-4. 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 the assembly-language 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. For assembly language programs, 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(I), pr(I), roff(I), or nroff(I).

The cat command simply dumps ASCII text on the terminal, with no processing at all. The pr command paginates the text and supplies headings. The nroff command is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file. The roff command is a somewhat less elaborate 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. To leave a message the presence of which will be announced to another user when he next logs in, mail(I) is used. 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

IV. SPECIAL FILES

V. FILE FORMATS

VI. USER MAINTAINED PROGRAMS

VII. MISCELLANEOUS

VIII. SYSTEM MAINTAINANCE

NAME : -- place a label

SYNOPSIS : [label]

DESCRIPTION The purpose of : is to place a label for the goto command. It has no effect when executed.

FILES --

SEE ALSO goto(I)

DIAGNOSTICS --

BUGS --

NAME ar -- archive

SYNOPSIS ar key afile name$_1$ ...

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)

DIAGNOSTICS "Bad usage", "afile -- not in archive format", "cannot open temp file", "name -- cannot open",

"name -- phase error", "name -- cannot create", "no archive file", "cannot create archive file", "name -- not found".

BUGS

Option vt 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.

"ar x" changes the modified-date of the current directory to a random number.

NAME

as -- assembler

SYNOPSIS

as [-] name$_1$ ...

DESCRIPTION

as assembles the concatenation of name$_1$, ... 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.

FILES

/etc/as2	pass 2 of the assembler
/tmp/atm1?	temporary
/tmp/atm2?	temporary
/tmp/atm3?	temporary
a.out	object

SEE ALSO

ld(I), nm(I), un(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	Odd-- word quantity assembled at odd address
P	Phase error-- "." different in pass 1 and 2
R	Relocation error
U	Undefined symbol
X	syntaX error

BUGS

Symbol table overflow is not checked.

NAME bas -- basic

SYNOPSIS bas [file]

DESCRIPTION bas is a dialect of basic [1]. 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.

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]]
list 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.)

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 re-set. Control is passed to the lowest numbered internal statement.

Expressions have the following syntax:

name
A name is used to specify a variable. Names are composed of a letter ('a' - 'z') 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 composed of digits, at

most one decimal point ('.') and possibly a scale factor of the form e digits or e- digits.

(expression)
Parentheses are used to alter normal order of evaluation.

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
Arg(i) is the value of the i th actual parameter on the current level of function call.

exp
Exp(x) is the exponential function of x.

log
Log(x) is the logarithm base e of x.

sin
Sin(x) is the sine of x (radians).

cos
Cos(x) is the cosine of x (radians).

atn
Atn(x) is the arctangent of x.

rnd
Rnd() is a uniformly distributed random number between zero and one.

expr
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
Int(x) returns x truncated to an integer.

FILES /tmp/btm? temporary

SEE ALSO [1] DEC-11-AJPB-D

DIAGNOSTICS Syntax errors cause the incorrect line to be typed with an underscore where the parse failed.

NAME — cat -- concatenate and print

SYNOPSIS — cat $file_1$ ...

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.

FILES — --

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] $sfile_1.c$ ... $ofile_1$ ...

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

Arguments whose names end with ".c" are assumed to be C source programs; they are compiled, and the object program is left on the file $sfile_1.o$ (i.e., the file whose name is that of the source with ".o" substituted for ".c").

Other arguments (except for "-c") are assumed 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.

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

FILES

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

SEE ALSO C reference manual (in preparation), cdb(I)

DIAGNOSTICS Diagnostics are intended to be self-explanatory.

BUGS --

NAME cdb -- C debugger

SYNOPSIS cdb [core [a.out]]

DESCRIPTION cdb is a debugging program for use with C programs. It is by no means completed, and this section is essentially only a placeholder for the actual description.

Cdb resembles db in many respects, except that all integers are decimal.

Even the present cdb has one useful feature: the command

```
$
```

will give a stack trace of the core image of a terminated C program. The calls are listed in the order made; the actual arguments to each routine are given in octal.

FILES --

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

DIAGNOSTICS "?"

BUGS --

NAME chdir -- change working directory

SYNOPSIS chdir directory

DESCRIPTION directory becomes the new working 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.

FILES --

SEE ALSO sh(I)

DIAGNOSTICS "Bad directory" if the directory cannot be changed to.

BUGS --

NAME chmod -- change mode

SYNOPSIS chmod octal file$_1$ ...

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

01 write for non-owner
02 read for non-owner
04 write for owner
10 read for owner
20 executable
40 set-UID

Only the owner of a file may change its mode.

FILES --

SEE ALSO stat(I), ls(I)

DIAGNOSTICS "?"

BUGS --

NAME chown -- change owner

SYNOPSIS chown owner $file_1$ ...

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 is allowed to change the owner. It is illegal to change the owner of a file with the set-user-ID mode.

FILES /etc/passwd

SEE ALSO stat(I)

DIAGNOSTICS "Who?" if owner cannot be found, "file?" if file cannot be found.

BUGS --

NAME cmp -- compare two files

SYNOPSIS cmp $file_1$ $file_2$

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

FILES --

SEE ALSO proof(I)

DIAGNOSTICS Messages are given for inability to open either argument, premature EOF on either argument, and incorrect usage.

BUGS If the shorter of the two files is of odd length, cmp acts as if a null byte had been appended to it.

NAME — cp — copy

SYNOPSIS — cp $file_1$ $file_2$

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 $file_2$ is a directory, then the target file is a file in that directory with the file-name of $file_1$.

FILES — --

SEE ALSO — cat(I), pr(I), mv(I)

DIAGNOSTICS — Error returns are checked at every system call, and appropriate diagnostics are produced.

BUGS — Copying a file onto itself destroys its contents.

NAME cref -- make cross reference listing

SYNOPSIS cref [-soi] name1 ...

DESCRIPTION CREF makes a cross reference listing of files in assembler format (see AS(I)). The files named as arguments in the command line are searched for symbols (defined as a succession of alphabetics, numerics, '.', or '_', beginning with an alphabetic, '.', or '_').

The output report is in four columns:

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

The third column contains the line number in the file by default; the -s option will cause the most recent name symbol to appear there instead.

CREF uses either an ignore file or an only file. If the -i option is given, it will take the next file name to be an ignore file; if the -o option is given, the next file name will be taken as an only file. Ignore and only files should be lists of symbols separated by new lines. If an ignore file is given, all the symbols in the file will be ignored in columns (1) and (3) of the output. If an only file is given, only symbols appearing in the file will appear in column (1), but column (3) will still contain the most recent name encountered. Only one of the options -i or -o may be used. The default setting is -i; all symbols predefined in the assembler are ignored, except system call names, which are collected.

FILES Files t.0, t.1, t.2, t.3 are created (i.e. DESTROYED) in the working directory of anyone using cref. This nuisance will be repaired soon. The output is left in file s.out in the working directory.

/usr/lem/s.tab is the default ignore file.

SEE ALSO as(I)

DIAGNOSTICS "line too long" -- input line >131 characters

"symbol too long" -- symbol >20 characters

"too many symbols" -- >10 symbols in line

"cannot open t.?" -- bug; see LEM

"cannot fork; examine t.out" -- can't start sort

process; intermediate results are on files t.0, t.1, t.2, t.3. These may be sorted independently and the results concatenated by the user.

"cannot sort" -- odd response from sort; examine intermediate results, as above.

"impossible situation" -- system bug

"cannot open" file -- one of the input names cannot be opened for reading.

BUGS

The destruction of unsuspecting users' files should soon be fixed. A limitation that may eventually go away is the restriction to assembler language format. There should be options for FORTRAN, English, etc., lexical analysis.

File names longer than eight characters cause misalignment in the output if tabs are set at every eighth column.

It should write on the standard output, not s.out.

NAME crypt -- encode/decode

SYNOPSIS crypt [password]

DESCRIPTION crypt is an exact implementation of Boris Hagelin's cryptographic machine called the M-209 by the U. S. Army [1].

crypt reads from the standard input file and writes on the standard output. For a given password, the encryption process is idempotent; that is,

```
crypt znorkle <clear >cypher
crypt znorkle <cypher
```

will print the clear.

crypt is suitable for use as a filter:

```
pr <"crypt bandersnatch"<cypher
```

FILES --

SEE ALSO [1] U. S. Patent 2,089,603.

DIAGNOSTICS --

BUGS --

NAME date -- print and set the date

SYNOPSIS date [mmddhhmm]

DESCRIPTION If no argument is given, the current date is printed to the second. If an argument is given, the current date is set. mm is the month number; dd is the day number in the month; hh is the hour number (24 hour system); mm is the minute number. For example:

```
date 10080045
```

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

FILES --

SEE ALSO --

DIAGNOSTICS "?" 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 "*.".

6. 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 octal 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.

Db should not be used to examine special files, for example disks and tapes, since it reads one byte at a time. Use od(I) instead.

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, some systems cause a "squashed" core image to be written. In such a core image, addresses in the stack must be mapped according to the degree of squashing which has been employed. 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 last 512(10) bytes of the core file, can be addressed at apparent locations 160000-160777.

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.

FILES --

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

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

BUGS --

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.

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

lx
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 pushed on the stack. Thus the top value 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 nesting 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 ≥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 to calculate the monthly, weekly and hourly rates for a $10,000/year salary.

```
10000
100*     (now in cents)
dsa      (non-destructive store)
12/      (pennies per month)
la52/    (pennies per week)
d10*     (deci-pennies per week)
375/     (pennies per hour)
f        (print all results)
     512
     19230
```

```
    83333
"a" 1000000
```

An example which prints the first ten values of n! is

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

FILES —

SEE ALSO msh(VII), salloc(III)

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 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)

DIAGNOSTICS --

BUGS --

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 anything else, the file is not removed.

FILES --

SEE ALSO rm(I)

DIAGNOSTICS "?"

BUGS The name "dsw" is a carryover from the ancient past. Its etymology is amusing but the name is nonetheless ill-advised.

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.

FILES — .

SEE ALSO — --

DIAGNOSTICS — --

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_1 ...]

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.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME

ed -- editor

SYNOPSIS

ed [name]

DESCRIPTION

ed is the standard text editor.

If the optional argument is given, ed simulates an e command on the named file; that is to say, the file is read into ed's buffer so that it can be edited.

ed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a write (w) 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 (s, see 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 discussed 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 nth line of the buffer.

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.

5. 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.

7. An address followed by a plus sign "+" or a minus sign "-" followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.

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

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.

(.)kx

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.

(.,.)mA
The move command will reposition the addressed lines after the line addressed by "A". The line originally after the last line moved becomes the current line; if the lines moved were originally at the end, the new last line becomes the current line.

n
The marknames command will print the current mark names.

(.,.)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.

The 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 17 (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.

(.+1)<newline>
An address alone on a line causes that 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) 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, and on the maximum number of characters in a line, in a global's command list, and in a remembered file name. These limitations vary with the physical core size of the PDP11 computer on which ed is being used. The range of limiting sizes for the above mentioned items is; 1300 - 4000 lines per file, 256 - 512 characters per line, 63 - 256 characters per global command list, and 64 characters per file name.

FILES
/tmp/etm? temporary
/etc/msh to implement the "!" command.

SEE ALSO --

DIAGNOSTICS "?" for any error

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.

FILES --

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

DIAGNOSTICS --

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 7.2E16), 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. It takes 1 minute to factor a prime near 10^13.

FILES — --

SEE ALSO — --

DIAGNOSTICS — "Ouch." for input out of range or for garbage input.

BUGS — --

NAME fc — fortran compiler

SYNOPSIS fc [-c] $sfile_1.f$ ... $ofile_1$ ...

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 $sfile_1.o$ (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. The 'standard' implicit statement is recognized.

3. The types doublecomplex, logical*1, integer*2 and real*8 (doubleprecision) 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.

In I/O statements, only unit numbers 0-19 are supported. Unit number nn corresponds to file "fortnn;" (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.

FILES

file.f	input file
a.out	loaded output
f.tmp[123]	temporary (deleted)
/usr/fort/fc[1234]	compilation phases
/usr/lib/fr0.o	runtime startoff
/usr/lib/filib.a	interpreter library
/usr/lib/libf.a	builtin functions, etc.
/usr/lib/liba.a	system library

SEE ALSO

ANSI standard

DIAGNOSTICS

Compile-time diagnostics are given by number. If the source code is available, it is printed with an underline at the current character pointer. Errors possible are:

1	statement too long
2	syntax error in type statement
3	redeclaration
4	missing (in array declarator
5	syntax error in dimension statement
6	inappropriate or gratuitous array declarator
7	syntax error in subscript bound
8	illegal character
9	common variable is a parameter or already in common
10	common syntax error
11	subroutine/blockdata/function not first statement
12	subroutine/function syntax error
13	block data syntax error
14	redeclaration in external
15	external syntax error
16	implicit syntax error
17	subscript on non-array
18	incorrect subscript count
19	subscript out of range
20	subscript syntax error
21	DATA syntax error
22	DATA semantics error
23	Illegal variable in DATA

23	equivalence inconsistency
24	equivalence syntax error
25	separate common blocks equivalenced
26	common block illegally extended by equivalence
27	common inconsistency created by equivalence
28	DATA table overflow
29	() imbalance in expression
30	expression syntax error
31	illegal variable in equivalence
32	Storage initialized twice by DATA
33	non array/function used with subscripts/arguments
35	goto syntax error
37	illegal return
38	continue, return, stop, call, end, or pause syntax error
39	assign syntax error
40	if syntax error
41	I/O syntax error
42	do or I/O iteration error
43	do end missing
50	illegal statement in block data
51	multiply defined labels
52	undefined label
53	dimension mismatch
54	expression syntax error
55	end of statement in hollerith constant
56	array too large
99	β table overflow
101	unrecognized statement

Runtime diagnostics:

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
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
999	unimplemented input conversion

BUGS

The following is a list of those features not yet implemented:

arithmetic statement functions

backspace, endfile, rewind runtime

binary I/O

no scale factors on input

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 single space and ending with a new line. The conventions of the Shell with respect to '*' and '?' hold for all commands but m where literal string names are expected. The commands are:

e $name_1$ ...

edit writes the string whose name is $name_1$ onto a temporary file and executes the system editor ed. On exit from the system editor the temporary file is copied back into the associative memory. Each argument is operated on separately. The sequence of commands to add the string from 'file' to memory with name 'newname' is as follows:

```
e newname
0          (printed by ed)
r file
200
w
200
q          (get out of ed)
q          (get out of fe)
```

To dump a string onto a file:

```
e name
200        (printed by ed)
w filename
200
q          (get out of ed)
q          (get out of fe)
```

d [$name_1$ ...]

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 returns to fed's command level. Any other response does nothing.

m $name_1$ $name_2$ ...

(move) changes the name of $name_1$ to $name_2$ and removes previous string $name_2$ if one exists. Several pairs of arguments may be given.

n [$name_1$ ...]

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

p $name_1$ ...

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

q (quit) 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)

DIAGNOSTICS

'?' unknown command
'Cannot open temp. file'-- cannot create a temporary file for ed command
'name not in memory.' if string 'name' is not in the associative memory and is used as an argument for d or m.

