UNIX PROGRAMMER'S MANUAL

Second Edition

K. Thompson

D. M. Ritchie

June 12, 1972

June 12, 1972

PREFACE
to the Second Edition

In the months since this manual first appeared, 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, and J. F. Ossanna for their contributions.

Finally, the number of UNIX installations has grown to 10, 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. One area to watch concerns commands which deal with special files (I/O devices). Another is places which talk about such things as absolute core locations which are likely to vary with the memory configuration and existence of protection hardware. Also, not all installations have the latest versions of all the software. In particular, the assembler and loader have just undergone major reorganizations in anticipation of a UNIX for the PDP-11/45.

INTRODUCTION

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 Inevitably, this means that many sections will soon be out of date.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 seven sections:

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

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 /usr/lib/liba.a.

The special files section IV discusses the characteristics of each system "file" which actually refers to an I/O device.

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.

Each section consists of a number of independent entries of a page or so each. The name of the entry is in the upper right corner of its pages, its preparation date in the upper left. Entries within each section are alphabetized. It was thought better to avoid running page numbers, since it is hoped that the manual will be updated frequently. Therefore each entry is numbered starting at page 1.

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.

The _owner_ section gives the name of the person or persons to be consulted in case of difficulty. The rule has been that the last one to modify something owns it, so the owner is not necessarily the author. The owner's nicknames stand for:

ken	K. Thompson
dmr	D. M. Ritchie
jfo	J. F. Ossanna
rhm	R. Morris
doug	M. D. McIlroy
lem	L. E. McMahon
llc	L. L. Cherry
csr	C. S. Roberts

These nicknames also happen to be UNIX user ID's, so messages may be transmitted by the _mail_ command or, if the addressee is logged in, by _write_.

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_.

TABLE OF CONTENTS

I. COMMANDS

	:(I): place label
	a.out(V): assembler and loader output
chmod(I): change	access mode of files
wtmp(V):	accounting files
acct(I): get connect-time	accounting
tacct(I): connect-time	accounting
	acct(I): get connect-time accounting
dn0(IV): 801	ACU
getty(VII):	adapt to typewriter
salloc(III): storage	allocator
mail(I): send mail to	another user
write(I): write to	another user
ar(I):	archive (combine) files
archive(V):	archive file
	archive(V): archive file
atan(III):	arctangent
glob(VII):	argument expander
echo(I): print command	arguments
	ar(I): archive (combine) files
sort(I): sort	ASCII file
dpt(VI): read DEC	ASCII paper tapes
atof(III): convert	ASCII to floating
atoi(III): convert	ASCII to integer
	ascii(VII): map of ASCII
ascii(VII): map of	ASCII
ctime(III): convert time to	ASCII
convert floating to	ASCII...ftoa(III):
itoa(III): convert integer to	ASCII
	as(I): assembler
a.out(V):	assembler and loader output
as(I):	assembler
sync(II):	assure synchronization
	atan(III): arctangent
	atof(III): convert ASCII to floating
	atoi(III): convert ASCII to integer
bc(VI): compile	B program
log(III): logarithm	base e
	bas(I): BASIC dialect
bas(I):	BASIC dialect
	basic(VI): DEC supplied BASIC
basic(VI): DEC supplied	BASIC
	bc(VI): compile B program
su(I):	become super-user
dli(VI): load DEC	binary paper tapes
remove symbols, relocation	bits...strip(I):
	bj(VI): the game of black jack
bj(VI): the game of	black jack
bproc(VII):	boot procedure
	bproc(VII): boot procedure
	break(II): set program break
break(II): set program	break
istat(I): file status	by i-number

atof(III):	convert ASCII to floating
atoi(III):	convert ASCII to integer
ftoa(III):	convert floating to ASCII
itoa(III):	convert integer to ASCII
ctime(III):	convert time to ASCII
cp(I):	copy file
core(V):	core image file
mem(IV):	core memory
	core(V): core image file
sin(III): sine,	cosine
wc(I): get (English) word	count
	cp(I): copy file
makdir(II):	create directory
mkdir(I):	create directory
creat(II):	create file
fork(II):	create new process
	creat(II): create file
	cref(VI): cross-reference table
cref(VI):	cross-reference table
	ctime(III): convert time to ASCII
dpd(I): spawn data-phone	daemon
salv(I): repair	damaged file system
	das(VI): disassembler
dpd(I): spawn	data-phone daemon
dp0(IV): 201	Dataphone
date(I): get	date and time of day
mdate(II): set	date modified of file
	date(I): get date and time of day
date(I): get date and time of	day
	db(I): symbolic debugger
	dc(I): desk calculator
db(I): symbolic	debugger
dpt(VI): read	DEC ASCII paper tapes
dli(VI): load	DEC binary paper tapes
basic(VI):	DEC supplied BASIC
tap(V):	DECtape format
rew(I): rewind	DECtape
tap(I): manipulate	DECtape
tap0(IV):	DECtape
sleep(II):	delay execution
rmdir(I): remove	(delete) directory
dsw(I):	delete files interactively
rm(I): remove	(delete) file
unlink(II): remove	(delete) file
mesg(I): permit or	deny messages
switch(III): transfer	depending on value
dc(I):	desk calculator
kill(II):	destroy process
mount(I): mount	detachable file system
	df(I): find free disk space
bas(I): BASIC	dialect
directory(V):	directory format
ds(I): verify	directory hierarchy
	directory(V): directory format
chdir(I): change working	directory
chdir(II): change working	directory

	istat(I): file status by i-number
	itoa(III): convert integer to ASCII
bj(VI): the game of black	jack
	kbd(VII): map of TTY 37 keyboard
kbd(VII): map of TTY 37	keyboard
	kill(II): destroy process
:(I): place	label
	ld(I): link editor (loader)
form(I): generate form	letter
lpr(IV):	line printer
ld(I):	link editor (loader)
link(II):	link to file
ln(I):	link to file
	link(II): link to file
ls(I):	list contents of directory
nlist(III): read name	list
	ln(I): link to file
dli(VI):	load DEC binary paper tapes
a.out(V): assembler and	loader output
ld(I): link editor	(loader)
login(I):	log on to system
login(VII): how to	log onto system
log(III):	logarithm base e
utmp(V):	logged-in user information
	log(III): logarithm base e
	login(I): log on to system
	login(VII): how to log onto system
hog(II): set	low-priority status
	lpr(IV): line printer
	ls(I): list contents of directory
m6(I):	macroprocessor
mt(I): save/restore files on	magtape
mt0(IV):	magtape
mail(I): send	mail to another user
	mail(I): send mail to another user
	makdir(II): create directory
	man(I): run off manual section
tap(I):	manipulate DECtape
man(I): run off	manual section
uids(V):	map names to user ID's
ascii(VII):	map of ASCII
kbd(VII):	map of TTY 37 keyboard
	mdate(II): set date modified of file
	mem(IV): core memory
mem(IV): core	memory
	mesg(I): permit or deny messages
	mesg(III): print string on typewriter
mesg(I): permit or deny	messages
tss(I): communicate with	MH-TSS (GCOS)
msh(VII):	mini Shell
	mkdir(I): create directory
chmod(I): change access	mode of files
chmod(II): change	mode of file
stty(II): set	mode of typewriter
stty(I): set typewriter	modes
gtty(II): get typewriter	mode

mdate(II): set date	modified of file
	moo(VI): the game of MOO
moo(VI): the game of	MOO
mount(I):	mount detachable file system
mount(II):	mount file system
	mount(I): mount detachable file system
	mount(II): mount file system
mv(I):	move or rename file
seek(II):	move read or write pointer
	msh(VII): mini Shell
	mt(I): save/restore files on magtape
	mt0(IV): magtape
	mv(I): move or rename file
	m6(I): macroprocessor
nlist(III): read	name list
tty(I): find	name of terminal
nm(I): print	namelist
uids(V): map	names to user ID's
find(I): find file with given	name
fork(II): create	new process
	nlist(III): read name list
	nm(I): print namelist
	nroff(I): format text for printing
od(I):	octal dump of file
	od(I): octal dump of file
man(I): run	off manual section
opr(I): print file	off-line
login(VII): how to log	onto system
close(II): close	open file
fstat(II): status of	open file
open(II):	open file
	open(II): open file
	opr(I): print file off-line
cat(I): concatenate	(or print) files
assembler and loader	output...a.out(V):
ov(I): page	overlay file print
	ov(I): page overlay file print
chown(I): change	owner of files
chown(II): change	owner of file
ov(I):	page overlay file print
type(I): print file	page-by-page
dli(VI): load DEC binary	paper tapes
dpt(VI): read DEC ASCII	paper tapes
ppt(IV): punched	paper tape
	passwd(V): password file
passwd(V):	password file
mesg(I):	permit or deny messages
ptx(VI):	permuted index
:(I):	place label
seek(II): move read or write	pointer
tell(II): find read or write	pointer
	ppt(IV): punched paper tape
chash(VI):	prepare symbol table
	pr(I): print file with headings
cal(VI):	print calendar
echo(I):	print command arguments

	roff(I): format text for printing
sqrt(III): square	root
rp0(IV):	RP disk
	rp0(IV): RP disk
man(I):	run off manual section
	salloc(III): storage allocator
	salv(I): repair damaged file system
mt(I):	save/restore files on magtape
man(I): run off manual	section
	seek(II): move read or write pointer
mail(I):	send mail to another user
exit(I): end command	sequence
mdate(II):	set date modified of file
hog(II):	set low-priority status
stty(II):	set mode of typewriter
break(II):	set program break
stime(II):	set system time
tabs(VII):	set tab stops on typewriter
stty(I):	set typewriter modes
setuid(II):	set user ID
	setuid(II): set user ID
msh(VII): mini	Shell
	sh(I): command interpreter
fptrap(III): floating-point	simulator
sin(III):	sine, cosine
	sin(III): sine, cosine
	sleep(II): delay execution
sort(I):	sort ASCII file
	sort(I): sort ASCII file
qsort(III): quicker	sort
df(I): find free disk	space
dpd(I):	spawn data-phone daemon
	sqrt(III): square root
sqrt(III):	square root
	stat(I): get file status
	stat(II): get file status
istat(I): file	status by i-number
fstat(II):	status of open file
hog(II): set low-priority	status
stat(I): get file	status
stat(II): get file	status
	stime(II): set system time
tabs(VII): set tab	stops on typewriter
salloc(III):	storage allocator
mesg(III): print	string on typewriter
	strip(I): remove symbols, relocation bits
	stty(I): set typewriter modes
	stty(II): set mode of typewriter
	su(I): become super-user
sum(I):	sum file
	sum(I): sum file
su(I): become	super-user
basic(VI): DEC	supplied BASIC
	switch(III): transfer depending on value
chash(VI): prepare	symbol table
db(I):	symbolic debugger

gtty(II): get	typewriter mode
getty(VII): adapt to	typewriter
mesg(III): print string on	typewriter
stty(II): set mode of	typewriter
tabs(VII): set tab stops on	typewriter
tty(IV): console	typewriter
tty0(IV): remote	typewriter
	uids(V): map names to user ID's
	umount(I): dismount removable file system
	umount(II): dismount file system
un(I): find	undefined symbols
	un(I): find undefined symbols
	unlink(II): remove (delete) file
du(I): find disk	usage
uids(V): map names to	user ID's
getuid(II): get	user ID
setuid(II): set	user ID
utmp(V): logged-in	user information
mail(I): send mail to another	user
write(I): write to another	user
	utmp(V): logged-in user information
transfer depending on	value...switch(III):
ds(I):	verify directory hierarchy
wait(II):	wait for process
	wait(II): wait for process
	wc(I): get (English) word count
who(I):	who is on the system
	who(I): who is on the system
gerts(III): communicate	with GCOS
find(I): find file	with given name
pr(I): print file	with headings
tss(I): communicate	with MH-TSS (GCOS)
wc(I): get (English)	word count
putc(III): write character or	word
chdir(I): change	working directory
chdir(II): change	working directory
putc(III):	write character or word
write(II):	write file
seek(II): move read or	write pointer
tell(II): find read or	write pointer
write(I):	write to another user
	write(I): write to another user
	write(II): write file
	wtmp(V): accounting files
time(II): get time of	year
dn0(IV):	801 ACU
dp0(IV):	201 Dataphone
kbd(VII): map of TTY	37 keyboard

NAME	: -- place a label
SYNOPSIS	_:_ [label]
DESCRIPTION	_:_ does nothing. Its only function is to place a label for the _goto_ command. _:_ is a command so the Shell doesn't have to be fixed to ignore lines with :'s.
FILES	--
SEE ALSO	goto(I)
DIAGNOSTICS	--
BUGS	--
OWNER	dmr

NAME	acct -- login accounting
SYNOPSIS	_acct_ [wtmp]
DESCRIPTION	_acct_ produces a printout giving connect time and total number of connects for each user who has logged in during the life of the current _wtmp_ file. A total is also produced. If no wtmp file is given, _/tmp/wtmp_ is used.
FILES	/tmp/wtmp
SEE ALSO	init(VII), tacct(I), login(I), wtmp(V).
DIAGNOSTICS	"Cannot open 'wtmp'" if argument is unreadable.
BUGS	--
OWNER	dmr, ken

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.
OWNER	ken, dmr

NAME as -- assembler

SYNOPSIS *as* [*-*] name$_1$ ...

DESCRIPTION *as* assembles the concatenation of name$_1$, ... *as* is based on the DEC-provided assembler PAL-11R [1], although it was coded locally. Therefore, only the differences will be recorded.

If the optional first argument *-* is used, all undefined symbols in the assembly are treated as global.

Character changes are:

for	use
@	*
#	$
;	/

In *as*, the character ";" is a logical new line; several operations may appear on one line if separated by ";". Several new expression operators have been provided:

\>	right shift (logical)
\<	left shift
*	multiplication
\/	division
%	remainder (no longer means "register")
!	one's complement
[]	parentheses for grouping
^	result has value of left, type of right

For example location 0 (relocatable) can be written "0^."; another way to denote register 2 is "2^r0".

All of the preceding operators are binary; if a left operand is missing, it is taken to be 0. The "!" operator adds its left operand to the one's complement of its right operand.

There is a conditional assembly operation code different from that of PAL-11R (whose conditionals are not provided):

```
.if expression
...
.endif
```

If the *expression* evaluates to non-zero, the section of code between the ".if" and the ".endif" is assembled; otherwise it is ignored. ".if"s may be nested.

Temporary labels like those introduced by Knuth [2] may be employed. A temporary label is defined as follows:

```
     n:
```

where _n_ is a digit 0 ... 9. Symbols of the form "_n_f" refer to the first label "_n_:" following the use of the symbol; those of the form "_n_b" refer to the last "_n_:". The same "_n_" may be used many times. Labels of this form are less taxing both on the imagination of the programmer and on the symbol table space of the assembler.

The PAL-11R opcodes ".word", ".eot" and ".end" are redundant and are omitted.

The symbols

```
     r0 ... r5
     fr0 ... fr5 (floating-point registers)
     sp
     pc
     ac
     mq
     div
     mul
     lsh
     ash
     nor
     csw
     ..
```

are predefined with appropriate values. The symbol "csw" refers to the console switches. ".." is the relocation constant and is added to each relocatable reference. On a PDP-11 with relocation hardware, its value is 0; on most systems without protection, its value is 40000(8).

The new opcode "sys" is used to specify system calls. Names for system calls are predefined. See section (II).

The opcodes "bes" (branch on error set) and "bec" (branch on error clear) are defined to test the error status bit set on return from system calls.

Strings of characters may be assembled in a way more convenient than PAL-11's ".ascii" operation (which is, therefore, omitted). Strings are included between the string quotes "<" and ">":

```
     <here is a string>
```

Escape sequences exist to enter non graphic and other difficult characters. These sequences are also effective in single and double character constants introduced by single (') and double (") quotes respectively.

<u>use</u>	<u>for</u>
\n	newline (012)
\0	NULL (000)
\>	>
\t	TAB (011)
\a	ACK (006)
\r	CR (015)
\p	ESC (033)
\\	\ (134)

<u>as</u> provides a primitive segmentation facility. There are three segments: <u>text</u>, <u>data</u> and <u>bss</u>. The text segment is ordinarily used for code. The data segment is provided for initialized but variable data. The bss segment cannot be initialized, but symbols may be defined to lie within this segment. In the future, it is expected that the text segment will be write-protected and sharable. Assembly begins in the text segment. The pseudo-operations

```
.text
.data
.bss
```

cause the assembler to switch to the text, data, or bss segment respectively. Segmentation is useful at present for two reasons: Non-initialized tables and variables, if placed in the bss segment, occupy no space in the output file. Also, alternative use of the text and data segments provides a primitive dual location-counter feature.

In the output file, all text-segment information comes first, followed by all data-segment information, and finally bss information. Within each segment, information appears in the order written.

Note: since nothing explicit can be assembled into the bss segment, the usual appearance of this segment is in the following style:

```
.bss
var1:   .=.+2
tab1:   .=.+100.
...
```

That is, space is reserved but nothing explicit is placed in it.

As is evident from the example, it is legal to assign to the location counter ".". It is also permissible in segments other than ".bss". The restriction is made, however, that the value so assigned must be defined in the first pass and it must be a value associated with the same segment as ".".

The pseudo-op

```
        .comm   symbol,expression
```

makes symbol an undefined global symbol, and places the value of the expression in the value field of the symbol's definition. Thus the above declaration is equivalent to

```
        .globl  symbol
        symbol = expression ^ symbol
```

The treatment of such a symbol by the loader ld(I) is as follows: If another routine in the same load defines the symbol to be an ordinary text, data, bss, or absolute symbol, that definition takes precedence and the symbol acts like a normal undefined external. If however no other routine defines the symbol, the loader defines it as an external bss-segment symbol and reserves _n_ bytes after its location, where _n_ is the value of the expression in the .comm operation. Thus ".comm x,100" effectively declares x to be a common region 100 bytes long. Note: all such declarations for the same symbol in various routines should request the same amount of space.

The binary output of the assembler is placed on the file "a.out" in the current directory. _a.out_ also contains the symbol table from the assembly and relocation bits. The output of the assembler is executable immediately if the assembly was error-free and if there were no unresolved external references. The link editor _ld_ may be used to combine several assembly outputs and resolve global symbols.

The assembler does not produce a listing of the source program. This is not a serious drawback; the debugger _db_ discussed below is sufficiently powerful to render a printed octal translation of the source unnecessary.

On the last pages of this section is a list of all the assembler's built-in symbols. In the case of instructions, the addressing modes are as follows:

src, dst	source, destination
r	general register
fsrc,fdst	floating source, destination
fr	floating register
exp	expression

The names of certain 11/45 opcodes are different from those in the 11/45 manual; some were changed to avoid conflict with EAE register names, others to draw analogies with existing 11/20 instructions.