BUGS

--

WARNING

It is legal but an unwise idea to have string names with blanks, ":" or "?" in them.

NAME

form — form letter generator

SYNOPSIS

form proto arg_1 ...

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 nth argument arg_n 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

users 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 allocater, 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), type(I), roff(I)

DIAGNOSTICS

"cannot open output file" "cannot open memory file" when the appropriate files cannot be located or created.

BUGS

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

SYNOPSIS

formL [name] ...

DESCRIPTION

A streamlined program for typing form letters. The names pick out prestored form letters prepared according to the conventions of form and roff. The program prompts to get each blank filled in. When all the forms are completed, it prompts "Set paper." It waits for a newline before printing each letter.

If more than one name is given, the name of each letter is announced before the prompts for it begin. If no names are given, the program asks "Which letter?" before each. Respond with the name and a newline, or newline only when done.

On a 2741 type terminal, the program assumes the letter is to be typed with a correspondence ball, and also prompts "Change ball." Replace the ball at the end.

FILES

form.m (memory),
forma, formb, ... temporaries

SEE ALSO

form(I), fed(I), roff(I)

DIAGNOSTICS

"Try again"--can't get a process

BUGS

--

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 only serves to place a label.

FILES --

SEE ALSO sh(I), :(I)

DIAGNOSTICS "goto error", if the input file is a typewriter; "label not found".

BUGS --

NAME — hyphen -- find hyphenated words

SYNOPSIS — hyphen file$_1$ ...

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.

FILES — --

SEE ALSO — --

DIAGNOSTICS — yes

BUGS — yes, it gets confused, but with no ill effects other than spurious extra output.

NAME if -- conditional command

SYNOPSIS if expr command [arg_1 ...]

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.

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.

FILES --

SEE ALSO sh(I)

DIAGNOSTICS "if error", if the expression has the wrong syntax; "command not found."

BUGS --

NAME ld -- link editor

SYNOPSIS ld [-sulxr] name$_1$ ...

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 ld combines them, producing an object module which can be either executed or become the input for a further ld run. In the latter case, the "-r" option must be given to preserve the relocation bits.

The 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. 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 "-":

-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 "/usr/lib/liba.a", which is the standard system library for assembly language programs. "-lx" stands for "/usr/lib/libx.a" where x is any character. There are libraries for Fortran (x="f"), C (x="c"), Explor (x="e") and B (x="b").

-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.

The output of ld is left on a.out. This file is executable only if no errors occurred during the load.

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

SEE ALSO as(I), ar(I)

DIAGNOSTICS "file not found"-- bad argument

"bad format"-- bad argument

"relocation error"-- bad argument (relocation bits corrupted)

"multiply defined"-- same symbol defined twice in same load

"un"-- stands for "undefined symbol"

"symbol not found"-- loader bug

"can't move output file"-- can't move temporary to a.out file

"no relocation bits"-- an input file lacks relocation information

"too many symbols"-- too many references to external symbols in a given routine

"premature EOF"

"can't create l.out"-- cannot make temporary file

"multiple entry point"-- more than one entry point specified (not possible yet).

BUGS --

NAME ln -- make a link

SYNOPSIS ln $name_1$ [$name_2$]

DESCRIPTION ln creates a link to an existing file $name_1$. If $name_2$ is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of $name_1$.

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

FILES --

SEE ALSO rm(I)

DIAGNOSTICS "?"

BUGS There is nothing particularly wrong with ln, but links don't work right with respect to the backup system: one copy is backed up for each link, and (more serious) in case of a file system reload both copies are restored and the information that a link was involved is lost.

NAME

login -- sign onto UNIX

SYNOPSIS

login [username [password]]

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" (p. vi) 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
/tmp/wtmp	accounting
mailbox	mail
/etc/motd	message-of-the-day
/etc/passwd	password file

SEE ALSO

init(VII), getty(VII), 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 councilor.

BUGS

--

NAME ls -- list contents of directory

SYNOPSIS ls [-ltasd] $name_1$ ...

DESCRIPTION ls lists the contents of one or more directories under control of several options:

-l list in long format, giving i-number, mode, owner, size in bytes, and time of last modification for each file. (see stat for format of the mode)

-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)

If no argument is given, "." is listed. If an argument is not a directory, its name is given.

FILES /etc/passwd to get user ID's for ls -l

SEE ALSO stat(I)

DIAGNOSTICS "name nonexistent"; "name unreadable"; "name unstatable."

BUGS --

NAME

m6 -- general purpose macro processor

SYNOPSIS

m6 [-d arg1] [arg2 [arg3]]

DESCRIPTION

m6 takes input from file arg2 (or standard input if arg2 is missing) and places output on file arg3 (or standard output). A working file of definitions, "m.def", is initialized from file arg1 if that is supplied. M6 differs from the standard [1] in these respects:

#trace:, #source: and #end: are not defined.

#meta,arg1,arg2: transfers the role of metacharacter arg1 to character arg2. If two metacharacters become identical thereby, the outcome of further processing is not guaranteed. For example, to make []{} play the roles of #:<> type

```
\#meta,<\#>,[:
[meta,<:>,]:
[meta,[substr,<<>>,1,1;,{]
[meta,[substr,{{>>,2,1;,}]
```

#del,arg1: deletes the definition of macro arg1.

#save: and #rest: save and restore the definition table together with the current metacharacters on file m.def.

#def,arg1,arg2,arg3: works as in the standard with the extension that an integer may be supplied to arg3 to cause the new macro to perform the action of a specified builtin before its replacement text is evaluated. Thus all builtins except #def: can be retrieved even after deletion. Codes for arg3 are:

```
0 - no function
1,2,3,4,5,6 - gt,eq,ge,lt,ne,le
7,8 - seq,sne
9,10,11,12,13 - add,sub,mpy,div,exp
20 - if
21,22 - def,copy
23 - meta
24 - size
25 - substr
26,27 - go,gobk
28 - del
29 - dnl
30,31 - save,rest
```

FILES

m.def--working file of definitions
/usr/lang/mdir/m6a--m6 processor proper
(/usr/bin/m6 is only an initializer)
/usr/lang/mdir/m6b--default initialization for

m.def
/bin/cp--used for copying initial value of m.def

SEE ALSO

[1] A. D. Hall, The M6 Macroprocessor, Bell Telephone Laboratories, 1969

DIAGNOSTICS

"err" -- a bug, an unknown builtin or a bad definition table
"oprd"--can't open input or initial definitions
"opwr"--can't open output
"ova" -- overflow of nested arguments
"ovc" -- overflow of calls
"ovd" -- overflow of definitions
"Try again" -- no process available for copying m.def

BUGS

Characters in internal tables are stored one per word. They really should be packed to improve capacity. For want of space (and because of unpacked formats) no file arguments have been provided to #save: or #rest:, and no check is made on the actual opening of file m.def. Again to save space, garbage collection makes calls on #save: and #rest: and so overwrites m.def.

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 -yn 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 appended 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 to locate persons
mailbox	input mail
mbox	saved mail

SEE ALSO

login(I)

DIAGNOSTICS

"Who are you?" if the user cannot be identified for some reason (a bug). "Cannot send to user" if mailbox cannot be opened.

BUGS

--

NAME man -- run off section of UNIX manual

SYNOPSIS man title [section]

DESCRIPTION man is a shell command file that will locate and run off a particular section of this manual. Title is the the desired part of the manual. Section is the section number of the manual. (In Arabic, not Roman numerals.) If section is missing, 1 is assumed. For example,

```
man man
```

would reproduce this page.

FILES /sys/man/man?/*

SEE ALSO sh(I), roff(I)

DIAGNOSTICS "File not found", "Usage .."

BUGS --

NAME mesg -- permit or deny messages

SYNOPSIS mesg [n][y]

DESCRIPTION mesg n forbids messages via write by revoking non-user write permission on the user's typewriter. mesg y reinstates permission. mesg with no argument 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 17.

The standard entries "." and ".." are made automatically.

FILES --

SEE ALSO rmdir(I)

DIAGNOSTICS "dirname ?"

BUGS --

NAME mt — manipulate magtape

SYNOPSIS mt [key] [name ...]

DESCRIPTION mt saves and restores selected portions of the file system hierarchy on magtape. 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 tabled.

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. The old contents of the tape are lost.

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 are tabled.

l is the same as t except that an expanded listing is produced giving all the available information about the listed files.

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

0, ..., 7 This modifier selects the drive on which the tape is mounted. "0" is the default.

v Normally mt does its work silently. The v (verbose) option causes it to type the name of each file it treats preceded by a letter to indicate what is happening.

a file is being added
x file is being extracted

The v option can be used with r and x only.

f causes new entries copied on tape to be

'fake' in that only the entries, not the data associated with the entries are updated. Such fake entries cannot be extracted. Usable only with r.

w causes mt 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 mt command terminates immediately. In the x function, files previously asked about have been extracted already. With r, no change has been made to the tape.

m make (create) directories during an x if necessary.

FILES /dev/mt?

SEE ALSO tap(I), tap(V)

DIAGNOSTICS
Tape open error
Tape read error
Tape write error
Directory checksum
Directory overflow
Seek error
Tape overflow
Phase error (a file has changed after it was selected for dumping but before it was dumped)

BUGS If, during an "x", the files are specified in a different order than they are on the tape, seek errors will result because the tape cannot be rewound.

NAME — mv -- move or rename a file

SYNOPSIS — mv $name_1$ $name_2$

DESCRIPTION — mv changes the name of $name_1$ to $name_2$. If $name_2$ is a directory, $name_1$ is moved to that directory with its original file-name. Directories may only be moved within the same parent directory (just renamed).

FILES — --

SEE ALSO — --

DIAGNOSTICS — yes

BUGS — --

NAME nm -- print name list

SYNOPSIS nm [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), or "B" (bss segment symbol). Global symbols have their first character underlined. The output is sorted alphabetically.

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

FILES a.out

SEE ALSO as(I), ld(I)

DIAGNOSTICS "?"

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.
II. Text Filling, Adjusting, and Centering				
.br	-	-	yes	BReak.
.fi	fill	-	yes	FIll output lines.
.nf	fill	-	yes	NoFill.
.ad c	adj,norm	adjust	no	ADjust mode on.
.na	adjust	-	no	NoAdjust.
.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	OR-
.sv N	-	N=1	no	SaVe N lines.
.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.
.rm xx	-	-	no	ReMove macro name.
.di xx	-	end	no	DIvert output to macro "xx".
.wh -N xx		-	no	WHen; set a line trap.
.ch -N -M		-	no	OR-
.ch xx -M		-	no	OR-
.ch -N y		-	no	OR-
.ch xx y		-	no	CHange trap line.
VI. Number Registers				
.nr a ±N -M		-	no	OR-
.nr ab ±N -M		-	no	Number Register.
.nc c	\n	\n	no	Number Character.
.ar	arabic	-	no	Arabic numbers.

.ro	arabic	-	no	Roman numbers.
.RO	arabic	-	no	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.
.li N	-	N=1	no	Accept input lines LIterally.
.tr abcd....		-	no	TRanslate on output.

VIII. Hyphenation.

.nh	on	-	no	No Hyphen.
.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 c anything		-	no	OR-
.if !c anything		-	no	OR-
.if N anything		-	no	OR-
.if !N anything		-	no	IF true accept line of "anything".

XII. Environment Switching.

.ev N	N=0	N=prev	no	EnVironment switched.

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.

.sp

XV. Miscellaneous

.ig	-	-	no	IGnore.
.fl	-	-	no	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 an end-of-file condition or until halted by sending an interrupt signal.

FILES --

SEE ALSO db(I)

DIAGNOSTICS --

BUGS --

NAME opr -- off line print

SYNOPSIS opr [--] [-] [+] [+-]file$_1$ ...

DESCRIPTION opr will arrange 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 --, the output is remoted to station R1. (Station R1 has a 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(I), passwd(V)

DIAGNOSTICS --

BUGS --

NAME ov -- overlay pages

SYNOPSIS ov [file]

DESCRIPTION ov is a postprocessor for producing double column formatted text when using nroff(I). ov literally overlays successive pairs of 66-line pages.

If the file argument is missing, the standard input is used. Thus ov may be used as a filter.

FILES none

SEE ALSO nroff(I), pr(I)

DIAGNOSTICS none

BUGS Other page lengths should be permitted.

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 user ID 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)

DIAGNOSTICS Diagnostics are given for a non-match of the login name, lack of permission and for password file format errors.

BUGS --

NAME

pr -- print file

SYNOPSIS

pr [-cm] [-h name] [-n] [+n] [file$_1$ ...]

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 the standard input file, and is thus usable as a filter.

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

-c print current date
-m print date file last modified (default)

-n produce n-column output

+n begin printing with page n

-h treats the next argument as a header

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

In multi-column output, non-printing characters other than new-line cause misalignment.

NAME proof -- compare two text files

SYNOPSIS proof oldfile newfile

DESCRIPTION proof lists those lines of newfile that differ from corresponding lines in oldfile. The line number in newfile is given. When changes, insertions or deletions have been made the program attempts to resynchronize the text in the two files by finding a sequence of lines in both files that again agree.

FILES --

SEE ALSO cmp(I)

DIAGNOSTICS yes, but they are undecipherable, e.g. "?1".

BUGS proof is still evolving. Any bugs discovered or suggestions should be brought to ENP.

NAME reloc -- relocate object files

SYNOPSIS reloc file [-]octal [-]

DESCRIPTION reloc modifies the named object program file so that it will operate correctly at a different core origin than the one for which it was assembled or loaded.

The new core origin is the old origin increased by the given octal number (or decreased if the number has a "-" sign).

If the object file was generated by the link-editor ld, the "-r" ld option must have been given to preserve the relocation information in the file.

If the optional last argument is given, then any "setd" instruction at the start of the file will be replaced by a no-op.

The purpose of this command is to simplify the preparation of object programs for systems which have no relocation hardware. It is hard to imagine a situation in which it would be useful to attempt directly to execute a program treated by reloc.

FILES --

SEE ALSO as(I), ld(I), a.out(V)

DIAGNOSTICS As appropriate

BUGS --

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 — /dev/tap?
/dev/mt?

SEE ALSO — --

DIAGNOSTICS — "?" if there is no tape mounted on the indicated drive or if the file cannot be opened.

BUGS — --

NAME

rm -- remove (unlink) files

SYNOPSIS

rm [-f] [-r] $name_1$ ...

DESCRIPTION

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

If 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 the above 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 -r flag

SEE ALSO

rmdir(I)

DIAGNOSTICS

"name: non existent"
"name: not removed" if cannot remove
"name: try again" error from fork

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_1 ...

DESCRIPTION rmdir removes (deletes) directories. The directory must be empty (except for the standard entries "." and "..", which rmdir itself removes). Write permission is required in the directory in which the directory appears.

FILES none

SEE ALSO --

DIAGNOSTICS "dir?" is printed if directory dir cannot be found, is not a directory, or is not removable.

"dir -- directory not empty" is printed if dir has entries other than "." or "..".

BUGS --

NAME roff -- format text

SYNOPSIS roff [±number] [-s] [-h] file$_1$ ...