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), sh(I), un(I), db(I), a.out(V), fptrap(III), [1] PAL-11R Assembler; DEC-11-ASDB-D, [2] Knuth, _The Art of Computer Programming_, Vol. I; Fundamental Algorithms.

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.

If "." is moved backwards by an odd number of bytes, relocation bits are corrupted.

OWNER

dmr

Special variables:

```
    .
    ..
```

Register:

```
    r0
    r1
    r2
    r3
    r4
    r5
    sp
    pc
    fr0
    fr1
    fr2
    fr3
    fr4
    fr5
```

Eae & switches:

```
    csw
    div
    ac
    mq
    mul
    sc
    sr
    nor
    lsh
    ash
```

System calls:

```
    exit
    fork
    read
    write
    open
    close
    wait
    creat
    link
    unlink
    exec
    chdir
    time
    makdir
    chmod
    chown
    break
    stat
    seek
    tell
    mount
    umount
    setuid
    getuid
    stime
    quit
    intr
    fstat
    cemt
    mdate
    stty
    gtty
    ilgins
    hog
```

Double operand:

```
    mov       src,dst
    movb        "
    cmp         "
    cmpb        "
    bit         "
    bitb        "
    bic         "
    bicb        "
    bis         "
    bisb        "
    add         "
    sub         "
```

Branch:

```
    br
    bne
    beq
    bge
    blt
    bgt
    ble
    bpl
    bmi
    bhi
    blos
    bvc
    bvs
    bhis
    bec       (= bcc)
    bcc
    blo
    bcs
    bes       (= bcs)
```

Single operand:

clr	dst	
clrb	"	
com	"	
comb	"	
inc	"	
incb	"	
dec	"	
decb	"	
neg	"	
negb	"	
adc	"	
adcb	"	
sbc	"	
sbcb	"	
ror	"	
rorb	"	
rol	"	
rolb	"	
asr	"	
asrb	"	
asl	"	
aslb	"	
jmp	"	
swab	"	
tst	src	
tstb	src	

Miscellaneous:

jsr	r,dst	
rts	r	
sys	exp	(= trap)

Flag-setting:

clc
clv
clz
cln
sec
sev
sez
sen

Floating point ops:

cfcc		
setf		
setd		
seti		
setl		
clrf	fdst	
negf	fdst	
absf	fdst	
tstf	fsrc	
movf	fsrc,fr	(= ldf)
movf	fr,fdst	(= stf)
movif	src,fr	(= ldcif)
movfi	fr,dst	(= stcfi)
movof	fsrc,fr	(= ldcdf)
movfo	fr,fdst	(= stcfd)
addf	fsrc,fr	
subf	fsrc,fr	
mulf	fscr,fr	
divf	fsrc,fr	
cmpf	fsrc,fr	
modf	fsrc,fr	

11/45 operations

als	src,r	(= ash)
alsc	src,r	(= ashc)
mpy	src,r	(= mul)
dvd	src,r	(= div)
xor	src,r	
sxt	dst	
mark	exp	
sob	r,exp	

Specials

.byte
.even
.if
.endif
.globl
.text
.data
.bss
.comm

NAME bas -- basic

SYNOPSIS <u>bas</u> [file]

DESCRIPTION <u>bas</u> is a dialect of basic [1]. If a file argument is provided, the file is used for input before the console is read.

<u>bas</u> 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: (<u>expr</u> is short for expression)

expr
The expression is executed for its side effects (assignment or function call) or for printing as described above.

<u>done</u>
Return to system level.

<u>for</u> name <u>=</u> expr expr statement
<u>for</u> name <u>=</u> expr expr
...
<u>next</u>
The <u>for</u> 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.

<u>goto</u> expr
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.

<u>if</u> expr statement
The statement is executed if the expression evaluates to non-zero.

<u>list</u> [expr [expr]]

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 expr
The expression is evaluated and printed.

return expr
The expression is evaluated and the result is passed back as the value of a function call.

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.

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

expr op expr
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.

expr ([expr [, expr ...]])
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 [expr [, expr ...]]
Arrays are not yet implemented.

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 ith 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. (Not implemented.)

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. All other diagnostics are self explanatory.

BUGS   Arrays [] are not yet implemented. In general, program sizes, recursion, etc are not checked, and cause trouble.

OWNER   ken

NAME

cat -- concatenate and print

SYNOPSIS

cat file$_1$ ...

DESCRIPTION

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

FILES

--

SEE ALSO

pr(I), cp(I)

DIAGNOSTICS

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

BUGS

--

OWNER

ken, dmr

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
a.out	loaded output
c.tmp	temporary (deleted)
/sys/c/nc	compiler
/usr/lib/crt0.o	runtime startoff
/usr/lib/libc.a	builtin functions, etc.
/usr/lib/liba.a	system library

SEE ALSO C reference manual (in preparation), bc(VI)

DIAGNOSTICS Diagnostics are intended to be self-explanatory.

BUGS --

OWNER dmr

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	--
OWNER	ken, dmr

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 used blocks, and the number of free blocks.
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. The swap space on the RF file system is not accounted for and will therefore show up as 'missing'.
OWNER	ken, dmr

NAME	chmod -- change mode
SYNOPSIS	<u>chmod</u> 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	--
OWNER	ken, dmr

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 name found in /etc/uids.

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/uids

SEE ALSO stat(I)

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

BUGS --

OWNER ken, dmr

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.
FILES	--
SEE ALSO	--
DIAGNOSTICS	Messages are given for inability to open either argument, premature EOF on either argument, and incorrect usage.
BUGS	If the two files differ in length by one byte, the extra byte does not enter into the comparison.
OWNER	dmr

NAME cp -- copy

SYNOPSIS *cp* file$_1$ file$_2$

DESCRIPTION The first file is opened for reading, the second created mode 17. Then the first is copied into the second.

FILES --

SEE ALSO cat(I), pr(I)

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

BUGS The second file should be created in the mode of the first.

A directory convention as used in *mv* should be adopted for *cp*.

OWNER ken, dmr

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 --

OWNER dmr

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

4. The symbol "." indicates the current pointer of db. The current pointer is set by many db requests.

5. 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.

6. 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.

7. Expressions are evaluated left to right.

Names for registers are built in:

```
r0 ... r5
sp
pc
ac
mq
```

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 multiplied by 2, then printed in octal (used with B programs, whose addresses are word addresses).

? 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.

It is illegal for the word-oriented commands to have odd addresses. 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 The "^" request always decrements "." by 2, even in byte mode.

OWNER dmr

NAME dc -- desk calculator

SYNOPSIS dc

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. If the number starts with a zero, it is taken to be octal, otherwise it is decimal.

+ - * / %
The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), or remaindered (%). The two entries are poppped off of 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.

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 in decimal. The top value remains unchanged.

f
All values on the stack are popped off and printed in decimal.

q
exits the program

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

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

r
All values on the stack are popped.

n*k*
   A scale factor of 10^n is set for all subsequent multiplication and division.

new-line
space
   ignored.

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)
1a52/     (pennies per week)
d10*      (deci-pennies per week)
375/      (pennies per hour)
f         (print all results)
(3) 512
(2) 19230
(1) 83333
```

FILES --

SEE ALSO --

DIAGNOSTICS
(x) ? for unrecognized character x.
(x) ? for not enough elements on the stack to do what was asked.
"Out of space" when the free list is exhausted.

BUGS
f is not implemented
% is not implemented

OWNER
rhm

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(I)

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

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 the file is prepended with a form feed.

_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 --

OWNER	ken

NAME	ds -- directory consistency check
SYNOPSIS	_ds_ [output]
DESCRIPTION	_ds_ will walk the directory tree from the root keeping a list of every file encountered. The second pass will read the i-list and compare the number of links there with the actual number found. All discrepancies are noted. If an argument is given, a complete printout of file names by i-number is output on the argument.
FILES	/, /dev/rk0, /tmp/dstmp
SEE ALSO	check(I)
DIAGNOSTICS	inconsistent i-numbers
BUGS	the root is noted as inconsistent due to the fact that _/_ exists in no directory. (Its i-number is 41.) _ds_ should take an alternate file system argument.
OWNER	ken

NAME	dsw -- delete interactively
SYNOPSIS	<u>dsw</u> [directory]
DESCRIPTION	For each file in the given directory ("." if not specified) <u>dsw</u> types its name. If "y" is typed, the file is deleted; if "x", <u>dsw</u> 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.
OWNER	dmr, ken

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.
OWNER	dmr

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	--
OWNER	doug

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 an explicit 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 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 <u>ed</u> 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.

(.,.)l

The list command prints the addressed lines in an unambiguous way. Non-printing characters are over-struck as follows:

char	prints
bs	⍀
tab	→
ret	←
SI	Ɨ
SO	ɵ

All characters preceded by a prefix (ESC) character are printed over-struck with ^ without the prefix. Long lines are folded with the sequence \newline.

(.,.)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.

<u>q</u>
The <u>q</u>uit command causes <u>ed</u> to exit. No automatic write of a file is done.

($)r filename
The <u>r</u>ead 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 <u>e</u> and <u>f</u> commands). The remembered file name is not changed unless "filename" is the very first file name mentioned. Address "0" is legal for <u>r</u> 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 <u>s</u>ubstitute 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 <u>g</u>lobal command except that the command list is executed with "." initially set to every line <u>except</u> those matching the regular expression

(1,$)w filename
The <u>w</u>rite 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 <u>not</u> 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

BUGS --

OWNER ken, dmr, jfo

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	--
OWNER	dmr

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 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 ".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.

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
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
29	( ) imbalance in expression
30	expression syntax error
31	illegal variable in equivalence
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
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:

loading of common (a BLOCK DATA program must be written to allocate common).

arithmetic statement functions

data statements

backspace, endfile, rewind runtime

binary I/O

no scale factors on input

OWNER

dmr, ken

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 _e_ and _m_ where literal string names are expected. The commands are:

e $name_1$ ...

_e_dit 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
w
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
q        (get out of ed)
q        (get out of fe)
```

d [$name_1$ ...]