DESCRIPTION roff formats text according to control lines embedded in the text in files name$_1$, ... Encountering a nonexistent file terminates printing. The optional argument "±number" causes printing to begin at the first page with that number. The optional argument "-s" causes printing to stop before each page including the first to allow paper manipulation; printing is resumed upon receipt of an interrupt signal. The optional argument "-h" causes the output to contain horizontal tabs for two or more spaces that end on a tab stop. An interrupt signal received during printing terminates all printing. Incoming interconsole messages are turned off during printing, and the original message acceptance state is restored upon termination.

At the present time, there is no document describing ROFF in full. A Request Summary is attached.

FILES

/etc/suftab	suffix hyphenation tables
/tmp/rtm?	temporary

SEE ALSO --

DIAGNOSTICS none

BUGS -

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 contiguous block of n blank lines. If necessary, a new page will be started to accomodate the entire block.
.bp +n	yes	n=1	Begin new page and number it n. If n is not given, normal sequencing occurs.
.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 with a line beginning with "..").
.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 until and including 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	n blank lines are put between the top of a new page and the head title.
.m2 n	no	n=2	n blanks lines put between head title and beginning of text on page.
.m3 n	no	n=1	n blank lines put between the end of text and the 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	Output lines are numbered sequentially beginning with 1 on each new page. Head and foot titles are not numbered.
.n2	no	no	Output lines are numbered sequentially beginning with 1 on the next output line.
.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	-	n pages with head and foot titles but otherwise blank will be output beginning with the next page containing text.
.sp n	yes	-	Insert block of n blank lines. If the bottom of a page is reached, remaining lines are not put on next page.
.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 spaces.
.tr abcd..	no	-	Translate a into b, c into d, etc.
.ul n	no	-	Underline the letters and numbers on the next n input lines.

NAME sh -- shell (command interpreter)

SYNOPSIS sh [name [arg_1 ... [arg_9]]]

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.

Command lines

Command lines are sequences of commands separated by command delimiters. 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 "overflow" 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.

The remaining non-special arguments are simply passed to the command without further interpretation by the shell.

Command delimiters

There are three command delimiters: the new-line, ";", and "&". The semicolon ";" specifies sequential execution of the commands so separated; that is,

```
coma; comb
```

causes the execution first of command coma, then of comb. The ampersand "&" causes simultaneous execution:

```
coma & comb
```

causes coma to be called, followed immediately by comb without waiting for coma to finish. Thus coma and comb execute simultaneously. As a special case,

```
coma &
```

causes coma to be executed and the shell immediately to request another command without waiting for coma.

Termination Reporting

If a command (not followed by "&") terminates abnormally, a message is printed. (All terminations other than exit and interrupt are considered abnormal.) The following is a list of the abnormal termination messages:

```
Bus error
Trace/BPT trap
Illegal instruction
IOT trap
Power fail trap
EMT trap
Bad system call
Quit
PIR trap
Floating exception
Memory violation
Killed
User I/O
Error
```

If a core image is produced, " -- Core dumped" is appended to the appropriate message.

Redirection of I/O

Three character sequences cause the immediately following string to be interpreted as a special argument to the shell itself, not passed to the command.

An argument of the form "<arg" causes the file arg to be used as the standard input file of the given command.

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

Pipes and Filters

A pipe is a channel such that information can be written into one end of the pipe by one program, and read at the other end by another program. (See pipe (II)). A filter is a program which reads the standard input file, performs some transformation, and writes the result on the standard output file. By extending the syntax used for redirection of I/O, a command line can specify that the

output produced by a command be passed via a pipe through another command which acts as a filter. For example:

```
command >filter>
```

More generally, special arguments of the form

$>f_1>f_2>...>$

specify that output is to be passed successively through the filters f_1, f_2, ..., and end up on the standard output stream. By saying instead

$>f_1>f_2>...>$file

the output finally ends up in file. (The last ">" could also have been a ">>" to specify concatenation onto the end of file.)

In exactly analogous manner input filtering can be specified via one of

$<f_1<f_2<...<$
$<f_1<f_2<...<$file

Both input and output filtering can be specified in the same command, though not in the same special argument.

For example:

```
ls >pr>
```

produces a listing of the current directory with page headings, while

```
ls >pr>xx
```

puts the paginated listing into the file xx.

If any of the filters needs arguments, quotes can be used to prevent the required blank characters from violating the blankless syntax of filters. For example:

```
ls >"pr -h 'My directory'">
```

uses quotes twice, once to protect the entire pr command, once to protect the heading argument of pr. (Quotes are discussed fully below.)

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.

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 [arg_1 ... [arg_9]]]

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 0, ..., 9, are replaced by the nth argument to the invocation of the shell (arg_n). "$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 end of file.

Special commands

Two commands are treated specially by the shell.

"Chdir" is done without spawning a new process by executing the sys chdir primitive.

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

These peculiarities are inexorably imposed upon the shell by the basic structure of the UNIX process control system. It is a rewarding exercise to work out why.

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.

FILES /etc/glob, which interprets "*", "?", and "[".

SEE ALSO "The UNIX Time-sharing System", which gives the theory of operation of the shell.

DIAGNOSTICS
"Input not found", when a command file is specified which

cannot be read;
"Arg count", if the number of arguments to the chdir pseudo-command is not exactly 1, or if "*", "?", or "[" is used inappropriately;
"Bad directory", if the directory given in "chdir" cannot be switched to;
"Try again", if no new process can be created to execute the specified command;
"" imbalance", if single or double quotes are not matched;
"Input file", if an argument after "<" cannot be read;
"Output file", if an argument after ">" or ">>" cannot be written (or created);
"Command not found", if the specified command cannot be executed.
"No match", if no arguments are generated for a command which contains "*", "?", or "[".
Termination messages described above.

BUGS If any argument contains a quoted "*", "?", or "[", then all instances of these characters must be quoted. This is because sh calls the glob routine whenever an unquoted "*", "?", or "[" is noticed; the fact that other instances of these characters occurred quoted is not noticed by glob.

When output is redirected, particularly through a filter, diagnostics tend to be sent down the pipe and are sometimes lost altogether.

NAME size -- size of an object file

SYNOPSIS size [object ...]

DESCRIPTION The size, in bytes, of the object files are printed. If no file is given, a.out is default. The size is printed in octal for the text, data, and bss portions of each file. The sum of these is also printed in octal and decimal.

FILES a.out default

SEE ALSO --

DIAGNOSTICS "object not found" if the input cannot be read. "bad format: object" if the input file does not have a valid object header.

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'.

The following is a list of differences between sno and SNOBOL III:

There are no unanchored searches. To get the same effect:

```
a ** b          unanchored search for b
a *x* b = x c   unanchored assignment
```

No back referencing

```
x = "abc"
a *x* x         unanchored search for "abc"
```

Different function declaration. 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.

```
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.

Variable length patterns at the end of a pattern match are not treated specially. They still match the shortest rather than longest text.

Parentheses for arithmetic are not needed. Normal (eg FORTRAN) 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.

FILES --

SEE ALSO SNOBOL III manual. (JACM; Vol. 11 No. 1; Jan 1964; pp 21)

DIAGNOSTICS As appropriate

BUGS Runtime diagnostics give the last program line number rather than the executing statement line number.

NAME — sort — sort a file

SYNOPSIS — sort [-] [input [output]]

DESCRIPTION — sort will sort the input file and write the sorted file on the output file. If the output file is not given, the input file is rewritten. If the input file is missing, sort uses the standard input as input and the standard output for output. Thus sort may be used as a filter.

The sort is line-by-line in increasing ASCII collating sequence, except that upper-case letters are considered the same as the lower-case letters.

The optional argument - will cause a reverse sort.

sort is implemented in such a way that

```
sort /dev/mt0
```

works correctly provided the tape is not too big.

FILES — /tmp/stm?

SEE ALSO — --

DIAGNOSTICS — --

BUGS — The largest file that can be sorted is about 128K bytes.

NAME split -- split a file into pieces

SYNOPSIS split [[file1] file2]

DESCRIPTION Split reads file1 and writes it in 1000-line pieces, as many as are necessary, onto a set of output files. The name of the first output file is file2 with an "a" appended, and so on through the alphabet and beyond. If no output name is given, "x" is default.

If no input file is given, or the first argument is "-", then the standard input file is used.

FILES -

SEE ALSO --

DIAGNOSTICS yes

BUGS Watch out for 8-character file names.

NAME

speak -- word to voice translator

SYNOPSIS

speak [-] [vocabulary]

DESCRIPTION

speak turns a stream of ascii words into utterances and outputs them to a voice synthesizer. 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 phonetic code is given in vsp(VII).
- empty lines
- command lines - beginning with !. The following forms 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
!s file	(save) append working word and phonetics to file in style of !l

Each working line replaces its predecessor. Its first word is the "working word". Each phonetic line replaces the phonetics stored 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 spelled as strings of one-letter words. Unknown one-letter words burp.

A phonetic line of comma only will delete the entry for the working word.

speak is initialized with a coded vocabulary stored in file /etc/speak.m. The vocabulary option substitutes a different file for speak.m.

The - option suppresses all utterances.

FILES

/etc/speak.m

SEE ALSO

vsp(VII), speakm(V), vt(IV)

BUGS

Vocabulary overflow is unchecked. Excessively long words cause dumps. Space is not reclaimed from deleted entries.

NAME stat -- get file status

SYNOPSIS stat $name_1$ ...

DESCRIPTION stat gives several kinds of information about one or more files:

i-number
access mode
number of links
owner
size in bytes
date and time of last modification
name (useful when several files are named)

All information is self-explanatory except the mode. The mode is a six-character string whose characters mean the following:

1 s: file is small (smaller than 4096 bytes)
l: file is large

2 d: file is a directory
x: file is executable
u: set user ID on execution
-: none of the above

3 r: owner can read
-: owner cannot read

4 w: owner can write
-: owner cannot write

5 r: non-owner can read
-: non-owner cannot read

6 w: non-owner can write
-: non-owner cannot write

The owner is almost always given in symbolic form; however if he cannot be found in "/etc/passwd" a number is given.

If the number of arguments to stat is not exactly 1 a header is generated identifying the fields of the status information.

FILES /etc/passwd

SEE ALSO istat(I), ls(I) (-l option)

DIAGNOSTICS "name?" for any error.

NAME — strip -- remove symbols and relocation bits

SYNOPSIS — strip $name_1$ ...

DESCRIPTION — strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

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

FILES — /tmp/stm? temporary file

SEE ALSO — ld(I), as(I)

DIAGNOSTICS — Diagnostics are given for: non-existent argument; inability to create temporary file; improper format (not an object file); inability to re-read temporary file.

BUGS — --

NAME stty -- set teletype options

SYNOPSIS stty option$_1$ ...

DESCRIPTION Stty will set certain I/O options on the current output teletype. 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)
-raw	negate raw mode
-nl	allow cr for lf (and echo lf cr)
nl	allow nl only
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
tabs	preserve tabs
delay	calculate cr and tab delays.
-delay	no cr/tab delays
ebcdic	ebcdic ball conversion (2741 only)
corres	correspondence ball conversion (2741 only)

FILES standard output.

SEE ALSO stty(II)

DIAGNOSTICS "Bad options"

BUGS --

NAME sum -- sum file

SYNOPSIS sum $name_1$ ...

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.

FILES none

SEE ALSO --

DIAGNOSTICS "oprd" if the file cannot be opened; "?" if an error is discovered during the read.

BUGS none

NAME tap -- manipulate DECtape

SYNOPSIS tap [key] [name ...]

DESCRIPTION tap saves and restores selected portions of the file system hierarchy on DECtape. 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 tabled.

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 (hence the "r"). "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.

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 are tabled.

l is the same as t except that an expanded listing is produced giving all the available information about the listed files.

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

0, ..., 7 This modifier selects the drive on which the tape is mounted. "0" is the default.

v Normally tap does its work silently. The v (verbose) option causes it to type the name of each file it treats preceded by a letter to indicate what is happening.

r file is being replaced
a file is being added (not there before)
x file is being extracted
d file is being deleted

The v option can be used with r, u, d, and x only.

c means a fresh dump is being created; the tape directory will be zeroed before beginning. Usable only with r and u.

f causes new entries copied 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.

w causes tap 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 tap 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.

m make (create) directories during an x if necessary.

FILES /dev/tap?

SEE ALSO mt(I)

DIAGNOSTICS
Tape open error
Tape read error
Tape write error
Directory checksum
Directory overflow
Tape overflow
Phase error (a file has changed after it was selected for dumping but before it was dumped)

BUGS Asks about "fake" entries on "xw", when it should

ignore them. If a fake entry is extracted, and the file already exists on disk, the extraction does not take place (as is correct), but the mode and user ID of the file are set to 0.

NAME time -- time a command

SYNOPSIS time command

DESCRIPTION The given command is timed; after it is complete, time prints the time spent in the system, waiting for disk, and in execution of the command.

The disk I/O time can be variable depending on other activity in the system.

FILES --

SEE ALSO tm (VIII)

DIAGNOSTICS "?"
"command terminated abnormally"
"Command not found."

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/tmg1.o -- the compiler-compiler
/sys/tmg[abc] -- libraries
alloc.d -- table storage

SEE ALSO A Manual for the Tmg Compiler-writing Language, MM-72-1271-8.

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.

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 tss -- interface to Honeywell TSS

SYNOPSIS tss

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

An interrupt signal (ASCII DEL) is transmitted as a "break" to 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. Underlined text comes from TSS. AFTname is the 6070 file to be dealt with.

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

SEE ALSO —

DIAGNOSTICS DONE when communication is broken.

BUGS When diagnostic problems occur, tss exits rather abruptly.

NAME tty -- get tty name

SYNOPSIS tty

DESCRIPTION tty gives the name of the user's typewriter in the form "ttyn" for n a digit. The actual path name is then "/dev/ttyn".

FILES --

SEE ALSO --

DIAGNOSTICS "not a tty" if the standard input file is not a typewriter.

BUGS --

NAME type -- type on single sheet paper

SYNOPSIS type $file_1$ ...

DESCRIPTION type copies its input files to the standard output. Before each new page (66 lines) and before each new file, type stops and reads the standard input for a new line character before continuing. This allows time for insertion of single sheet paper.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME typo -- find possible typo's

SYNOPSIS typo [-] file$_1$ ...

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

All 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 "-" option suppresses the help from English and should be used if the document is written in, for example, Urdu.

Roff and Nroff 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??, /etc/salt, /etc/w2006

SEE ALSO --

DIAGNOSTICS yes, lots

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

NAME un -- undefined symbols

SYNOPSIS un [name]

DESCRIPTION un prints a list of undefined symbols from an assembly or loader run. If the file argument is not specified, a.out is the default. Names are listed alphabetically except that non-global symbols come first. Undefined global symbols (unresolved external references) have their first character underlined.

FILES a.out

SEE ALSO as(I), ld(I)

DIAGNOSTICS "?" if the file cannot be found.

BUGS --

NAME uniq — report repeated lines in a file

SYNOPSIS uniq [-ud] [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. Note that the normal mode output is the union of the -u and -d mode outputs.

The following example will print one copy of all lines in the file a that do not occur in b:

```
sort a x
uniq x a1
sort b x
uniq x b1
cat a1 b1 >x
sort x
uniq -u x >>a1
sort a1
uniq -d a1
```

FILES —

SEE ALSO sort(I)

DIAGNOSTICS "cannot open input", "cannot create output"

BUGS —

NAME vs -- phoneme list to voice synthesizer

SYNOPSIS vs

DESCRIPTION vs accepts phoneme descriptor lists and translates them into byte strings suitable for the Federal Screw Works Voice Synthesizer. Phoneme descritors should be separated by commas and have the general form "%NIxx" where "xx" is a one or two character phoneme name, "I" is an optional inflection parameter, and "%N" is an optional count of the number of times the phoneme is to be repeated (maximum 9). "I" can have the values 0, 1, 2, 3 representing decreasing strength (default is 2). A description of the phonemes and their names can be found in the file vsp(VII). For example,

```
a0,o1,t,r,1ai,1ay,d,j,ih,u1,%2s
```

will generate the word "outrageous". The output is buffered; a newline will cause the buffered output to be sent to the Voice Synthesizer.

FILES -

SEE ALSO vsp(VII), speak(I)

DIAGONOSTICS -

BUGS -

NAME wc -- get (English) word count

SYNOPSIS wc name$_1$ ...

DESCRIPTION wc provides a count of the words, text lines, and control lines for each argument file.

A text line is a sequence of characters not beginning with ".", "!" or "'" and ended by a newline. A control line is a line beginning with "." "!" or "'". A word is a sequence of characters bounded by the beginning of a line, by the end of a line, or by a blank or a tab.

When there is more than one input file, a grand total is also printed.

FILES --

SEE ALSO roff(I)

DIAGNOSTICS none; arguments not found are ignored.

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, last character of input device name (with /dev/tty suppressed), date and time. Certain 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(VII)

DIAGNOSTICS "?" if a named file cannot be read.

BUGS --

NAME write -- write to another user

SYNOPSIS write user

DESCRIPTION write copies lines from your typewriter to that of another user. When first called, write 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.

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), msh(VII)

DIAGNOSTICS "user not logged in"; "permission denied".

BUGS write should check the mode of the other user's typewriter and refuse to proceed unless non-user write permission is given. Currently it is possible to write to another person with the same user-ID even though he has forbidden messages.

write should also allow specification of the typewriter name of a user who is logged in several times instead of picking out the instance with the lowest name.

NAME boot -- reboot UNIX

SYNOPSIS sys boot / boot = 39. not in assembler

DESCRIPTION UNIX will clean up outstanding I/O, and then execute the reboot read-only program. This call is restricted to the super-user. All users will be logged out.

SEE ALSO boot procedures (VII)

DIAGNOSTICS the c-bit is set if you are not the super-user

BUGS It often doesn't work (for unknown reasons).
It depends on switch settings.

NAME break -- set program break

SYNOPSIS sys break; addr / break = 17.

DESCRIPTION break sets the system's idea of the highest location used by the program to addr. Locations greater than addr and below the stack pointer are not swapped and are thus liable to unexpected modification.

An argument of 0 is taken to mean 16K bytes. If the argument is higher than the stack pointer the entire user core area is swapped.

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)

DIAGNOSTICS none; strange addresses cause the break to be set at 16K bytes.

BUGS --

NAME cemt -- catch emt traps

SYNOPSIS sys cemt; arg / cemt = 29.

DESCRIPTION This call allows one to catch traps resulting from the emt instruction. Arg is a location within the program; emt traps are sent to that location. The normal effect of emt traps may be restored by giving an arg equal to 0.

To return after catching the emt trap, execute the rti instruction.

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME chdir -- change working directory

SYNOPSIS
```
sys chdir; dirname / chdir = 12.
```

DESCRIPTION dirname is the address of the pathname of a directory, terminated by a 0 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.

BUGS --

NAME chmod -- change mode of file

SYNOPSIS

```
sys chmod; name; mode / chmod = 15.
```

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:

```
01 write, non-owner
02 read, non-owner
04 write, owner
10 read, owner
20 executable
40 set user ID on execution
```

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

SEE ALSO chmod(I)

DIAGNOSTICS 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.

BUGS --

NAME chown -- change owner of file

SYNOPSIS sys chown; name; owner / chown = 16.

DESCRIPTION The file whose name is given by the null-terminated string pointed to by name has its owner changed to owner. Only the present owner of a file (or the super-user) may donate the file to another user. Also, one may not change the owner of a file with the set-user-ID bit on, otherwise one could create Trojan Horses.

SEE ALSO chown(I), uids(V)

DIAGNOSTICS The error bit (c-bit) is set on illegal owner changes.

BUGS --

NAME close -- close a file

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

DESCRIPTION Given a file descriptor such as returned from an open or creat call, close closes the associated file. A close of all files is automatic on exit, but since processes are limited to 10 simultaneously open files, close is necessary for programs which deal with many files.

SEE ALSO creat(II), open(II)

DIAGNOSTICS The error bit (c-bit) is set for an unknown file descriptor.

BUGS --

NAME creat -- create a new file

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

DESCRIPTION creat creates a new file or prepares to rewrite an existing file called name; name is the address of a null-terminated string. If the file did not exist, it is given mode mode; if it 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)

DIAGNOSTICS The error bit (c-bit) may be set if: a needed directory is not readable; 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 10 files open.

BUGS —

NAME csw -- read console switches

SYNOPSIS
```
sys csw / csw = 38. not in assembler
(value of csw in r0)
(value of buttons in r1)
```

DESCRIPTION The setting of the console switches is returned in r0. The setting of the external buttons is returned in r1. The return is synced to a 30 CPS clock for graphical applications.

SEE ALSO --

DIAGNOSTICS none

BUGS Currently the buttons are unavailable.

NAME dup -- duplicate an open file descriptor

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

DESCRIPTION Given a file descriptor returned from an open 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 manipulate the value of file descriptors than to genuinely duplicate a file descriptor. Since the algorithm to allocate file descriptors is known to use the lowest available value between 0 and 9, 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)

DIAGNOSTICS The error bit (c-bit) is set if: the given file descriptor is invalid; there are already 10 open files.

BUGS --

NAME exec -- execute a file

SYNOPSIS

```
sys exec; name; args  / exec = 11.
...
name: <...\0>
...
args: arg1; arg2; ...; 0
arg1: <...\0>
...
```

DESCRIPTION

exec overlays the calling process with the named file, then transfers to the beginning of the core image of the file. The first argument to exec is a pointer to the name of the file to be executed. The second is the address of a 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.

There can be no return from the file; the calling core image is lost.

The program break is set from the executed file; see the format of a.out.

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.

```
 sp->  nargs
       arg1
       ...
       argn

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

The arguments are placed as high as possible in core: just below 57000(8).

Files remain open across exec calls. However, the illegal instruction, emt, quit, and interrupt trap specifications are reset to the standard values. (See ilgins, cemt, quit, intr.)

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

SEE ALSO fork(II)

DIAGNOSTICS If the file cannot be read or if it is not executable, a return from exec constitutes the diagnostic. The error bit (c-bit) is set.

BUGS Very high core and very low core are used by exec to construct the argument list for the new core image. If the original copies of the arguments reside in these places, problems can result.

NAME exit -- terminate process

SYNOPSIS
```
(status in r0)
sys exit  / exit = 1
```

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

This call can never return.

SEE ALSO wait(II)

DIAGNOSTICS -

BUGS --

NAME fork -- spawn new process

SYNOPSIS
```
sys fork / fork = 2.
(new process return)
(old process return)
```

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 process contains the process ID of the new process. This process ID is used by wait.

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.

BUGS See wait(II) for a subtile bug in process destruction.

NAME — fpe -- set floating exception handling

SYNOPSIS — sys fpe; arg / fpe = 40. not in assembler

DESCRIPTION — This call allows one to catch traps resulting from floating point exceptions. Arg is a location within the program; floating exception traps are sent to that location. The normal effect of floating exception traps may be restored by giving an arg equal to 0.

To return after catching the fpe trap, execute the rti instruction.

SEE ALSO — --

DIAGNOSTICS — --

BUGS — The floating point exception (FEC) register is not saved per process. Examining this register for possible remedial action after a floating point exception trap is not guaranteed to work.

NAME fstat -- get status of open file

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

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.

BUGS --

NAME getuid -- get user identification

SYNOPSIS
```
sys getuid / getuid = 24.
(user ID in r0)
```

DESCRIPTION getuid returns the real 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 each 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 --

BUGS --

NAME gtty -- get typewriter status

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

DESCRIPTION gtty stores in the three words addressed by arg the status of the typewriter whose file descriptor is given in r0. 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.

BUGS --

NAME ilgins -- catch illegal instruction trap

SYNOPSIS

```
sys ilgins; arg  / ilgins = 33.
```

DESCRIPTION

ilgins allows a program to catch illegal instruction traps. If arg is zero, the normal instruction trap handling is done: the process is terminated and a core image is produced. If arg is a location within the program, control is passed to arg when the trap occurs.

This call is used to implement the floating point simulator, which catches and interprets 11/45 floating point instructions.

To return after catching the ilgins trap, execute the rti instruction.

SEE ALSO PDP-11 manual

DIAGNOSTICS --

BUGS --

NAME intr -- set interrupt handling

SYNOPSIS
```
sys intr; arg  / intr = 27.
```

DESCRIPTION When arg is 0, interrupts (ASCII DELETE) are ignored. When arg is 1, interrupts cause their normal result, that is, force an exit. When arg is a location within the program, control is transferred to that location when an interrupt occurs.

After an interrupt is caught, it is possible to resume execution by means of an rti instruction; however, great care must be exercised, since all I/O is terminated abruptly upon an interrupt. In particular, reads of the typewriter tend to return with 0 characters read, thus simulating an end of file.

SEE ALSO quit(II)

DIAGNOSTICS --

BUGS --

NAME kill -- destroy process

SYNOPSIS
```
(process number in r0)
sys kill  / kill = 37.; not in assembler
```

DESCRIPTION kill destroys a process, given its process number. The process leaves a core image.

This call is restricted to the super-user, and is intended only to kill an otherwise unstoppable process.

SEE ALSO --

DIAGNOSTICS c-bit set if user is not the super-user, or if process does not exist.

BUGS Under strange circumstances, kill is ineffective.

NAME link -- link to a file

SYNOPSIS sys link; $name_1$; $name_2$ / link = 9.

DESCRIPTION A link to $name_1$ is created; the link has name $name_2$. Either name may be an arbitrary path name.

SEE ALSO link(I), unlink(II)

DIAGNOSTICS The error bit (c-bit) is set when $name_1$ cannot be found; when $name_2$ already exists; when the directory of $name_2$ 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.

BUGS --

NAME makdir -- make a directory

SYNOPSIS sys makdir; name; mode / makdir = 14.

DESCRIPTION makdir creates an empty directory whose name is the null-terminated string pointed to by name. The mode of the directory is mode. The special entries "." and ".." are not present.

makdir may be invoked only by the super-user.

SEE ALSO mkdir(I)

DIAGNOSTICS Error bit (c-bit) is set if the directory already exists or if the user is not the super-user.

BUGS --

NAME mdate -- set modified date on file

SYNOPSIS
```
(time to r0-r1)
sys mdate; file  / mdate = 30.
```

DESCRIPTION File is the address of a null-terminated string giving the name of a file. The modified time of the file is set to the time given in the r0-r1 registers.

This call is allowed only to the super-user or to the owner of the file.

SEE ALSO --

DIAGNOSTICS Error bit is set if the user is neither the owner nor the super-user or if the file cannot be found.

BUGS --

NAME mount -- mount file system

SYNOPSIS

```
sys  mount; special; name  / mount = 21.
```

DESCRIPTION

mount announces to the system that a removable file system has been mounted on 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. If it had contents, they are inaccessible while the file system is mounted.

SEE ALSO mount(I), umount(II)

DIAGNOSTICS Error bit (c-bit) set if: special is inaccessible; name does not exist; special is already mounted; name is not on the RF; there are already four special files mounted.

BUGS At most four removable devices can be mounted at a time. This call should be restricted to the super-used.

NAME nice -- set program in low priority

SYNOPSIS sys nice / nice = 34.

DESCRIPTION The currently executing process is set into the lowest priority execution queue. Background jobs that execute a very long time should do this. Once done, there is no way to restore a process to normal priority.

SEE ALSO formerly known as "hog"

DIAGNOSTICS --

BUGS --

NAME

open -- open for reading or writing

SYNOPSIS

sys open; name; mode / open = 5.
(descriptor in r0)

DESCRIPTION

open opens the file name for reading (if mode is 0) or writing (if mode is non-zero). name is the address of a string of ASCII characters representing a path name, terminated by a null character.

The file descriptor should be saved for subsequent calls to read (or write) and close.

In both the read and write case the file pointer is set to the beginning of the file.

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 10 files are open.

BUGS

--

NAME

pipe -- create a pipe

SYNOPSIS

```
sys pipe        / pipe = 42.; not in assembler
(file descriptor in r0)
```

DESCRIPTION

The pipe system call creates an I/O mechanism called a pipe. The file descriptor returned can be used in both read and write operations. When the pipe is written, the data is buffered up to 504 bytes at which time the writing process is suspended. A read on the pipe will pick up the buffered 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 (no synonymous file descriptors resulting from fork or dup) 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 10 files are already open.

BUGS

--

NAME quit -- turn off quit signal

SYNOPSIS

```
sys quit; flag  / quit = 26.
```

DESCRIPTION When flag is 0, this call disables quit signals from the typewriter (ASCII FS). When flag is non-zero, quits are re-enabled, and cause execution to cease and a core image to be produced.

Quits should be turned off only with due consideration.

SEE ALSO intr(II)

DIAGNOSTICS --

BUGS --

NAME read -- read from file

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

DESCRIPTION A file descriptor is a word returned from a successful open or creat 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 r0 returns with value 0, then end-of-file has been reached.

SEE ALSO open(II), creat(II)

DIAGNOSTICS As mentioned, r0 is 0 on return 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.

BUGS --

NAME rele -- release processor

SYNOPSIS sys rele / rele = 0; not in assembler

DESCRIPTION This call causes the process to be swapped out immediately if another process wants to run. Its main reason for being is internal to the system, namely to implement timer-runout swaps. However, it can be used beneficially by programs which wish to loop for some reason without consuming more processor time than necessary.

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME seek -- move read/write pointer

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

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.

SEE ALSO --

DIAGNOSTICS The error bit (c-bit) is set for an undefined file descriptor.

BUGS A file can conceptually be as large as 2**20 bytes. Clearly only 2**16 bytes can be addressed by seek. The problem is most acute on the large special files.

NAME setuid -- set process ID

SYNOPSIS
```
(process ID in r0)
sys setuid  / setuid = 23.
```

DESCRIPTION The user ID of the current process is set to the argument in r0. Both the effective and the real user ID are set. This call is only permitted to the super-user or if r0 is the real user ID.

SEE ALSO getuid(II)

DIAGNOSTICS Error bit (c-bit) is set as indicated.

BUGS --

NAME — sleep -- stop execution for interval

SYNOPSIS

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

DESCRIPTION — The current process is suspended from execution for the number of seconds specified by the contents of register 0.

SEE ALSO — --

DIAGNOSTICS — --

BUGS — Due to the implementation, the sleep interval is only accurate to 256 60ths of a second (4.26 sec). Even then, the process is placed on a low priority queue and must be scheduled.

NAME stat -- get file status

SYNOPSIS sys stat; name; buf / stat = 18.

DESCRIPTION name points to a null-terminated string naming a file; buf is the address of a 34(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 format:

buf, +1	i-number
+2,+3	flags (see below)
+4	number of links
+5	user ID of owner
+6,+7	size in bytes
+8,+9	first indirect block or contents block
...	
+22,+23	eighth indirect block or contents block
+24,+25,+26,+27	creation time
+28,+29,+30,+31	modification time
+32,+33	unused

The flags are as follows:

100000	used (always on)
040000	directory
020000	file has been modified (always on)
010000	large file
000040	set user ID
000020	executable
000010	read, owner
000004	write, owner
000002	read, non-owner
000001	write, non-owner

SEE ALSO stat(I), fstat(II)

DIAGNOSTICS Error bit (c-bit) is set if the file cannot be found.

BUGS --

NAME stime -- set time

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

DESCRIPTION stime sets the system's idea of the time and date. Only the super-user may use this call.

SEE ALSO date(I), time(II)

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

BUGS --

NAME stty -- set mode of typewriter

SYNOPSIS
```
(file descriptor in r0)
sys stty; arg  / stty = 31.
...
arg: dcrsr; dctsr; mode
```

DESCRIPTION

stty sets mode bits for a typewriter whose file descriptor is passed in r0. First, the system delays until the typewriter is quiescent. Then, the argument dcrsr is placed into the typewriter's receiver control and status register, and dctsr is placed in the transmitter control and status register. The DC-11 manual must be consulted for the format of these words. For the purpose of this call, the most important rôle of these arguments is to adjust to the speed of the typewriter.

The mode argument contains several bits which determine the system's treatment of the typewriter:

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 LF-CR
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)

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 new-line function).

Additional bits in the high order byte of the mode argument are used to indicate that the terminal is an IBM 2741 and to specify 2741 modes. These mode bits are:

400	terminal is an IBM 2741
1000	the 2741 has the transmit interrupt feature (currently ignored)
2000	use correspondence code conversion on output

4000 use correspondence code conversion on input (currently ignored)

Normal input and output code conversion for 2741s is EBCDIC (e. g. 963 ball and corresponding keyboard). The presence of the transmit interrupt feature permits the system to do read-ahead while no output is in progress. In 2741 mode, the low order bits 331 are ignored.

SEE ALSO stty(I), gtty(II)

DIAGNOSTICS The error bit (c-bit) is set if the file descriptor does not refer to a typewriter.

BUGS This call should be used with care.

NAME sync -- update super-block

SYNOPSIS sys sync / sync = 36.; not in assembler

DESCRIPTION sync causes the super block for all file systems to be written out. It is only necessary on systems in which this writing may be delayed for a long time, i.e., those which incorporate hardware protection facilities.

It should be used by programs which examine a file system, for example check, df, tm, etc.

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME time -- get time of year

SYNOPSIS
```
sys time / time = 13.
(time r0-r1)
```

DESCRIPTION time returns the time since 00:00:00, Jan. 1, 1972, measured in sixtieths of a second. The high order word is in the r0 register and the low order is in the r1.

SEE ALSO date(I), mdate(II)

DIAGNOSTICS --

BUGS The time is stored in 32 bits. This guarantees a crisis every 2.26 years.

NAME times -- get process times

SYNOPSIS

```
sys  times; buffer  / times = 43.; not in assembler
...
buffer: .=.+[24.*3]
```

DESCRIPTION

times returns time-accounting information for the system as a whole, for the current process, and for the terminated child processes of the current process. All the times are 2-word (32-bit) numbers, and the unit of measurement is 1/60 second.

After the call, the buffer will appear as follows:

```
buffer:
system:
        .=.+4           / absolute time
        .=.+4           / total system time
        .=.+4           / total swap time
        .=.+4           / other I/O wait time
        .=.+4           / idle time
        .=.+4           / total user time

process:
        .=.+4           / (ignore)
        .=.+4           / time in system
        .=.+4           / (ignore)
        .=.+4           / I/O wait time
        .=.+4           / (ignore)
        .=.+4           / processor time

child:
        .=.+24.
```

The format of the "child" times is the same as that for the process times; the numbers are the sum of the times for all terminated direct or indirect descendants of the current process.

The "absolute" time returned is the same as that given by time(II). The "total system times" are times since the last cold boot.

FILES --

SEE ALSO time(II), time(I)

DIAGNOSTICS --

BUGS --

NAME umount -- dismount file system

SYNOPSIS sys umount; special / umount = 22.

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).

The user must take care that all activity on the file system has ceased.

SEE ALSO umount(I), mount(II)

DIAGNOSTICS Error bit (c-bit) set if no file system was mounted on the special file.

BUGS Use of this call should be restricted to the super-user.

NAME unlink -- remove directory entry

SYNOPSIS

```
sys unlink; name / unlink = 10.
```

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).

BUGS --

NAME

wait -- wait for process to die

SYNOPSIS

```
sys wait  / wait = 7.
(process ID in r0)
(termination status/user status in r1)
```

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 error bit set. In the case of several children several waits are needed to learn of all the deaths.

If the error bit is not set on return, the r1 high byte contains the low byte of the child process r0 when it terminated. The r1 low byte contains the termination status of the process from the following list:

0	exit
1	bus error
2	illegal instruction
3	trace trap
4	IOT trap
5	power fail trap
6	EMT trap
7	bad system call
8	PIR interrupt
9	floating point exception
10	memory violation
11	quit
12	interrupt
13	kill (see kill(II))
14	User I/O (not currently possible)
+16	core image produced

SEE ALSO

exit(II), fork(II)

DIAGNOSTICS

error bit (c-bit) on if no children not previously waited for.

BUGS

A child which dies, but is never waited for consumes a slot in the process table. When this table is full, the system is effectively hung.

NAME write -- write on file

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

DESCRIPTION A file descriptor is a word returned from a successful open or creat 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)

DIAGNOSTICS The error bit (c-bit) is set on an error: bad descriptor, buffer address, or count; physical I/O errors.

BUGS --

NAME atan -- arc tangent function

SYNOPSIS jsr r5,atan[2]

DESCRIPTION The atan entry returns the arc tangent of fr0 in fr0. The range is $-\pi/2$ to $\pi/2$.

The atan2 entry returns the arc tangent of fr0/fr1 in fr0. The range is $-\pi$ to π.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS there is no error return

BUGS --

NAME atof -- ascii to floating

SYNOPSIS jsr r5,atof; subr

DESCRIPTION atof will convert an ascii stream to a floating number returned in fr0.

The subroutine subr (supplied by the caller) is called on r5 for each character of the ascii stream. subr should return the character in r0. The first character not used in the conversion is left in r0.

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.

The subroutine subr must not disturb any registers.

FILES kept in /lib/liba.a

SEE ALSO Calls atoi (III)

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 with the carry bit.

NAME atoi -- ascii to integer

SYNOPSIS jsr r5,atoi; subr

DESCRIPTION atoi will convert an ascii stream to a binary number returned in r1.

The subroutine subr (supplied by the caller) is called on r5 for each character of the ascii stream. subr should return the character in r0. The first character not used in the conversion is left in r0.

The numbers recognized are: an optional minus sign followed by a string of digits.

The subroutine subr must not disturb any registers.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS There are none; the routine charges on regardless of consequences; see BUGS.

BUGS It pays no attention to overflow - you get whatever the machine instructions mul and div happen to leave in the low order half - in fact, the carry bit should be set and isn't.

The routine should accept initial "+" and initial blanks.

NAME compar -- default comparison routine for qsort

SYNOPSIS jsr pc,compar

DESCRIPTION Compar is the default comparison routine called by qsort and is separated out so that the user can supply his own comparison.

The routine is called with the width (in bytes) of an element in r3 and it compares byte-by-byte the element pointed to by r0 with the element pointed to by r4.

Return is via the condition codes, which are tested by the instructions "blt" and "bgt". That is, in the absence of overflow, then the condition (r0) < (r4) should leave the Z-bit off and N-bit on while (r0) > (r4) should leave Z and N off. Still another way of putting it is that for elements of length 1 the instruction

```
cmpb    (r0),(r4)
```

suffices.

Only r0 is changed by the call.

FILES kept in /lib/liba.a

SEE ALSO qsort (III)

DIAGNOSTICS --

BUGS It could be recoded to run faster.

NAME crypt -- password encoding

SYNOPSIS
```
mov $key,r0
jsr pc,crypt
```

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".

Login uses this result as a password.

FILES kept in /lib/liba.a

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

DIAGNOSTICS there are none; garbage is accepted.

BUGS --

NAME ctime -- convert date and time to ASCII

SYNOPSIS

```
sys     time
mov     $buffer,r2
jsr     pc,ctime
```

DESCRIPTION The output buffer is 16 characters long and the time has the format

```
Oct  9 17:32:24\0
```

The input time must be in the r0 and r1 registers in the form returned by sys time.

FILES kept in /lib/liba.a

SEE ALSO ptime(III), time(II)

DIAGNOSTICS --

BUGS The routine must be reassembled for leap year. Dec 31 is followed by Dec 32 and so on.

NAME ddsput -- put a character on display data set

SYNOPSIS

```
(file descriptor in r0)
jsr     pc,ddsinit

(character in r0)
jsr     pc,ddsput
```

DESCRIPTION

These routines provide an interface to the Display Data Set, a peculiar device which can be called by Picturephone sets and which will display some of the ASCII character set and certain other graphics on the Picturephone screen.

If the DC11 or other interface hardware is not already set up to talk to the Display Data Set, the ddsinit entry should be called with the appropriate file descriptor in r0. On the only known DDS attached to UNIX, the associated special file is called "/dev/ttyc". ddsinit also clears the display.

Thereafter, characters may be displayed by calling ddsput. To the extent possible, ddsput simulates an ordinary terminal. Characters falling to the right of the 22X22 screen area are ignored; the 23rd line on the screen causes the screen to be erased and that line to be put at the top of the new display. Certain ASCII characters are interpreted specially as follows:

FF	clear screen, go to top left
HT	expand to right number of spaces
DC1	treat as reverse line feed (move N)
DC2	move cursor 1 place right (move E)
DC3	forward line feed (move S)
DC4	backspace 1 position (move W)
SO	enter graph mode
SI	leave graph mode
CR	put cursor at start of current line

Graph mode allows display of the non-ASCII characters and will be described when hell freezes over.

Lower-case ASCII alphabetics are mapped into upper case. Several ASCII non-alphabetic graphics are unavailable as well. Also the lower right circle of the "%" character is missing. Also one of the circuit cards in the DDS has a crack in it and sometimes it doesn't work. All in all, it is best to avoid this device.

FILES kept in /lib/liba.a

SEE ALSO AT&T writeup on DDS

NAME ecvt, fcvt -- output conversion

SYNOPSIS

```
jsr pc,ecvt

or

jsr pc,fcvt
```

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).

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".

FILES kept in /lib/liba.a

SEE ALSO ftoa(III)

DIAGNOSTICS --

BUGS --

NAME exp -- exponential function

SYNOPSIS
```
jsr     r5,exp
```

DESCRIPTION The exponential of fr0 is returned in fr0.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS If the result is not representable, the c-bit is set and the largest positive number is returned.

Zero is returned if the result would underflow.

BUGS ---

NAME ftoa -- floating to ascii conversion

SYNOPSIS jsr r5,ftoa; subr

DESCRIPTION ftoa will convert the floating point number in fr0 into ascii in the form

[-]ddddd.dd*

if possible, otherwise in the form

[-]d.ddddddde[-]dd*.

For each character generated by ftoa, the subroutine subr (supplied by the caller) is called on register r5 with the character in r0.

The number of digits can be changed by changing the value of "_ndigits" in ecvt (default is 10.).

The subroutine subr must not disturb any registers.

FILES kept in /lib/liba.a

SEE ALSO ecvt(III), itoa(III)

DIAGNOSTICS —

BUGS —

NAME ftoo -- floating to octal conversion

SYNOPSIS jsr r5,ftoo; subr

DESCRIPTION ftoo wil convert the floating point number in fr0 into ascii in the conventional octal form

000000;000000;000000;000000

For each character generated by ftoo, the subroutine subr (supplied by the caller) is called on register r5 with the character in r0.

The subroutine subr must not disturb any registers.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME connect, gerts -- Gerts communication over 201

SYNOPSIS

```
jsr     r5,connect
(error return)
...

jsr     r5,gerts; fc; oc; ibuf; obuf
(error return)
...
```

other entry points: gcset, gout

DESCRIPTION The GCOS GERTS interface is so bad that a description here is inappropriate. Anyone needing to use this interface should seek divine guidance.

FILES /dev/dn0, /dev/dp0
kept in /lib/liba.a

SEE ALSO dn(IV), dp(IV), HIS documentation

DIAGNOSTICS —

BUGS —

NAME

getw, getc, fopen -- buffered input

SYNOPSIS

```
mov     $filename,r0
jsr     r5,fopen; iobuf

jsr     r5,getc; iobuf
(character in r0)

jsr     r5,getw; iobuf
(word in r0)
```

DESCRIPTION

These routines are used to 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
```

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.

getc returns the next byte from the file in r0. The error bit is set on end of file or a read error.

getw returns the next word in r0. getc and getw may be used alternately; there are no odd/even problems.

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.

FILES

kept in /lib/liba.a

SEE ALSO

open(II), read(II), putc(III)

DIAGNOSTICS

c-bit set on EOF or error

BUGS

--

NAME hypot -- calculate hypotenuse

SYNOPSIS

```
movf    a,fr0
movf    b,fr1
jsr     r5,hypot
movf    fr0,...
```

DESCRIPTION The square root of fr0*fr0 + fr1*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.

FILES kept in /lib/liba.a

SEE ALSO sqrt(III)

DIAGNOSTICS The c-bit is set if the result cannot be represented.

BUGS --

NAME itoa -- integer to ascii conversion

SYNOPSIS

```
jsr     r5,itoa; subr
```

DESCRIPTION itoa will convert the number in r0 into ascii decimal preceded by a - sign if appropriate. For each character generated by itoa, the subroutine subr (supplied by the caller) is called on register r5 with the character in r0.

The subroutine subr must not disturb any registers.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME log -- logarithm (base e)

SYNOPSIS jsr r5,log

DESCRIPTION The logarithm (base e) of fr0 is returned in fr0.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS The error bit (c-bit) is set if the input argument is less than or equal to zero and the result is set to the largest negative number.

BUGS --

NAME mesg -- write message on typewriter

SYNOPSIS jsr r5,mesg; <Now is the time\0>; .even

DESCRIPTION mesg writes the string immediately following its call onto the standard output file. The string must be terminated by an ASCII NULL byte.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME nlist -- get entries from name list

SYNOPSIS

```
        jsr r5,nlist; file; list
        ...
file:   <file name\0>; .even
list:
        <name1xxx>; type1; value1
        <name2xxx>; type2; value2
        ...
        0
```

DESCRIPTION nlist will examine the name list in the given assembler output file and selectively extract 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 zero. 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 /sys/sys/unix. In this way programs can obtain system 'magic' numbers that are up to date.

FILES kept in /lib/liba.a

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 — pow -- floating exponentiation x^y

SYNOPSIS

```
movf    x,fr0
movf    y,fr1
jsr     pc,pow
movf    fr0,...
```

DESCRIPTION — The value of x^y (i.e. x^y) is returned in fr0.

0^x returns zero for all x.

(-x)^y returns a result only if y is an integer.

FILES — kept in /lib/liba.a

SEE ALSO — exp(III), log(III)

DIAGNOSTICS — The carry bit is set on return in case of overflow or in case of 0^0 or (-x)^y for y non-integer.

BUGS — --

NAME ptime -- print date and time

SYNOPSIS

```
sys     time
mov     file,r2
jsr     pc,ptime
```

DESCRIPTION ptime prints the date and time in the form

```
Oct  9 17:20:33
```

on the file whose file descriptor is in r2. The string is 15 characters long. The time to be printed must be placed in the r0 and r1 registers in the form returned by sys time.

FILES kept in /lib/liba.a

SEE ALSO time(II), ctime(III) (used to do the conversion)

DIAGNOSTICS —

BUGS see ctime

NAME putc, putw, fcreat, flush -- buffered output

SYNOPSIS

```
mov     $filename,r0
jsr     r5,fcreat; iobuf

(get byte in r0)
jsr     r5,putc; iobuf

(get word in r0)
jsr     r5,putw; iobuf

jsr     r5,flush; iobuf
```

DESCRIPTION

fcreat creates the given file (mode 17) and sets up the buffer iobuf (size 518(10) bytes); putc and putw write a byte or word respectively onto the file; flush forces the contents of the buffer to be written, 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
```

fcreat sets the error bit (c-bit) if the file creation failed; none of the other routines return error information.