_d_eletes 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 command level. Any other response does nothing.

m $name_1$ $name_2$ ...

(_m_ove) 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. 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	--
OWNER	rhm,llc

NAME find -- find file with given name

SYNOPSIS *find* name or number ...

DESCRIPTION *find* searches the entire file system hierarchy and gives the path names of all files with the specified names or (decimal) i-numbers.

FILES */*

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER dmr

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.
OWNER	rhm,llc

NAME	goto -- command transfer
SYNOPSIS	<u>goto</u> label
DESCRIPTION	<u>goto</u> 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 <u>label</u>. If such a line is found, the <u>goto</u> 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	--
OWNER	dmr

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

_-c_ file
true if the file either exists and is writable, or does not exist and is creatable.

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 "-c" always indicates the file is creatable, even if it isn't.

OWNER dmr

NAME istat -- get inode status

SYNOPSIS <u>istat</u> inumber_1 ...

DESCRIPTION <u>istat</u> gives information about one or more i-nodes on the file system <u>/dev/rk0</u>.

The information is basically in the same for as that for stat(I). All information is self-explanatory except the mode. The mode is a seven-character string whose characters mean the following:

```
1 a: i-node is allocated
  u: i-node is free (no file)
2 s: file is small (smaller than 4096 bytes)
  l: file is large

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

4 r: owner can read
  -: owner cannot read

5 w: owner can write
  -: owner cannot write

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

7 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/uids" a number is given.

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

FILES /etc/uids, /dev/rk0

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

DIAGNOSTICS "name?" for any error.

BUGS <u>istat</u> should take an optional alternate filesystem argument.

OWNER dmr

NAME ld -- link editor

SYNOPSIS _ld_ [_-usaol_] $name_1$ ...

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 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"-- and 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	Instructions in the data segment are not relocated properly.
OWNER	dmr

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.
OWNER	ken, dmr

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 _login_ (VII) 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

SEE ALSO

login(VII), 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

--

OWNER

dmr, ken

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 _l_ong 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/uids to get user ID's for ls -l
SEE ALSO	stat(I)
DIAGNOSTICS	"name nonexistent"; "name unreadable"; "name unstatable."
BUGS	--
OWNER	dmr, ken

NAME

mail -- send mail to another user

SYNOPSIS

mail [letter 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. The answer to the above question may be supplied in the *letter* argument.

When followed by the names of a letter and one or more people, the letter is appended to each person's *mailbox*. Each letter is preceded by the sender's name and a postmark.

A *person* is either the name of an entry in the directory */usr*, in which case the mail is sent to /*usr*/*person*/*mailbox*, 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/uids	to map uids
mailbox	input mail
mbox	saved mail

SEE ALSO

login(I)

DIAGNOSTICS

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

BUGS

--

OWNER

ken

NAME	man -- run off section of UNIX manual
SYNOPSIS	<u>man</u> title [section]
DESCRIPTION	<u>man</u> 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, <u>1</u> 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	--
OWNER	ken

NAME	mesg -- permit or deny messages
SYNOPSIS	<u>mesg</u> [<u>n</u>][<u>y</u>]
DESCRIPTION	<u>mesg</u> <u>n</u> forbids messages via <u>write</u> by revoking non-user write permission on the user's typewriter. <u>mesg</u> <u>y</u> reinstates permission. <u>mesg</u> 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	--
OWNER	dmr, ken

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	--
OWNER	ken, dmr

NAME	mount -- mount file system
SYNOPSIS	_/etc/mount_ special dir
DESCRIPTION	_mount_ announces to the system that a removable file system has been mounted on the device corresponding to special file _special_. Directory _dir_ (which must exist already) becomes the name of the root of the newly mounted file system.
FILES	--
SEE ALSO	umount(I)
DIAGNOSTICS	"?", if the special file is already in use, cannot be read, or if _dir_ does not exist.
BUGS	Should be usable only by the super-user. It is possilbe to mount the same file system pack twice. This is a very efficient way to destroy a pack.
OWNER	ken, dmr

NAME mt -- manipulate magtape

SYNOPSIS <u>mt</u> [key] [name ...]

DESCRIPTION <u>mt</u> saves and restores selected portions of the file system hierarchy on magtape. Its actions are controlled by the <u>key</u> 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 <u>t</u> 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 <u>mt</u> does its work silently. The <u>v</u> (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 <u>v</u> option can be used with <u>r</u> and <u>x</u> 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.

i ignore tape errors. It is suggested that this option be used with caution to read damaged tapes.

FILES /dev/mt?

SEE ALSO tap(I), tap(V)

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 The _m_ option does not work correctly. The _i_ option is not yet implemented.

OWNER ken

NAME	mv -- move or rename a file
SYNOPSIS	*mv* name$_1$ name$_2$
DESCRIPTION	*mv* changes the name of name$_1$ by linking to it under the name name$_2$ and then unlinking name$_1$. If the new name is a directory, the file is moved to that directory under its old name. Directories may only be moved within the same parent directory (just renamed).
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	Since *mv* is implemented by combinations of *link* and *unlink*, it cannot be used to move between file systems.
OWNER	ken, dmr

NAME m6 — general purpose macro processor

SYNOPSIS <u>m6</u> [<u>-d</u> arg1] [arg2 [arg3]]

DESCRIPTION <u>m6</u> 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 <u>m.def</u>.

#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 bultins 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
/sys/lang/mdir/m6a--m6 processor proper (/bin/m6 is only an initializer)
/sys/lang/mdir/m6b--default initialization for

m.def

SEE ALSO	[1] M6 reference
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 "ovc" -- overflow of nested calls "ova" -- overflow of nested arguments "ovd" -- overflow of definitions "rdd" -- can't read definition table "wrd" -- can't write definition table, either on #save: or on garbage collection
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: Again to save space, garbage collection makes calls on #save: and #rest: and so overwrites m.def.
OWNER	doug

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	--
OWNER	dmr, ken

NAME	nroff -- format text
SYNOPSIS	_nroff_ [$\pm$number1] [$\underline{-}$number2] [_-stop_] name_1 ...
DESCRIPTION	_nroff_ formats text according to control lines embedded in the text in files name_1, ... in a manner similar to roff(I). _nroff_ permits wider page layout flexibility that roff; examples include arbitrary format and length for page headings and footings, page shaping, some footnote capability, and double column output (with the aid of a postprocessor, ov(I)). Encountering a nonexistent file terminates printing. The optional argument "$\pm$number1" causes printing to begin at the first page numbered number1; the optional argument "$\underline{-}$number2" stops printing after the page numbered number2. The optional argument "_-stop_" or "_-s_" causes printing to stop before each page including the first to allow paper manipulation; printing is resumed upon receipt of an interrupt signal. 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. _nroff_ is described in a separate publication [1].
FILES	/etc/suftab suffix hyphenation tables rtm? temporary
SEE ALSO	[1] (See J. F. Ossanna)
DIAGNOSTICS	none
BUGS	--
OWNER	jfo

NAME od -- octal dump

SYNOPSIS <u>od</u> name [origin]

DESCRIPTION <u>od</u> dumps a file in octal, eight words per line with the origin of the line on the left. If an octal origin is given it is truncated to 0 mod 16 and dumping starts from there, otherwise from 0. Printing continues until an end-of-file condition or until halted by sending an interrupt signal.

Since <u>od</u> does not seek, but reads to the desired starting point, <u>od</u> (rather than <u>db</u>) should be used to dump special files.

FILES --

SEE ALSO db(I)

DIAGNOSTICS "?"

BUGS --

OWNER ken, dmr

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, 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 *+* then *opr* will make a copy for the daemon to print. If the file argument is preceded by *-* then opr will unlink the file.

FILES

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

SEE ALSO dpd(I), ident(V)

DIAGNOSTICS --

BUGS

Since all but the *+* option in *opr* is implemented with *links*, one cannot use these options for files not in */usr*.

opr should recognize *+* and *-* alone and apply them to all subsequent arguments.

OWNER ken

NAME	ov -- overlay pages
SYNOPSIS	*ov* filename
DESCRIPTION	*ov* is a postprocessor for producing double column formatted text when using nroff(I). *ov* assumes that the named file contains an even number of 66 line pages and literally overlays successive pairs of pages.
FILES	none
SEE ALSO	nroff(I)
DIAGNOSTICS	none
BUGS	Other page lengths should be permitted.
OWNER	jfo

NAME pr -- print file

SYNOPSIS <u>pr</u> [<u>-lcm</u>] name_1 ...

DESCRIPTION <u>pr</u> produces a printed listing of one or more files. The output is separated into pages headed by the name of the file, a date, and the page number.

The optional flag <u>-l</u> causes each page to contain 78 lines instead of the standard 66 to accommodate legal size paper.

The optional flags <u>-c</u> (current date) and <u>-m</u> (modified date) specify which date will head all subsequent files. <u>-m</u> is default.

Interconsole messages via <u>write</u>(I) are forbidden during a <u>pr</u>.

FILES /dev/tty? to suspend messages.

SEE ALSO cat(I), cp(I), mesg(I)

DIAGNOSTICS -- (files not found are ignored)

BUGS none

OWNER ken, dmr

NAME	rew -- rewind tape
SYNOPSIS	_rew_ [digit]
DESCRIPTION	_rew_ rewinds DECtape drives. The digit is the logical tape number, and should range from 0 to 7. A missing digit indicates drive 0.
FILES	/dev/tap?
SEE ALSO	--
DIAGNOSTICS	"?" if there is no tape mounted on the indicated drive or if the file cannot be opened.
BUGS	--
OWNER	ken, dmr

NAME	rm -- remove (unlink) files
SYNOPSIS	*rm* 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. Directories cannot be removed by *rm*; cf. *rmdir*.
FILES	none
SEE ALSO	rmdir(I)
DIAGNOSTICS	If the file cannot be removed or does not exist, the name of the file followed by a question mark is typed.
BUGS	*rm* probably should ask whether a read-only file is really to be removed.
OWNER	ken, dmr

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	--
OWNER	ken, dmr

NAME roff -- format text

SYNOPSIS _roff_ [_+_number1] [_-_number2] [_-stop_] $name_1$ ...

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 "_+_number1" causes printing to begin at the first page numbered number1; the optional argument "_-_number2" stops printing after the page numbered number2. The optional argument "_-stop_" or "_-s_" causes printing to stop before each page including the first to allow paper manipulation; printing is resumed upon receipt of an interrupt signal. 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.

roff is described in a separate publication [1].

FILES

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

SEE ALSO [1] (See J. F. Ossanna)

DIAGNOSTICS none

BUGS --

OWNER jfo

NAME salv -- file system salvage

SYNOPSIS <u>/etc/salv</u>

DESCRIPTION <u>salv</u> will reconstruct the file system /dev/rk0 to a consistent state. This is the first step in putting things together after a bad crash. Salv performs the following functions:

> A valid free list is constructed.
>
> 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.

After a salv, a warm boot must be performed instantly to effect the change made. (Because the salv works on the disk copy of the file system super-block, and the core copy is unaffected.)

After a salv, files may be safely created and removed without causing more trouble. However, it is more likely than not that directories are corrupted as well, so a <u>ds</u> should be performed.

FILES /dev/rk0

SEE ALSO check(I), ds(I)

DIAGNOSTICS --

BUGS The file system to be salvaged should be an argument.

OWNER ken

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 simply passed 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 the standard commands, which reside in "/bin", are found.) If the "/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 input 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 trap
Illegal instruction
IOT trap
Power fail trap
EMT trap
Bad system call
Quit
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.

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.

Each "[" must be paired with a matching "]". The characters between "[" and "]" specify a class of characters. It matches any single character in a file name which is in the class. An ordinary character in the brackets specifies that character to be in the class. A pair of characters separated by "-" specifies each character lexically greater than or equal to the first and less than or equal to the second member of the pair is to be included in the class. If the first member of the pair lexically exceeds the second, the second member is the sole character specified.

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 <u>not</u> 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 the file, character sequences of the form "$n", where _n_ is a digit 0, ..., 9, are replaced by the _n_th 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

Any shell-detected error in a file of commands causes that shell to cease executing 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);
"No command", 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*.

OWNER

dmr, ken

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. The sort is line-by-line in increasing ASCII collating sequence.

Space required is 6*number-of-lines in bytes.

FILES

/tmp/stm?

SEE ALSO

--

DIAGNOSTICS

--

BUGS

Sort does not put a maximum on the size of file that it sorts. Thus a bus error will occur if too large an input file is supplied.

The input is copied to a temporary file. Thus the maximum file that can be sorted is the maximum non-special file (currently 64K bytes.)

OWNER

dmr, ken

NAME stat -- get file status

SYNOPSIS <u>stat</u> name_1 ...

DESCRIPTION <u>stat</u> 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/uids" a number is given.

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

FILES /etc/uids

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

DIAGNOSTICS "name?" for any error.

OWNER dmr

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	--
OWNER	dmr

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 input (no erase/kill)
-canon	"
-raw	negate raw mode (erase/kill)
canon	"
cr	allow (and echo) cr for lf.
-nl	"
nl	negate cr mode.
-cr	"
echo	echo back every character typed.
full	"
-half	"
half	do not echo characters as typed.
-echo	"
-full	"
lcase	map upper case to lower case
-ucase	"
ucase	do not map case
-lcase	"
space	map tabs into spaces
-tab	"
tab	do not map tabs
-space	"
delay	calculate cr and tab delays.
-delay	no cr/tab delays
ebcdic	ebcdic ball conversion (2741 only)
-corres	"
corres	correspondence ball conversion (2741 only)
-ebcdic	"

FILES standard output.

SEE ALSO stty(II)

DIAGNOSTICS "Bad options"

BUGS --

OWNER jfo

NAME	su -- become privileged user
SYNOPSIS	<u>su</u> password
DESCRIPTION	su allows one to become the super-user, who has all sorts of marvelous powers. In order for su to do its magic, the user must pass as an argument 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.
FILES	--
SEE ALSO	sh(I)
DIAGNOSTICS	"Sorry" if password is wrong
BUGS	--
OWNER	dmr, ken

NAME sum -- sum file

SYNOPSIS <u>sum</u> name_1 ...

DESCRIPTION <u>sum</u> sums the contents of one or more files. A separate sum is printed for each file specified, along with the number of whole or partial 512-word blocks read.

In practice, <u>sum</u> 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 opened; "?" if if an error is discovered during the read.

BUGS none

OWNER ken

NAME	tacct -- login accounting by date
SYNOPSIS	<u>tacct</u> [wtmp]
DESCRIPTION	<u>tacct</u> will produce a printout giving daily connect time and total number of connects for all transactions found in the <u>wtmp</u> file. If no wtmp file is given, <u>/tmp/wtmp</u>is used.
FILES	/tmp/wtmp
SEE ALSO	init(VII), acct(I), login(I), wtmp(V)
DIAGNOSTICS	"Cannot open 'wtmp'"
BUGS	acct(I) and tacct(I) should be compined
OWNER	dmr, ken

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 only the entries, not the data associated with the entries are updated. 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.

i ignore tape errors. It is suggested that this option be used with caution to read damaged tapes.

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 The _m_ option does not work correctly. The _i_ option is not yet implemented.

OWNER ken

NAME tm -- provide time information

SYNOPSIS *tm* [command arg_1 ....]

DESCRIPTION *tm* is used to provide timing information. When used without an argument, 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 *tim* row is total real time (hours:minutes: seconds); unlike the other times, its origin is the creation date of *tm*'s temporary file. *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).

tm can be invoked with arguments which are assumed to constitute a command to be timed. In this case the output is as follows:

```
tim        2.7
ovh        0.3
swp        0.5
dsk        1.8
idl        0.0
usr        0.0
```

The given times represent the number of seconds spent in each category during execution of the command.

FILES /tmp/ttmp, /dev/rf0 (for absolute times) contains the information used to calculate the differential times.

SEE ALSO file system(V)

DIAGNOSTICS "?" if the command cannot be executed; "can't creat temp file" if trouble with *ttmp*; "cant read super-block" if times cannot be read from system.

BUGS (1) when invoked with a command argument, everything going on at the moment is counted, not just the command itself. (2) Two users doing _tm_ simultaneously interfere with each other's use of the temporary file.

OWNER ken, dmr

NAME	tss -- interface to Honeywell TSS
SYNOPSIS	<u>tss</u>

DESCRIPTION

<u>tss</u> 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 <u>!</u> are interpreted as UNIX commands. Input lines beginning with <u>~</u> 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.

<u>SYSTEM?</u> CSR/DACCOPY (s) AFTname
<u>Send Encoded File</u> ~s file

<u>SYSTEM?</u> CSR/DACCOPY (<u>r</u>) AFTname
<u>Receive Encoded File</u> ~r file

FILES	/dev/dn0, /dev/dp0
SEE ALSO	--
DIAGNOSTICS	DONE when communication is broken.
BUGS	When diagnostic problems occur, <u>tss</u> exits rather abruptly.
OWNER	csr

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	--
OWNER	dmr, ken

NAME type — type on single sheet paper

SYNOPSIS _type_ name_1 ...

DESCRIPTION _type_ copys its input files to the standard output. After every 66 lines, 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 —

OWNER dmr

NAME	umount -- dismount file system
SYNOPSIS	<u>/etc/umount</u> special
DESCRIPTION	<u>umount</u> announces to the system that the removable file system previously mounted on special file <u>special</u> is to be removed. Only the super-user may issue this command.
FILES	--
SEE ALSO	mount(I)
DIAGNOSTICS	"?"
BUGS	This command is not, in fact, restricted to the super-user.
OWNER	ken, dmr

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	--
OWNER	dmr, ken

NAME	wc -- get (English) word count
SYNOPSIS	wc name$_1$ ...
DESCRIPTION	wc provides a count of the words, text lines, and roff control lines for each argument file. A text line is a sequence of characters not beginning with "." and ended by a new-line. A roff control line is a line beginning with ".". 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.
FILES	--
SEE ALSO	roff(I)
DIAGNOSTICS	none; arguments not found are ignored.
BUGS	--
OWNER	jfo

NAME	who -- who is on the system
SYNOPSIS	<u>who</u> [who-file]
DESCRIPTION	<u>who</u>, without an argument, lists the name, typewriter channel, and login time for each current UNIX user. Without an argument, <u>who</u> examines the <u>/tmp/utmp</u> file to obtain its information. If a file is given, that file is examined. Typically the given file will be <u>/tmp/wtmp</u>, which contains a record of all the logins since it was created. Then <u>who</u> will list all logins and logouts since the creation of the wtmp file.
FILES	/tmp/utmp
SEE ALSO	login(I), init(VII)
DIAGNOSTICS	"?" if a named file cannot be read.
BUGS	--
OWNER	dmr, ken

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 --

OWNER dmr, ken

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 8K words. 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*.

FILES --

SEE ALSO exec(II)

DIAGNOSTICS none; strange addresses cause the break to be set to include all of core.

BUGS --

OWNER ken, dmr

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. Prior to the use of this call, the result of an _emt_ instruction is a simulated _rts_ instruction. The operand field is interpreted as a register, and an _rts_ instruction is simulated for that register (after verifying that various registers have appropriate values). This feature is useful for debugging, since the most dangerous program bugs usually involve an _rts_ with bad data on the stack or in a register.
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

NAME	chdir -- change working directory
SYNOPSIS	sys chdir; dirname / chdir = 12.
DESCRIPTION	*dirname* is address of the pathname of a directory, terminated by a 0 byte. *chdir* causes this directory to become the current working directory.
FILES	--
SEE ALSO	chdir(I)
DIAGNOSTICS	The error bit (c-bit) is set if the given name is not that of a directory.
BUGS	--
OWNER	ken, dmr

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 _or_ing 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.

FILES --

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 --

OWNER ken, dmr

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 able to misuse other's files.
FILES	--
SEE ALSO	chown(I), uids(V)
DIAGNOSTICS	The error bit (c-bit) is set on illegal owner changes.
BUGS	--
OWNER	ken, dmr

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 to programs which deal with many files.
FILES	--
SEE ALSO	creat(II), open(II)
DIAGNOSTICS	The error bit (c-bit) is set for an unknown file descriptor.
BUGS	--
OWNER	ken, dmr

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.

If the last link to an open file is removed, the file is not destroyed until the file is closed.

FILES

--

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.

BUGS

--

OWNER

ken, dmr

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 passed as follows. The stack pointer points to 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 60000(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.

FILES	--
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	--
OWNER	ken, dmr

NAME exit -- terminate process

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

DESCRIPTION _exit_ is the normal means of terminating a process. All files are closed and the parent process is notified if it is executing a _wait_. The low byte of r0 is available as status to the parent process.

This call can never return.

FILES --

SEE ALSO wait(II)

DIAGNOSTICS -

BUGS --

OWNER ken, dmr

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*.
FILES	--
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 subtle bug in process destruction.
OWNER	ken, dmr

NAME	fstat -- get status of open file
SYNOPSIS	(file descriptor in r0) sys fstat; buf / fstat = 28.
DESCRIPTION	This call is identical to <u>stat</u>, 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.
FILES	--
SEE ALSO	stat(II)
DIAGNOSTICS	The error bit (c-bit) is set if the file descriptor is unknown.
BUGS	--
OWNER	ken, dmr

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.
FILES	/etc/uids can be used to map the user ID number into a name.
SEE ALSO	setuid(II)
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