Before terminating, a program should call flush to force out the last of the output.

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 flush, close the file, and call fcreat again.

FILES kept in /lib/liba.a

SEE ALSO creat(II), write(II), getc(III)

DIAGNOSTICS error bit possible on fcreat call

BUGS --

NAME qsort -- quicker sort

SYNOPSIS
```
(base of data in r1)
(end+1 of data in r2)
(element width in r3)
jsr pc,qsort
```

DESCRIPTION qsort is an implementation of the quicker sort algorithm. It 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, r4 are destroyed.

The routine compar (q.v.) is called to compare elements and may be replaced by the user.

FILES kept in /lib/liba.a

SEE ALSO compar(III)

DIAGNOSTICS —

BUGS It scribbles on r4.

NAME rand -- random number generator

SYNOPSIS
```
jsr pc,srand    /to initialize
jsr pc,rand     /to get a random number
```

DESCRIPTION The routine 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 in r0.

It can be set to a random starting point by calling srand with whatever you like in r0, for example the result left in r1 from sys time.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS --

BUGS --

WARNING The author of this routine has been writing random-number generators for many years and has never been known to write one that worked.

NAME salloc -- string manipulation routines

SYNOPSIS

```
(get size in r0)
jsr     pc,allocate

(get source pointer in r0,
destination pointer in r1)
jsr     pc,copy

jsr     pc,wc

(all following instructions assume r1 contains pointer)

jsr     pc,release

(get character in r0)
jsr     pc,putchar

jsr     pc,lookchar
(character in r0)

jsr     pc,getchar
(character in r0)

(get character in r0)
jsr     pc,alterchar

(get position in r0)
jsr     pc,seekchar

jsr     pc,backspace
(character in r0)

(get word in r0)
jsr     pc,putword

jsr     pc,lookword
(word in r0)

jsr     pc,getword
(word in r0)

(get word in r0)
jsr     pc,alterword

jsr     pc,backword
(word in r0)

jsr     pc,length
(length in r0)

jsr     pc,position
(position in r0)

jsr     pc,rewind
```

```
jsr     pc,create

jsr     pc,fsfile

jsr     pc,zero
```

DESCRIPTION

This package is a complete set of routines for dealing with almost arbitrary length strings of words and bytes. The strings are stored on a disk file, so the sum of their lengths can be considerably larger than the available core.

For each string there is a header of four words, namely a write pointer, a read pointer and pointers to the beginning and end of the block containing the string. Initially the read and write pointers point to the beginning of the string. All routines that refer to a string require the header address in r1. Unless the string is destroyed by the call, upon return r1 will point to the same string, although the string may have grown to the extent that it had to be be moved.

allocate obtains a string of the requested size and returns a pointer to its header in r1.

release releases a string back to free storage.

putchar and putword write a byte or word respectively into the string and advance the write pointer.

lookchar and lookword read a byte or word respectively from the string but do not advance the read pointer.

getchar and getword read a byte or word respectively from the string and advance the read pointer.

alterchar and alterword write a byte or word respectively into the string where the read pointer is pointing and advance the read pointer.

backspace and backword read the last byte or word written and decrement the write pointer.

All write operations will automatically get a larger block if the current block is exceeded. All read operations return with the error bit set if attempting to read beyond the write pointer.

seekchar moves the read pointer to the offset specified in r0.

length returns the current length of the string (beginning pointer to write pointer) in r0.

position returns the current offset of the read pointer in r0.

rewind moves the read pointer to the beginning of the string.

create returns the read and write pointers to the beginning of the string.

fsfile moves the read pointer to the current position of the write pointer.

zero zeros the whole string and sets the write pointer to the beginning of the string.

copy copies the string whose header pointer is in r0 to the string whose header pointer is in r1. Care should be taken in using the copy instruction since r1 will be changed if the contents of the source string is bigger than the destination string.

wc forces the contents of the internal buffers and the header blocks to be written on disc.

FILES

The allocator is in /lib/libs.a; the -s option to ld will link edit references to the allocator.

alloc.d is the temporary file used to contain the strings.

SEE ALSO

--

DIAGNOSTICS

"error in copy" if a disk write error occurs during the execution of the copy instruction. "error in allocator" if any routine is called with a bad header pointer. "Cannot open output file" if file alloc.d cannot be created or opened. "Out of space" if there's no available block of the requested size or no headers available for a new block.

BUGS

--

NAME sin, cos -- sine cosine

SYNOPSIS jsr r5,sin (cos)

DESCRIPTION The sine (cosine) of fr0 in radians is returned in fr0.

The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS there are none

BUGS --

NAME sqrt -- square root function

SYNOPSIS jsr r5,sqrt

DESCRIPTION The square root of fr0 is returned in fr0.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS The c-bit is set on negative arguments and 0 is returned.

BUGS --

NAME switch -- switch on value

SYNOPSIS
```
(switch value in r0)
jsr     r5,switch; swtab
(not-found return)
...
swtab: val1; lab1;
...
valn; labn
..; 0
```

DESCRIPTION switch compares the value of r0 against each of the val_i; if a match is found, control is transferred to the corresponding lab_i (after popping the stack once). If no match has been found by the time a null lab_i occurs, switch returns.

FILES kept in /lib/liba.a

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME ttyn -- return name of current tty

SYNOPSIS jsr pc,ttyn

DESCRIPTION The routine hunts up the name of the input tty attached to the process (one byte from the set {012345678abc} at present) and returns it in r0.

"x" is returned if no genuine input tty is attached to the process.

FILES kept in /lib/liba.a

SEE ALSO fstat(II)

DIAGNOSTICS --

BUGS --

NAME dc -- DC-11 communications interfaces

DESCRIPTION

The special files /dev/tty0, /dev/tty1, ... refer to the DC11 asynchronous communications interfaces. At the moment there are ten of them, but the number is subject to change.

When one of these files is opened, it causes the process to wait until a connection is established. (In practice, however, user's programs seldom open these files; they are opened by init and become a user's standard input and output file.) The very first typewriter file open in a process becomes the control typewriter 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 150 characters. When this is happening the character "#" is echoed for every lost input character.

When first opened, the interface mode is ASCII characters; 150 baud; even parity only 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 the following hardware states are program settable independently for input and output (see DC11 manual): 134.5, 150, 300, or 1200 baud; one or two stop bits on output; and 5, 6, 7, or 8 data bits/character. 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; the printing of tabs as spaces; and setting the system to handle IBM 2741s. See getty(VII) 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.

It is possible to use raw mode in which the program reading is awakened on each character. In raw mode, no erase or kill processing is done; and the EOT, quit and interrupt characters are not treated specially.

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 not passed on except in raw mode.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) 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 the interrupt signal. When this character is received from a given typewriter, a search is made for all processes which have this typewriter as their control typewriter, and which have not informed the system that they wish to ignore interrupts. If there is more than one such process, one of these is selected, for practical purposes at random. The process is either forced to exit or a trap is simulated to an agreed-upon location in the process. See intr(II).

The ASCII character FS is the quit signal. Its treatment is identical to the interrupt signal except that unless the receiving process has made other arrangements it will not only be terminated but a core image file will be generated. See quit(II). The character is not passed on

except in raw mode.

Output is prosaic compared to input. 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 to prevent terminals which respond to it from being hung up.

The system will handle IBM 2741 terminals. See getty(VII) for the way that 2741s are detected. In 2741 mode, the hardware state is: 134.5 baud; one output stop bit; and 7 bits/character. Because the 2741 is inherently half-duplex, input is not echoed. Proper function delays are provided. For 2741s without a feature known as "transmit interrupt" it is not possible to collect input ahead of the time that a program reads the typewriter, because once the keyboard has been enabled there is no way to send further output to the 2741. It is currently assumed that the feature is absent; thus the keyboard is unlocked only when some program reads. The interrupt signal (normally ASCII DEL) is simulated when the 2741 "attention" key is pushed to generate either a 2741 style EOT or a break. It is not possible to generate anything corresponding to the end-of-file EOT or the quit signal. Currently IBM EBCDIC is default for input and output; correspondence code output is settable (see stty(I)). The full ASCII character set is not available: "[", "]", "{", "}", "~", are missing on input and are printed as blank on output; "¢" is used for "\"; "¬" for "^"; "¦" for both "," and "{" on output; and "," maps into ";" on input. Similar mappings occur with correspondence code output.

FILES	/dev/tty[01234567ab]	113B dataphones
	/dev/ttyc	display data set
	/dev/ttyd	113B with /dev/dn1

SEE ALSO kl(IV), getty(VII)

BUGS The primarily Model 37 oriented delays may not be appropriate for all other ASCII terminals.

NAME dn -- dn-11 ACU interface

DESCRIPTION dn? is a write-only file. Bytes written on dn? must be ASCII as follows:

0-9	dial 0-9
:	dial *
;	dial #
=	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 only one of the ACU's (113C) actually requires it.

FILES

/dev/dn0	connected to 801 with dp0
/dev/dn1	connected to 113C with ttyd
/dev/dn2	not currently connected

SEE ALSO dp(IV), dc(IV), write(II)

BUGS --

NAME dp -- dp-11 201 data-phone interface

DESCRIPTION dp? is a 201 data-phone interface file. read and write calls to dp? are limited to a maximum of 400 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 20 second time out is set and a zero byte record is returned if nothing is received in that time.

FILES /dev/dp0 201 dataphone used to call GCOS

SEE ALSO dn(IV), gerts(III)

BUGS The dp file is GCOS oriented. It should be more flexible.

NAME kl -- KL-11/TTY-33 console typewriter

DESCRIPTION

tty (as distinct from tty?) refers to the console typewriter hard-wired to the PDP-11 via a KL-11 interface.

Generally, the disciplines involved in dealing with tty are similar to those for tty? and section dc(IV) should be consulted. The following differences are salient:

The system calls stty and gtty do not apply to this device. It cannot be placed in raw mode; on input, upper case letters are always mapped into lower case letters; a carriage return is echoed when a line-feed is typed.

The quit character is not FS (as with tty?), but is generated by the key labelled "alt mode."

By appropriate console switch settings, it is possible to cause UNIX to come up as a single-user system with I/O on this device.

FILES

/dev/tty
/dev/tty8 synonym for /dev/tty

SEE ALSO dc(IV), init(VII)

BUGS --

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.

Mem is a byte-oriented file; its bytes are numbered 0 to 65,535.

If a non-existent memory location is referenced, the user suffers the resultant bus error.

Memory referenced through the file is treated with movb instructions. Certain device registers do not implement DATOB cycles to odd addresses. Other registers react strangely to this addressing.

FILES /dev/mem

SEE ALSO --

BUGS --

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

SEE ALSO --

BUGS --

NAME rf -- RF11-RS11 fixed-head disk file

DESCRIPTION This file refers to the concatenation of both RS-11 disks. It may be either read or written, although writing is inherently very dangerous, since a file system resides there.

The disk contains 2048 256-word blocks, numbered 0 to 2047. Like the other block-structured devices (TC, RK) this file is addressed in blocks, not bytes. This has two consequences: seek calls refer to block numbers, not byte numbers; and sequential reading or writing always advance the read or write pointer by at least one block. Thus successive reads of 10 characters from this file actually read the first 10 characters from successive blocks.

FILES /dev/rf0

SEE ALSO tc(IV), rk(IV)

BUGS The fact that this device is addressed in terms of blocks, not bytes, is extremely unfortunate. It is due entirely to the fact that read and write pointers (and consequently the arguments to seek) are single-precision numbers.

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. Like the RF disk and the tape files, its addressing is block-oriented. Consult the rf(IV) section.

FILES

/dev/rk0	user available drive
/dev/rk1	/usr file system
/dev/rk2	/sys file system
/dev/rk3	/crp file system

SEE ALSO rf(IV), tc(IV)

BUGS See rf(IV)

NAME tc -- TC-11/TU56 DECtape

DESCRIPTION The files tap0 ... tap7 refer to the TC-11/TU56 DECtape drives 0 to 7. Since the logical drive number can be manually set, all eight files exist even though at present there are fewer physical drives.

The 256-word blocks on a standard DECtape are numbered 0 to 577. However, the system makes no assumption about this number; a block can be read or written if it exists on the tape and not otherwise. An error is returned if a transaction is attempted for a block which does not exist.

Addressing on the tape files, like that on the RK and RF disks, is block-oriented.

FILES /dev/tap?

SEE ALSO rf(IV), tap(I)

BUGS see rf(IV)

NAME

tm -- TM-11/TU-10 magtape interface

DESCRIPTION

mt? is the DEC TU10/TM11 magtape. When opened for reading or writing, the magtape is rewound. A tape consists of a series of 512 byte records terminated by an end-of-file. Reading less than 512 bytes causes the rest of a record to be ignored. Writing less than a record causes null padding to 512 bytes. When the magtape is closed after writing, an end-of-file is written.

Seek has no effect on the magtape. The magtape can only be opened once at any instant.

FILES

/dev/mt0 selected drive 0

SEE ALSO

mt(I)

BUGS

Seek should work on the magtape. Also, a provision of having the tape open for reading and writing should exist. A multi-file and multi-reel facility should be incorporated.

NAME vt -- 11/20 (vt01) interface

DESCRIPTION

The file vt0 provides the interface to a PDP 11/20 which runs both a VT01A-controlled Tektronix 611 storage display, and a Federal Screw Works (Vocal Interface Division) voice synthesizer. 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 and generate utterances via the voice synthesizer.