```
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*.

FILES

--

SEE ALSO

stty(II)

DIAGNOSTICS

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

BUGS

--

OWNER

ken, dmr

NAME	hog -- set program in low priority
SYNOPSIS	sys hog / hog = 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. A higher priority will be reinstituted as soon as the process is dismissed for any reason other than quantum overflow.
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

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.

FILES --

SEE ALSO fptrap(III)

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

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.

FILES --

SEE ALSO quit(II)

DIAGNOSTICS --

BUGS It should be easier to resume after an interrupt, but I don't know how to make it work.

OWNER ken, dmr

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.
FILES	--
SEE ALSO	--
DIAGNOSTICS	c-bit set if user is not the super-user, or if process does not exist.
BUGS	*kill* has been known to be ineffective.
OWNER	ken, dmr

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.

FILES --

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.

BUGS --

OWNER ken, dmr

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 can only be invoked by the super-user.

FILES --

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 --

OWNER ken, dmr

NAME mdate -- set modified date on file

SYNOPSIS

```
(time to AC-MQ)
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 AC-MQ registers.

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

FILES --

SEE ALSO --

DIAGNOSTICS Error bit is set if the user is not the super-user or if the file cannot be found.

BUGS --

OWNER ken, dmr

NAME mount -- mount file system

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

DESCRIPTION <u>mount</u> announces to the system that a removable file system has been mounted on special file <u>special</u>; from now on, references to file <u>name</u> will refer to the root file on the newly mounted file system. <u>Special</u> and <u>name</u> are pointers to null-terminated strings containing the appropriate path names.

<u>Name</u> must exist already. If it had useful contents, they are inaccessible while the file system is mounted.

Almost always, <u>name</u> should be a directory so that an entire file system, not just one file, may exist on the removable device.

FILES --

SEE ALSO mount(I), umount(II)

DIAGNOSTICS Error bit (c-bit) set if <u>special</u> is inaccessible or <u>dir</u> does not exist.

BUGS At most two removable devices can be mounted at a time. The use of this call should be restricted to the super-user.

OWNER ken, dmr

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.

If the last link to an open file is removed, the file is not destroyed until it is closed.

FILES --

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, or if the file is not readable.

BUGS --

OWNER ken, dmr

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 1, quits are re-enabled, and cause execution to cease and a core image to be produced. When _flag_ is an address in the program, a quit causes control to be sent to that address. Quits should be turned off only with due consideration.
FILES	--
SEE ALSO	intr(II)
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

NAME read -- read from file

SYNOPSIS

```
(file descriptor in r0)
sys read; buffer; nchars  / read = 3.
(nread in r0)
```

DESCRIPTION A file descriptor is a word returned from a successful _open_ call.

Buffer is the location of _nchars_ contiguous bytes into which the input will be placed. It is not guaranteed that all _nchars_ bytes will be read, however; 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.

FILES --

SEE ALSO open(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, all rare, can generate an error: physical I/O errors, bad buffer address, preposterous _nchars_, file descriptor not that of an input file.

BUGS --

OWNER ken, dmr

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.
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

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 (or 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_.

FILES --

SEE ALSO tell(II)

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 tape files and RK and RF. Something is going to be done about this.

OWNER ken, dmr

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.
FILES	--
SEE ALSO	getuid(II)
DIAGNOSTICS	Error bit (c-bit) is set if the current user ID is not that of the super-user.
BUGS	--
OWNER	ken, dmr

NAME	sleep -- stop execution for interval
SYNOPSIS	(60ths of a second in r0) sys sleep / sleep = 35.; not in assembler
DESCRIPTION	The current process is suspended from execution for the number of 60ths of a second specified by the contents of register 0.
FILES	--
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.
OWNER	ken, dmr

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
```

FILES	--
SEE ALSO	stat(I), fstat(II)
DIAGNOSTICS	Error bit (c-bit) is set if the file cannot be found.
BUGS	The format is going to change someday.
OWNER	ken, dmr

NAME stime -- set time

SYNOPSIS
```
(time in AC-MQ)
sys stime    / stime = 25.
```

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

FILES --

SEE ALSO date(I), time(II)

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

BUGS --

OWNER ken, dmr

NAME stty -- set mode of typewriter

SYNOPSIS

```
         (file descriptor in r0)
         sys stty; arg  / stty = 31.
         ...
    arg: dcrsr; dcpsr; 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 _dcpsr_ 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_ arguments 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 newline 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.

FILES --

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. It is all too easy to turn off your typewriter.

OWNER ken, dmr

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.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER ken

NAME tell -- get file pointer

SYNOPSIS

```
(file descriptor in r0)
sys tell; offset; ptrname  / tell = 20.
(value returned in r0)
```

DESCRIPTION The file descriptor refers to an open file. The value returned in r0 is one of:

if _ptrname_ is 0, the value returned is _offset_;

if _ptrname_ is 1, the value is the current pointer plus _offset_;

if _ptrname_ is 2, the value returned is the number of bytes in the file plus _offset_.

FILES --

SEE ALSO seek(II)

DIAGNOSTICS The error bit (c-bit) is set if the file descriptor is unknown.

BUGS Tell doesn't work. Complain if you need it.

OWNER ken, dmr

NAME	time -- get time of year
SYNOPSIS	sys time / time = 13. (time AC-MQ)
DESCRIPTION	*time* returns the time since 00:00:00, Jan. 1, 1971, measured in sixtieths of a second. The high order word is in the AC register and the low order is in the MQ.
FILES	--
SEE ALSO	date(I), stime(II)
DIAGNOSTICS	--
BUGS	The chronological-minded user will note that 2**32 sixtieths of a second is only about 2.5 years.
OWNER	ken, dmr

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.

FILES --

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.

OWNER ken, dmr

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.
FILES	--
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	Probably write permission should be required to remove the last link to a file, but this gets in other problems (namely, one can donate an undeletable file to someone else). If the system crashes while a file is waiting to be deleted because it is open, the space is lost.
OWNER	ken, dmr

NAME wait -- wait for process to die

SYNOPSIS

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

DESCRIPTION *wait* causes its caller to delay until one of its child processes terminates. If any child has already died, return is immediate; if there are no children, return is immediate with the error bit set. In the case of several children several *wait*s are needed to learn of all the deaths.

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

0	exit
1	bus error
2	trace trap
3	illegal instruction
4	IOT trap
5	power fail trap
6	EMT trap
7	bad system call
8	quit
9	interrupt
10	kill (see kill(II))
+16	core image produced

FILES --

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 is not really gone in that it still consumes disk swap and system table space. This can make it impossible to create new processes. The bug can be noticed when several "&" separators are given to the shell not followed by a command without an ampersand. Ordinarily things clean themselves up when an ordinary command is typed, but it is possible to get into a situation in which no commands are accepted, so no *wait*s are done; the system is then hung.

The fix, probably, is to have a new kind of *fork* which creates a process for which no *wait* is necessary (or possible); also to limit the number of active or inactive descendants allowed to a process.

NAME	write -- write on file
SYNOPSIS	(file descriptor in r0) sys write; buffer; nchars / 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 _nchars_ 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. For disk and tape files, writes which are multiples of 512 characters long and begin on a 512-byte boundary are more efficient than any others.
FILES	--
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	--
OWNER	ken, dmr

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 zero to pi/2. The atan2 entry returns the arc tangent of fr0/fr1 in fr0. The range is -pi to pi. The floating point simulation should be active in either floating or double mode, but in single precision integer mode.
FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap(III)
DIAGNOSTICS	--
BUGS	--
OWNER	rhm, dmr, ken

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_ 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 floating point simulation should be active in either floating or double mode, but in single precision integer mode.
FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap(III)
DIAGNOSTICS	--
BUGS	The subroutine _subr_ should not disturb any registers.
OWNER	ken

NAME	atoi -- ascii to integer
SYNOPSIS	jsr r5,atoi; subr
DESCRIPTION	_atoi_ will convert an ascii stream to a binary number returned in mq. The subroutine _subr_ 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.
FILES	kept in /usr/lib/liba.a
SEE ALSO	--
DIAGNOSTICS	--
BUGS	The subroutine _subr_ should not disturb any registers.
OWNER	ken

NAME	const -- floating point constants
SYNOPSIS	--
DESCRIPTION	The following floating point constants are correctly represented in double precision.