This special file operates in two basic modes, selected by bit 2 (octal 04) on the 11/20's console switches. If this bit is on at the opening of the file, all bytes written on the file are interpreted as ASCII characters 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 (octal 01). As explained below, this causes the program to pause before erasing until one of the attached pushbuttons is depressed.

If bit 2 of the switches is down, the display is in graphic mode. In this case bytes written on the file are interpreted as display and vocal 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 ($\pm$03777)X($\pm$03777) (numbers in octal); that is, 12 bits of precision. Attempts to plot points outside the screen limits are ignored.

The graphic and sonic commands are:

order (1); 1 parameter byte
The parameter indicates a subcommand, possibly followed by subparameter bytes, as follows:

erase (1)
The screen is erased. This action may be delayed, as explained below, until a pushbutton is depressed.

label (2); several subparameter bytes
The following bytes up to a null character are taken as a label and typed on the console TTY. One of the console switches gives labels a special interpretation, as explained below.

display 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.

speak(8); several parameter bytes
The following bytes up to a null byte are taken to represent phonemes which are fed to the voice synthesizer. vsp(VII) gives the encoding.

The 3 low-order console switches of the 11/20 modify the operation of the display as follows.

Bit 2 (octal 04) is examined at the time the display file is opened (more precisely, when the first byte is written after an open); as indicated, when on it selects character mode, otherwise graphic mode.

Bit 1 (octal 02) determines whether TTY labels are to be interpreted. Unless this bit is on, labels are ignored. (except to terminate skip mode, see below).

Bit 0 (octal 01) determines whether the display will pause before erasing the screen; if off there will be no pause. If bit 0 is on, the erase will occur and displaying will resume only when one of the 16 pushbuttons is depressed.

There is a box with 16 pushbuttons connected to the 11/20. Their state is at all times available in the 11/45 by executing the csw system call (II). They are used by the 11/20 when it is pausing before an erase. 14 of the buttons merely serve to allow the display to continue. If, however, button 7 is pushed, the display will ignore commands up to the next erase command, then ring the TTY console's bell, thereby skipping an entire picture.

If button 8 is depressed, the display will ignore commands up to the next TTY label (whether or not its typing is suppressed) before resuming the displays. Thus a sequence of frames may be skipped.

FILES /dev/vt0

SEE ALSO csw(II), vsp(VII)

BUGS Two users using vt0 simultaneously can interfere with each other, e.g. plot phonemes or speak display coordinates.

NAME a.out -- assembler and link editor output

DESCRIPTION

a.out is the output file of the assembler as and the link editor ld. In both cases, a.out may be executed provided 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(8))
2 The size of the program text segment
3 The size of the initialized data segment
4 The size of the uninitialized (bss) segment
5 The size of the symbol table
6 The entry location (always 0 at present)
7 The stack size required (0 at present)
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 segments are set up: the text segment, the data segment, and the bss (uninitialized data) segment, in that order. The text segment begins at the lowest location in the core image; the header is not loaded. The data segment begins immediately after the text segment, and the bss segment immediately after the data segment. The bss segment is initialized by 0's. In the future the text segment will be write-protected and shared.

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 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

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 the data segment
06 indicates the reference is to the bss segment
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, 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, ld, strip, nm, un(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

NAME format of core image

DESCRIPTION UNIX writes out a core image of a terminated process when any of various errors occur. See wait(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 size and structure of the core image file depend to some extent on which system is involved. In general there is a 512-byte area at the end which contains the system's per-process data for that process. (64 bytes in older systems). The remainder represents the actual contents of the user's core area when the core image was written. In the current system, this area is variable in size in that only the locations from user 0 to the program break, plus the stack, are dumped.

When any fatal trap occurs, all the useful registers are stored on the stack. In the current system, which has relocation and protection hardware, the stack used is the system stack, which is kept in the per-process area; in older systems, there is only one stack, and it is located in the user's core area.

The actual format of the information is complicated because it depends on what hardware is present (EAE, floating-point option), whether single- or double-precision floating mode is in effect, and also involves relocating addresses in the system's address space. 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), wait(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 10 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 2-9 represent the (8-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 format of file system

DESCRIPTION

Every file system storage volume (e.g. RF disk, RK 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. Blocks 0 and 1 are collectively known as the super-block for the device; they define its extent and contain an i-node map and a free-storage map. The first word contains the number of bytes in the free-storage map; it is always even. It is followed by the map. There is one bit for each block on the device; the bit is "1" if the block is free. Thus if the number of free-map bytes is n, the blocks on the device are numbered 0 through $8n-1$. The free-map count is followed by the free map itself. The bit for block k of the device is in byte $k/8$ of the map; it is offset $k \pmod 8$ bits from the right. Notice that bits exist for the super-block and the i-list, even though they are never allocated or freed.

After the free map is a word containing the byte count for the i-node map. It too is always even. I-numbers below 41(10) are reserved for special files, and are never allocated; the first bit in the i-node free map refers to i-number 41. Therefore the byte number in the i-node map for i-node i is $(i-41)/8$. It is offset $(i-41) \pmod 8$ bits from the right; unlike the free map, a "0" bit indicates an available i-node.

I-numbers begin at 1, and the storage for i-nodes begins at 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 $(i+31)/16$ of the file system, and begins $32 \cdot ((i+31) \pmod{16})$ bytes from its start.

There is always one file system which is always mounted; in standard UNIX it resides on the RF disk. This device is also used for swapping. On the primary file system device, there are several pieces of information following that previously discussed. There are two words with the calendar time (measured since 00:00 Jan 1, 1972); two words with the time spent executing in the system; two words with the time spent waiting for I/O on the RF and RK disks; two words with the time spent executing in a user's core; one byte with the count of errors on the RF disk; and one byte with the count of errors on the RK disk. All the times are measured in sixtieths of a second.

I-node 41(10) is reserved for the root directory of the file system. No i-numbers other than this one and those from 1 to 40 (which represent special files) have a built-in meaning. Each i-node represents one file. The

format of an i-node is as follows, where the left column represents the offset from the beginning of the i-node:

```
0-1     flags (see below)
2       number of links
3       user ID of owner
4-5     size in bytes
6-7     first indirect block or contents block
...
20-21   eighth indirect block or contents block
22-25   creation time
26-29   modification time
30-31           unused
```

The flags are as follows:

```
100000  i-node is allocated
040000  directory
020000  file has been modified (always on)
010000  large file
000040  set user ID on execution
000020  executable
000010  read, owner
000004  write, owner
000002  read, non-owner
000001  write, non-owner
```

The allocated bit (flag 100000) is believed even if the i-node map says the i-node is free; thus corruption of the map may cause i-nodes to become unallocatable, but will not cause active nodes to be reused.

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 corresponding to b is the bth entry in the address portion of the i-node.

Even if the file is large, b will be less than 128 (128*512 = 2^16). The first number in the i-node address portion gives the physical block number of the indirect block. b is doubled to give a byte offset in the indirect block and the word there found is the physical address of the block corresponding to b.

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.

BUGS

Two blocks are not enough to handle the i- and free-storage maps for an RP02 disk pack, which contains around 10 million words.

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
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

tap -- DEC/mag tape formats

DESCRIPTION

The DECtape command tap and the magtape command mt dump and extract files to and from their respective tape media. The formats of these tapes are the same except that magtapes have larger directories.

Block zero of the tape is not used. It is available to contain a boot program to be used in a stand-alone environment. This has proved valuable for DEC diagnostic programs.

Blocks 1 through 24 for DECtape (1 through 146 for magtape) contain a directory of the tape. There are 192 (resp. 1168) entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

path name	32 bytes
mode	1 byte
uid	1 byte
size	2 bytes
time modified	4 bytes
tape address	2 bytes
unused	20 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, size and time modified are the same as described under i-nodes (see 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. 147) on are available for file storage.

A fake entry (see mt(I), tap(I)) has a size of zero.

SEE ALSO

filesystem(V), mt(I), tap(I)

NAME /tmp/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 /etc/init, which maintains the file;
who(I), which interprets it.

NAME /tmp/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, and the typewriter name 'x' indicates that UNIX was rebooted at that point.

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.

This file resides in directory /tmp.

SEE ALSO init(VII), login(I), acct(VIII), swtmp(VIII)

NAME

bc -- B interpreter

SYNOPSIS

bc [-c] sfile$_1$.b ... ofile$_1$ ...

DESCRIPTION

bc is the UNIX B interpreter. It accepts three types of arguments:

Arguments whose names end with ".b" are assumed to be B source programs; they are compiled, and the object program is left on the file sfile$_1$.o (i.e. the file whose name is that of the source with ".o" substituted for ".b").

Other arguments (except for "-c") are assumed to be either loader flag arguments, or B-compatible object programs, typically produced by an earlier bc run, or perhaps libraries of B-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 language itself is described in [1].

The future of B is uncertain. The language has been totally eclipsed by the newer, more powerful, more compact, and faster language C.

FILES

file.b	input file
a.out	loaded output
b.tmp1	temporary (deleted)
b.tmp2	temporary (deleted)
/usr/lang/bdir/b[ca]	translator
/usr/lang/bdir/brt[12]	runtime initialization
/usr/lib/libb.a	builtin functions, etc.
/usr/lang/bdir/bilib.a	interpreter library

SEE ALSO

[1] K. Thompson; MM-72-1271-1; Users' Reference to B.
cc(I)

DIAGNOSTICS

see [1].

BUGS

Certain external initializations are illegal. (In particular: strings and addresses of externals.)

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 loss) is printed. To exit, hit the interrupt key (DEL) and the action and standing will be printed.

NAME ptx -- permuted index

SYNOPSIS ptx 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", "and", "as", "is", "for", "of", "on", "or", "the", "to", "up".

The index for this manual was generated using ptx.

FILES --

SEE ALSO sort(I)

DIAGNOSTICS some

BUGS --

NAME

yacc -- yet another compiler compiler

SYNOPSIS

/crp/scj/yacc [<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 tables are provided in readable form on the standard output and in b-compiler format on file actn.b; the program /crp/scj/bpar.b will parse strings using the actn.b file.

If your grammar is too big for yacc, you may try /crp/scj/bigyacc, some of whose size limits are larger, and others smaller.

FILES

actn.b	output tables
actn.tmp	temporary storage

Note that these files are created in the invoker's directory. The file actn.tmp is only created by /crp/scj/bigyacc (see above).

SEE ALSO

Yacc manual, by scj (available from ek); "LR Parsing", by A. V. Aho and S. C. Johnson, to be published.

DIAGNOSTICS

There are various diagnostics, but only one can be obtained in each run.

BUGS

The maximum number of terminal and non-terminal symbols is 50 each, and this is not checked. There are undoubtedly other bugs too.

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 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 gerts interface.

dpd uses the directory /usr/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. /usr/dpd is scanned for any file 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 /usr/dpd/snumb and converting that to decimal 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 the lock is gone, the daemon will exit.

FILES /dev/dn0, /dev/dp0, /usr/dpd/*

SEE ALSO opr(I)

DIAGNOSTICS --

BUGS --

NAME getty -- set typewriter mode and get user's name

SYNOPSIS /etc/getty

DESCRIPTION

getty is invoked by init (VII) immediately after a typewriter is opened following a dial-in. 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 150 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 37 Teletype terminal into full-duplex and unlock its keyboard). 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 300 baud and the "login:" is typed again, this time with the appropriate sequence which puts a GE TermiNet 300 into full-duplex. This sequence is acceptable to other 300 baud terminals also. If a subsequent null character is received, the speed is changed again. The general approach is to cycle through a set of speeds in response to null characters caused by breaks. The sequence at this installation is 150, 300, and 134.5 baud.

Detection of IBM 2741s is accomplished while the speed is set to 150 baud. The user sends a 2741 style "eot" character by pushing the attention key or by typing return; at 150 baud, this character looks like the ascii "~" (174_8). Upon receipt of the "eot", the system is set to operate 2741s and a "login:" message is typed.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to 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.

FILES --

SEE ALSO init(VII), login(I), stty(II)

NAME glob -- generate command arguments

SYNOPSIS /etc/glob

DESCRIPTION glob is used to expand arguments to the shell containing "*", '[', or "?". It is passed the argument list containing the metacharacters; glob expands the list and calls the command itself. The actions of glob are detailed in the Shell writeup.

FILES found in /etc/glob

SEE ALSO sh(I)

DIAGNOSTICS "No match", "No command", "No directory"

BUGS If any of '*', '[', or '?' occurs both quoted and unquoted in the original command line, even the quoted metacharacters are expanded.

glob gives the "No match" diagnostic only if no arguments at all result. This is never the case if there is any argument without a metacharacter.

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	∑	R
tau	τ	I	phi	φ	U	PHI	Φ	F
psi	ψ	V	PSI	Ψ	H	omega	ω	C
OMEGA	Ω	Z	nabla	∇	[	not	¬	_
partial	∂	]	integral	∫				

FILES

--

SEE ALSO

ascii (VII)

DIAGNOSTICS

--

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 tty is opened for reading and writing and the shell is invoked immediately. This feature is used to bring up a test system, or one which does not contain DC-11 communications interfaces. When the system is brought up in this way, the getty and login routines mentioned below and described elsewhere are not needed.

Otherwise, init does some housekeeping: the mode of each DECtape file is changed to 17 (in case the system crashed during a tap command); directory /usr is mounted on the RK0 disk; directory /sys is mounted on the RK1 disk. Also a data-phone daemon is spawned to restart any jobs being sent.

Then init forks several times to create a process for each typewriter mentioned in an internal table. 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 someone is dialled in (and carrier established) on the channel. Then the process executes the program /etc/getty (q.v.). getty will read the user's name and invoke 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 wtmp, which maintains a history of logins and logouts. Then the appropriate typewriter is reopened and getty reinvoked.

FILES /dev/tap?, /dev/tty, /dev/tty?, /tmp/utmp, /tmp/wtmp

SEE ALSO login(I), login(VII), getty(VII), sh(I), dpd(VII)

DIAGNOSTICS none possible

BUGS none possible

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.

FILES --

SEE ALSO sh(I), glob(VII)

DIAGNOSTICS "?"

BUGS --

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.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

NAME vsp -- voice synthesizer code

SYNOPSIS cat /usr/pub/vsp

DESCRIPTION vsp contains a list of phonemes understood by the voice synthesizer on device vt. Phonemes are usually written in the form

comma inflection phoneme

The inflection and the phoneme codes are or-ed together. The phoneme codes are as follows (numbers in octal).