```
one      1.0
pi2      0.5*3.1415...
```

FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap(III)
DIAGNOSTICS	--
BUGS	--
OWNER	rhm, dmr, ken

NAME ctime -- convert date and time to ASCII

SYNOPSIS

```
(move time to AC-MQ)
mov     $buffer,r0
jsr     pc,ctime
```

DESCRIPTION The buffer is 15 characters long. The time has the format

```
     Oct  9 17:32:24
```

The input time is in the AC and MQ registers in the form returned by _sys_ _time_.

FILES kept in /usr/lib/liba.a

SEE ALSO ptime(III), time(II)

DIAGNOSTICS --

BUGS --

OWNER dmr

NAME	exp -- exponential function
SYNOPSIS	jsr r5,exp
DESCRIPTION	The exponential of fr0 is returned in fr0. The floating point simulation should be active in either floating or double mode, but in single precision integer mode.
FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap(III)
DIAGNOSTICS	The c-bit is set if the result is not representable.
BUGS	--
OWNER	rhm, dmr, ken

NAME fptrap -- PDP-11/45 floating point simulator

SYNOPSIS

```
.globl fptrap
sys ilgins; fptrap
```

DESCRIPTION *fptrap* is a package which picks up instructions which are illegal for the PDP-11/20, and if they correspond to 11/45 floating point instructions, simulates their operation. The following instructions are supported:

```
cfcc
setf
seti
setd
setl
clrf    fdst
tstf    fsrc
absf    fdst
negf    fdst
mulf    fsrc,fr
modf    fsrc,fr
addf    fsrc,fr
movf    fsrc,fr (=ldf)
movf    fr,fdst (=stf)
subf    fsrc,fr
cmpf    fsrc,fr
divf    fsrc,fr
movfi   fr,dst  (=stcfi)
movif   src,fr  (=ldcif)
movfo   fr,fdst (=stcxy)
movof   fsrc,fr (=ldcyx)
```

Here *src* and *dst* stand for source and destination, *fsrc* and *fdst* for floating source and destination, and *fr* for floating register. Notice that the names of several of the opcodes have changed. The only strange instruction is *movf*, which turns into *stf* if its source operand is a floating register, and into *ldf* if not.

The simulator sets the floating condition codes on both *ldf* and *stf*. The 11/45 hardware does not set the fcc on stf.

Short and long format for both floating point numbers and integers is supported. Truncation mode is always in effect. Traps for overflow and other arithmetic errors are not supported. Illegal instructions or addresses cause a simulated trap so that a core image is produced.

The condition code bits are maintained correctly.

For floating-point source operands, immediate mode ((pc)+) is not supported, since the

PDP-11/45 handbook is not clear on what to do about it.

After an arithmetic error the result is generally meaningless.

The arithmetic is always done in double-precision, so exact but unrounded results are to be expected in single-precision mode. Double precision results are probably less correct than the hardware will be.

The lower parts of the floating registers become meaningless during single-precision operations.

FILES	kept in /usr/lib/liba.a
SEE ALSO	PDP-11/45 handbook, ilgins(II)
DIAGNOSTICS	trap, c-bit, v-bit
BUGS	see above
OWNER	ken, dmr

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 [-]d.dddddddd*e*[-]dd*. The floating point simulator should be active in either floating or double mode, but in single integer mode. For each character generated by ftoa, the subroutine *subr* is called on register r5 with the character in r0.
FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap(III)
DIAGNOSTICS	--
BUGS	The subroutine *subr* should not disturb any registers.
OWNER	ken

NAME	connect, gerts -- Gerts communication over 201

SYNOPSIS

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

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

DESCRIPTION

The GECOS GERTS interface is so bad that a description here is inappropriate. Anyone needing to use this interface should contact the owner.

FILES

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

SEE ALSO

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

DIAGNOSTICS

--

BUGS

--

OWNER

ken

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.

FILES kept in /usr/lib/liba.a

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

DIAGNOSTICS c-bit set on EOF or error

BUGS --

OWNER dmr

NAME hypot -- calculate hypotenuse

SYNOPSIS

```
(A in fr0)
(B in fr0)
jsr     r5,hypot
```

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.

The floating point simulator should be active in either single or double mode.

FILES kept in /usr/lib/liba.a

SEE ALSO fptrap(III)

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

BUGS --

OWNER ken, dmr

NAME	itoa -- integer to ascii conversion
SYNOPSIS	jsr r5,itoa; subr
DESCRIPTION	_itoa_ will convert the number in r0 into ascii decimal possibly preceded by a - sign. For each character generated by itoa, the subroutine _subr_ is called on register r5 with the character in r0.
FILES	kept in /usr/lib/liba.a
SEE ALSO	--
DIAGNOSTICS	--
BUGS	The subroutine _subr_ should not disturb any registers.
OWNER	ken

NAME	log -- logarithm base e
SYNOPSIS	jsr r5,log
DESCRIPTION	The logarithm base e of fr0 is returned in fr0. The floating point simulation should be active in either floating or double mode, but in single precision integer mode.
FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap
DIAGNOSTICS	The error bit (c-bit) is set if the input argument is less than or equal to zero.
BUGS	--
OWNER	ken

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 is terminated by a 0 byte.

FILES kept in /usr/lib/liba.a

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME nlist -- get entries from name list

SYNOPSIS

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

DESCRIPTION <u>nlist</u> will examine the name list in an assembler output file and selectively extract a list of values. The file name is a standard UNIX path name. 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 /usr/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 --

OWNER ken

NAME

ptime -- print date and time

SYNOPSIS

```
(move time to ac-mq)
mov     file,r0
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 r0. The string is 15 characters long. The time to be printed is placed in the AC and MQ registers in the form returned by sys time.

FILES

kept in /usr/lib/liba.a

SEE ALSO

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

DIAGNOSTICS

--

BUGS

see ctime

OWNER

dmr, ken

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.