0	= 300	strong inflection	p	= 32	penny pound
1	= 200		a0	= 33	contact car
2	= 100		a1	= 52	connect
3	= 000	weak inflection	ai	= 37	name came
			aj	= 71	namely
aw	= 02	awful law	s	= 40	seven six
ie	= 03	zero	d	= 41	do diet
e0	= 04	enter met	f	= 42	four five
e1	= 76	seven	g	= 43	get grand
e2	= 77	seven	h	= 44	hello how
er	= 05	weather	j	= 45	judge edge
th	= 06	three thick	k	= 46	came lock
dh	= 07	this then	l	= 47	hello light
yu	= 27	use you	oo	= 50	look book
iu	= 10	unite	ou	= 51	good shoud
ju	= 11	new you	ng	= 53	ring angle
o0	= 31	only no	z	= 55	zero hazy
o1	= 12	hello	sh	= 56	show ship
o2	= 13	notice	ch	= 57	chair chime
u0	= 14	but must	v	= 60	seven even
u1	= 15	uncle	b	= 61	ball bed
u2	= 16	stirrup	n	= 62	nine seven
u3	= 34	app_le ab_le	m	= 63	mile men
ae	= 21	cat sat	iy	= 66	lie
ea	= 20	antenna	zh	= 70	azure pleasure
w	= 22	won wish	ih	= 72	station condition
ee	= 23	three	ay	= 36	may lay
r	= 24	radio radar			
t	= 25	two time	-0	= 35	long space
ey	= 26	sixty eighty	-1	= 17	
i0	= 30	six mix	-2	= 01	
i1	= 64	inept inside	-3	= 74	short delay
i2	= 65	cryptic static			

SEE ALSO speak(I), vt(IV)

NAME 20boot -- install new 11/20 system

SYNOPSIS 20boot [x]

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.

If no argument is given, the 11/20 program should be executing but idle; the 11/20 program is sent preceded by a "reboot" command. If an argument is given, the 11/20 should have been restarted at its ROM location 777300.

FILES /dev/vt0;
/sys/mdec/20.o (11/20 program)

SEE ALSO vt0 (IV)

DIAGNOSTICS --

NAME acct -- login accounting

SYNOPSIS acct [-w wtmp] [-p] [-d] people

DESCRIPTION acct 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. The people argument will limit the printout to only the specified login names. If no wtmp file is given, /usr/adm/wtmp is used.

FILES /usr/adm/wtmp

SEE ALSO init(VII), login(I), wtmp(V).

DIAGNOSTICS "Cannot open 'wtmp'" if argument is unreadable.

BUGS --

NAME bos, maki, vcboot, msys, et al.

DESCRIPTION

On the RF disk, the highest 16K words are reserved for stand-alone programs. These 16K words are allocated as follows:

bos	(1K)
Warm UNIX	(7K)
Cold UNIX	(8K)

The program bos (Bootstrap Operating System) examines the console switches and executes one of several internal programs depending on the setting. The following settings are currently recognized:

???	Will read Warm UNIX from the RF into core location 0 and transfer to 600.
1	Will read Cold UNIX from the RF into core location 0 and transfer to 600.
10	Will dump all of memory from core location 0 onto DECtape drive 7 and then halt.
20	Will read 256 words from RK0 into core 0 and transfer to zero. This is the procedure to boot DOS from an RK.
40	This is the same as 10 above, but instead of halting, UNIX warm is loaded.
0	Will load a standard UNIX binary paper tape into core location 0 and transfer to 0.
77500	Will load the standard DEC absolute and binary loaders and transfer to 77500.

All manual methods of booting the system involve manipulation of the console switches. In order for this to be possible, the panel must be unlocked and the machine must be halted. Also, remember that at the time UNIX comes up, the console switches must contain 773030 for a single-user system; anything else gives a multi-user system.

There are four temperatures of boots. They are:

Hot boot: restart the system without refreshing its code, that is simply by transferring to its start. The only use for this procedure is if the system has been patched and one doesn't wish to redo the patches. The procedure is:

600 in switches
Load address

(773030 in switches for single-user system)
start

Warm boot: refresh system code from the RF disk, but the "panic" routine must be in core. Best for general use if it works, since outstanding I/O is cleaned up. Procedure:

602 in switches
load address
(773030 in switches for single-user system)
start (flushes any I/O, then executes bos)

Cool boot: RF disk is OK, but nothing in core. Procedure:

UTIL DECtape on drive 0
773030 in switches
load address
(602 in switches for multi-user system)
start
type "boot" on console tty to load bos

Cold boot: nothing in core, nothing on RF. Best to have an expert around for this one. Procedure:

INIT DECtape on drive 0
773030 in switches
load address
1 in switches
start
(machine halts. last chance to preserve RF!)
773030 in switches
continue
(reads in basic files)

UNIX is then up, but for various reasons, one should do a warm boot (single user) right away. At this point also, one might consider whether the INIT tape UNIX is the latest version. If there is reason for doubt, mount the /sys disk pack, change to directory /sys/sys, do "msys u unix", and reboot. Then get the /bin-/etc-/lib tape which contains the rest of of the RF disk, and do an "mt x". Conceivably, "create errors" due to lack of some directories will occur; make the directories, then try again. Set the date correctly; the system starts off at time 0.

At this point UNIX is in full operation and can be rebooted for a multi-user system.

Here is what happens during a cold boot: the INIT tape contains a program called vcboot. The ROM program reads vcboot from the tape into core location 0 and transfers to it. vcboot then reads 16K words from the DECtape (blocks 1-32) and copies the data to the highest 16K

words of the RF. Thus this initializes the read-only part of the RF. vcboot then reads in bos and executes it. bos reads in Cold UNIX and executes that. Cold UNIX halts for a last chance before it completely initializes the RF file system. When continue is pressed, Cold UNIX initializes the RF. It then reads the DECtape for initialization files starting from block 33. Normal operation then commences with the execution of "/etc/init".

The INIT tape is made by the program maki running under UNIX. maki writes vcboot on block 0 of /dev/tap7. It then copies the RF 16K words (using /dev/rf0) onto blocks 1 thru 32. It has internally a list of files to be copied from block 33 on. This list follows:

```
/etc/init
/bin/chmod
/bin/date
/bin/login
/bin/ls
/bin/mkdir
/etc/mount
/bin/sh
/bin/tap
/bin/mt
```

Thus this is the set of programs available after a cold boot. init and sh are mandatory. For multi-user UNIX, getty and login are also necessary. mkdir is necessary due to a bug in tap. mt, tap and mount are useful to bring in new files. As soon as possible, date should be done. That leaves ls and chmod as frosting.

The last link in this incestuous daisy chain is the program msys.

```
msys char file
```

will copy the file file onto the RF read only slot specified by the character char. Char is taken from the following set:

```
b bos
u Warm UNIX
1 Cold UNIX
```

FILES /dev/rf0, /dev/tap?

SEE ALSO init(VII), tap(I), sh(I), mkdir(I)

DIAGNOSTICS --

BUGS This section is very configuration dependent.

NAME

check -- file system consistency check

SYNOPSIS

check [filesystem [$blockno_1$ ...]]

DESCRIPTION

check will examine a file system, build a bit map of used blocks, and compare this bit map against the bit map maintained on the file system. If the file system is not specified, a check of all of the normally mounted file systems is performed. Output includes the number of files on the file system, the number of these that are 'large', the number of indirect blocks, the number of used blocks, and the number of free blocks.

check works by examining the i-nodes on the file system and is entirely independent of its directory hierarchy. The file system may be, but need not be, mounted.

FILES

/dev/rf?, /dev/rk?, /dev/rp?

SEE ALSO

find(I), ds(I)

DIAGNOSTICS

Diagnostics are produced for blocks missing, duplicated, and bad block addresses. Diagnostics are also produced for block numbers passed as parameters. In each case, the block number, i-number, and block class (i = inode, x indirect, f free) is printed.

BUGS

The checking process is two pass in nature. If checking is done on an active file system, extraneous diagnostics may occur.

NAME chk -- check + dcheck

SYNOPSIS chk

DESCRIPTION This command file does a check and a dcheck of all of the normally mounted file systems.

FILES /dev/[fkp]*

SEE ALSO check (VIII), dcheck (VIII)

DIAGNOSTICS see "SEE ALSO"

NAME clri -- clear i-node

SYNOPSIS clri i-number [file system]

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 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.

DIAGNOSTICS "error"

NAME dcheck -- directory consistency check

SYNOPSIS dcheck [-l] [device]

DESCRIPTION dcheck builds an image of the directory hierarchy of the specified device by reading all its directories (using physical I/O guided by the i-nodes on the device). A list entry is made for each file encountered. A second pass reads the i-nodes and for each file compares the number of links specified in its i-node with the number of entries actually seen. All discrepancies are noted.

If no device is specified, a default device is assumed.

The argument -l causes a complete listing of the file names on the device in i-node order.

FILES /dev/rk?

SEE ALSO check(VIII)

DIAGNOSTICS inconsistent i-numbers, unnamed files, unreachable files, loops in directory "hierarchy".

BUGS Unreachable files and loops are discovered only under the "-l" option.

NAME dli -- load DEC binary paper tapes

SYNOPSIS dli output [input]

DESCRIPTION dli will load a DEC binary paper tape into the output file. The binary format paper tape is read from the input file (/dev/ppt is default.)

FILES /dev/ppt

SEE ALSO --

DIAGNOSTICS "checksum"

BUGS --

NAME istat -- get inode status

SYNOPSIS istat [filesystem] $inumber_1$ ...

DESCRIPTION istat gives information about one or more i-nodes on the given file system or on /dev/rk0 if no file system is given.

The information is in exactly the same form as that for stat(I), except that mode letter "a" is used to indicate that the i-node is allocated, "u" that it is unallocated.

FILES /etc/uids, /dev/rk0

SEE ALSO stat(I), ls(I) (-l option)

DIAGNOSTICS --

BUGS istat ignores any read error and pretends to give status even if the file system is not physically present.

NAME kill -- terminate process with extreme prejudice

SYNOPSIS /usr/adm/kill processnumber

DESCRIPTION After ps (q.v.) has given you the unique ID of a process, you can terminate it by this command. A core image is produced in the process's working directory.

Only the super-user can exercise this privilege.

FILES --

SEE ALSO ps (VIII)

DIAGNOSTICS yes

BUGS If the process has executed sys nice (II) and there is another process which has not, but which loops, the first process cannot be done in properly, since it has to be swapped in so as cooperate in its own murder.

It would also be nice if ordinary people could kill their own processes.

NAME mount -- mount file system

SYNOPSIS /etc/mount special file

DESCRIPTION mount announces to the system that a removable file system is present on the device corresponding 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.

FILES --

SEE ALSO umount(VIII)

DIAGNOSTICS "?", if the special file is already in use, cannot be read, or if file does not exist.

BUGS Should be usable only by the super-user. Mounting file systems full of garbage can crash the system.

NAME ps -- process status

SYNOPSIS /usr/adm/ps [-xlt]

DESCRIPTION

ps prints certain facts about active processes. The information is columnar and consists of:

The (numerical) ID of the user associated with the process;

The last character of the control typewriter of the process or "x" if there is no control typewriter; "x" lines are suppressed unless the "x" option is given.

The number of 512-byte disk blocks holding the core image of the process;

The process's unique ID (only with "l" option)

The number of hours (mod 100) and minutes of system, disk, and user-process time accumulated by the process and all its terminated descendants (only with "t" option)

An educated guess as to the command line which caused the process to be created.

Some caveats:

The guess as to the command name and arguments is obtained by examining the process's stack. The process is entitled to destroy this information. Also, only processes whose core images are on disk have visible names. The ps command in particular does not, nor does any other process which happens to be in core at the same time. ps tries to overcome this limitation by spawning a subprocess designed to take up the other core slot, and is usually successful. Because ps examines a dynamically changing data structure, it can produce incorrect results, for example if a process's core image moves between the time ps gets its disk address and reads its stack.

Besides its utility for simple spying, ps is the only plausible way to find the process number of someone you are trying to kill (VIII).

FILES /dev/rf0, /sys/sys/unix (to get magic numbers).

SEE ALSO kill (VIII)

DIAGNOSTICS "Bad RF", if a bad swap address turns up; various missing-file diagnostics.

BUGS As described.

NAME salv -- file system salvage

SYNOPSIS /etc/salv filesystem [-akfs]

DESCRIPTION

salv will place a given file system in a consistent state with almost no loss of information. This is the first step in putting things together after a bad crash. Salv performs the following functions:

A valid free list is constructed.

The previous step is always performed; the following steps are performed only if the "a" option is given. If the file system's only defect is missing blocks, "a" should not be specified.

All bad pointers in the file system are zeroed.

All duplicate pointers to the same block are resolved by changing one of the pointers to point at a new block containing a copy of the data.

Inodes (not directory entries) for special files are generated (mode 16).

Files whose size is too large for the number of blocks they contain (after bad pointers are zeroed) have their size revised downward.

The file system should be unmounted while it is being salvaged. In cases of extreme need the permanently mounted file system may be salvaged; in such a case the system must be rebooted before it has a chance to write out the old, bad super-block.

The "k", "f", and "s" options tell salv what magic numbers to use to generate the size of the free list and the i-node map. "k" is default (RK disk); "f" is RF; "s" is RK with swap space on it. If salv is to be used away from the mother system its code should be checked to verify the numbers.

After a salv, files may be safely created and removed without causing more trouble. If the "a" option had to be used, a dcheck (VIII) should be done to find the degree of the damage to the hierarchy.

SEE ALSO check(VIII), dcheck(VIII)

BUGS In only one (known) way does salv destroy information: if some random block appears to be an indirect block for a file, all "bad pointers" (for example, ASCII text) in it will be zeroed. If the block also appears in another file, it may be scribbled on before it is copied.

NAME su -- become privileged user

SYNOPSIS su

DESCRIPTION su allows one to become the super-user, who has all sorts of marvelous (and correspondingly dangerous) powers. In order for su to do its magic, the user must supply a password. If the password is correct, su will execute the shell with the UID set to that of the super-user. To restore normal UID privileges, type an end-of-file to the super-user shell.

To remind the super-user of his responsibilities, the shell substitutes "#" for its usual prompt "%".

FILES --

SEE ALSO sh(I)

DIAGNOSTICS "Sorry" if password is wrong

BUGS --

NAME swtmp -- update accounting file

SYNOPSIS swtmp

DESCRIPTION This shell sequence concatenates /tmp/wtmp onto /usr/adm/wtmp and truncates /tmp/wtmp. It should be used before using acct(VIII) and every so often in any case if accounting is to be maintained.

FILES /tmp/wtmp, /usr/adm/wtmp

SEE ALSO acct(VIII), wtmp(V)

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.

The user must take care not only that all I/O activity on the file system has ceased, but that no one has his current directory on it.

Only the super-user may issue this command.

FILES --

SEE ALSO mount(VIII)

DIAGNOSTICS "?"

BUGS This command is not, in fact, restricted to the super-user.

NAME tm -- provide time information

SYNOPSIS tm

DESCRIPTION tm is used to provide timing information. Output like the following is given:

```
tim 371:51:09     2:00.8
ovh  20:00:33      17.0
swp  13:43:20       4.6
dsk  27:14:35       4.5
idl 533:08:03     1:33.3
usr  24:53:50       1.2
der    0,  54     0,   0
```

The first column of numbers gives totals in the named categories since the last time the system was cold-booted; the second column gives the changes since the last time tm was invoked. The top left number is badly truncated and should be ignored. ovh is time spent executing in the system; swp is time waiting for swap I/O; dsk is time spent waiting for file system disk I/O; idl is idle time; usr is user execution time; der is RF disk error count (left number) and RK disk error count (right number).

FILES /dev/rf0 (for absolute times); /tmp/ttmp for differential timing history.

SEE ALSO time(I), file system(V)

DIAGNOSTICS --

BUGS --