FILES

kept in /usr/lib/liba.a

SEE ALSO

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

DIAGNOSTICS

error bit possible on _fcreat_ call

BUGS

--

OWNER

dmr

NAME qsort -- quicker sort

SYNOPSIS

```
(base of data in r1)
(end 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 byte strings. 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, AC and MQ are destroyed.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS The user should be able to supply his own comparison routine.

OWNER ken

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 current position of the write pointer.

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

fsfile moves the write pointer to the current position of the read 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 proper is in /usr/llc/alloc/alloca.

The archive /usr/llc/alloc/allocb contains the individual routines discussed above.

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 —

OWNER llc,rhm

NAME	sin, cos -- sine cosine
SYNOPSIS	jsr r5,sin (cos)
DESCRIPTION	The sine (cosine) of fr0 (radians) is returned in fr0. The floating point simulation should be active in either floating or double mode, but in single precision integer mode. All floating registers are used.
FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap(III)
DIAGNOSTICS	--
BUGS	Size of the argument should be checked to make sure the result is meaningful.
OWNER	ken, dmr

NAME	sqrt -- square root function
SYNOPSIS	`jsr     r5,sqrt`
DESCRIPTION	The square root of fr0 is returned in fr0. The floating point simulation should be active in either floating or double mode, but in single precision integer mode.
FILES	kept in /usr/lib/liba.a
SEE ALSO	fptrap(III)
DIAGNOSTICS	The c-bit is set on negative arguments.
BUGS	--
OWNER	rhm, dmr, ken

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 /usr/lib/liba.a

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME	dn0 -- dn-11 ACU interface
SYNOPSIS	--
DESCRIPTION	*dn0* is a write-only file. Bytes written on *dn0* must be ASCII digits. Each digit corresponds to a digit of a telephone number to be called. The entire telephone number must be presented in a single *write* system call. The call must complete with the last digit.
FILES	found in /dev
SEE ALSO	dp0(IV), write(II)
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

NAME	dp0 -- dp-11 201 data-phone interface
SYNOPSIS	--
DESCRIPTION	*dp0* is a 201 data-phone interface file. *read* and *write* calls to *dp0* 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	found in /dev
SEE ALSO	dn0(IV), gerts(III)
DIAGNOSTICS	--
BUGS	The *dp* file is GECOS oriented. It should be more flexible.
OWNER	ken, dmr

NAME /dev/lpr -- line printer

SYNOPSIS --

DESCRIPTION The line printer special file is the UNIX interface to a DEC LP-11 line printer. This file may only be opened (or creat'ed) for writing. Anything written on this file is printed on the line printer. The following special cases for the printer are handled:

On opening and on closing, the paper is slewed to the top of the next page.

For the 64 character printer (LP11-FA), all lower case letters are converted to upper case.

Tabs are converted to align on every eighth column.

New lines and form feeds are ignored when the printer is at the top of a page. This is done so that *pr* and *roff* output may be directed to the printer and sync on page boundaries even with automatic page slew.

Carriage return and back space can cause multiple printing on a single line to allow for overstruck graphics.

FILES found in /dev

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME	mem -- core memory
SYNOPSIS	--
DESCRIPTION	mem maps the core memory of the computer into a file. 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.
FILES	found in /dev
SEE ALSO	--
DIAGNOSTICS	--
BUGS	If a location not corresponding to implemented memory is read or written, the system will incur a bus-error trap and, in panic, will reboot itself.
OWNER	ken, dmr

NAME	mt0 -- magtape
SYNOPSIS	--
DESCRIPTION	*mt0* is the DEC TU10/TM11 magtape. When opened for reading or writing, the magtape is rewound. A tape consists of a series of 256 word records terminated by an end-of-file. Reading less than 256 words (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	found in /dev
SEE ALSO	mt(I)
DIAGNOSTICS	--
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.
OWNER	ken, dmr

NAME ppt -- punched paper tape

SYNOPSIS --

DESCRIPTION

ppt refers to the 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 characters will usually appear at the beginning of the file; they correspond to the tape leader. 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 and are effectively ignored (however, the read/write pointers are maintained and an arbitrary sequence of reads or writes intermixed with seeks will give apparently correct results when checked with _tell_).

FILES found in /dev

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME	rf0 -- RF11-RS11 fixed-head disk file
SYNOPSIS	--
DESCRIPTION	This file refers to the entire RF disk. It may be either read or written, although writing is inherently very dangerous, since a file system resides there. The disk contains 1024 256-word blocks, numbered 0 to 1023. Like the other block-structured devices (tape, RK disk) 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	found in /dev
SEE ALSO	tap0(IV), rk0(IV)
DIAGNOSTICS	--
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_ and _tell_) are single-precision numbers. This really has to be changed but unfortunately the repercussions are serious.
OWNER	ken, dmr

NAME	rk0 -- RK03 (or RK05) disk
SYNOPSIS	--
DESCRIPTION	*rk0* refers to the 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 rf0(IV) section.
FILES	found in /dev
SEE ALSO	rf0(IV), tap0(IV)
DIAGNOSTICS	--
BUGS	See rf0(IV)
OWNER	ken, dmr

NAME	rp0 -- RP11/RP02 disk
SYNOPSIS	--
DESCRIPTION	_rp0_ refers to the entire RP02 disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 40599. Like the RF disk and the tape files, its addressing is block-oriented. Consult the rf0(IV) section.
FILES	found in /dev
SEE ALSO	rf0(IV), tap0(IV)
DIAGNOSTICS	--
BUGS	See rf0(IV) Due to a hardware bug, block 40599 on the RP cannot be accessed.
OWNER	ken, dmr

NAME	tap0 ... tap7
SYNOPSIS	--
DESCRIPTION	These files refer to 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. Like the RK and RF special files, addressing on the tape files is block-oriented. See the RF0 section.
FILES	found in /dev
SEE ALSO	/dev/rf0, /dev/rk0
DIAGNOSTICS	--
BUGS	see /dev/rf0
OWNER	ken, dmr

NAME	tty -- console typewriter
SYNOPSIS	--
DESCRIPTION	*tty* (as distinct from *tty0*, ..., *ttyn*) refers to the console typewriter hard-wired to the PDP-11. Generally, the disciplines involved in dealing with *tty* are similar to those for *tty0* ... and the appropriate section 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 *tty0*...) 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	found in /dev
SEE ALSO	tty0(IV), init(VII)
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

NAME tty0 ... tty7 -- communications interfaces

SYNOPSIS --

DESCRIPTION These files refer to DC11 asynchronous communications interfaces. At the moment there are eight 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 standard interface mode assumed includes: ASCII characters; 150 baud; even parity accepted; 10 bits/character (one stop bit); and newline action character. The system delays transmission after sending certain function characters; delays for horizontal tab, newline, and form feed are calculated for the Teletype Model 37; the delay for carriage return is calculated for the GE TermiNet 300. Most of these operating states can be changed by using the system call stty(II). In particular the following hardware states are program settable independently for input and output (see DC11 manual): 110, 134.5, 150, 300, 600, or 1200 baud; one or two stop bits on output; and 5, 6, 7, or 8 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 echoing of input 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 EOF. The character "@" kills the entire line up to the point where it was typed, but not beyond an EOF. 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 wakened on each character. The program waits only until at least one character has been typed. 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.

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. If interrupts aren't being ignored, 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).

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 program 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 found in /dev

SEE ALSO tty(I), getty(VII)

DIAGNOSTICS --

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

OWNER ken, dmr, jfo

NAME a.out -- assembler and link editor output

SYNOPSIS --

DESCRIPTION *a.out* is the output file of the assembler *as* and the link editor *ld*. In both cases, *a.out* is executable 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 "br .+20" instruction (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 logical segments are set up: the text segment, the data segment, and the uninitialized 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.

If _a.out_ contains no unresolved global references, the text portions are exactly as they will appear in core when the file is executed. If the value of a word in the text portion involves a reference to an undefined global, the word is replaced by the offset to be added to the symbol's value when it becomes defined.

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.

FILES	--
SEE ALSO	_as_ _ld_, _strip_, _nm_, _un_(_I_)
DIAGNOSTICS	--
BUGS	--
OWNER	dmr

NAME archive (library) file format

SYNOPSIS --

DESCRIPTION The archive command _ar_ is used to combine several files into one. Its use has three benefits: when files are combined, the file space consumed by the breakage at the end of each file (256 bytes on the average) is saved; directories are smaller and less confusing; archive files of object programs may be searched as libraries by the loader _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 -147(10), or 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.

FILES --

SEE ALSO _ar_, _ld_

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME format of core image

SYNOPSIS --

DESCRIPTION Three conditions cause UNIX to write out the core image of an executing program: the program generates an unexpected trap (by a bus error or illegal instruction); the user sends a "quit" signal (which has not been turned off by the program); a trap is simulated by the floating point simulator. The core image is called "core" and is written in the current 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. 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, is dumped.

When any trap which is not an I/O interrupt occurs, all the useful registers are stored on the stack. After all the registers have been stored, the contents of _sp_ are placed in the first cell of the user area; this cell is called _u.sp_. Therefore, within the core image proper, there is an area which contains the following registers in the following order (increasing addresses):

```
(u.sp)->sc
        mq
        ac
        r5
        r4
        r3
        r2
        r1
        r0
        pc (at time of fault)
        processor status (at time of fault)
```

The last two are stored by the hardware. It follows that the contents of _sp_ at the time of the fault were (u.sp) plus $22_{(10)}$.

The actual location of this data depends on which system is being used. In the current system, which has relocation and protection hardware, the stack discussed above is the system stack, and is kept in the per-user area; in older systems, there is only one stack, and it is located in the user's core area.

In general the debugger db(I) should be used to deal with core images.

FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

NAME format of directories

SYNOPSIS --

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-node 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 necessarily 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 without a _mount_ system call. Therefore in both cases ".." has the same meaning as ".".

FILES --

SEE ALSO file system format

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME format of file system

SYNOPSIS --

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 8_n_-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_(mod 8) bits from the right. Notice that bits exist for the superblock 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) (mod 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·((_i_+31)(mod 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. The swap areas are at the high addresses on the device. It would be convenient if these addresses did not appear in the free list, but in fact this is not so. Therefore a certain number of blocks at the top of the device appear in the free map, are not marked free, yet do not appear within any file. These are the blocks that show up "missing" in a _check_ of the RF disk.

Again on the primary file system device, there are several pieces of information following that previously discussed. They contain basically the information typed by the tm command; namely, the times spent since a cold boot in various categories, and a count of I/O errors. In particular, there are two words with the calendar time (measured since 00:00 Jan 1, 1971); 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 *b*th entry in the address portion of the i-node.

If the file is large, *b* is divided by 256 to yield a number which must be less than 8 (or the file is too large for UNIX to handle). The corresponding slot in the i-node address portion gives the physical block number of an indirect block. The residue mod 256 of *b* is multiplied by two (to give a byte offset in the indirect block) and the word found there is the physical address of the block corresponding to *b*.

If block *b* in a file exists, 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.

FILES --

SEE ALSO format of directories

DIAGNOSTICS --

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.

OWNER --

NAME	ident -- IDENT card file
SYNOPSIS	--
DESCRIPTION	*ident* is a file used to generate GECOS $IDENT cards by the off-line print program opr(I). There is one entry per line in the following style:

```
05:m1234,m789,name
```

which causes the following $IDENT card to be generated:

```
$       IDENT    m1234,m789,name
```

FILES	kept in /etc/ident.
SEE ALSO	opr(I)
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

NAME passwd -- password file

SYNOPSIS --

DESCRIPTION <u>passwd</u> contains for each user the following information:

name (login name)
password
numerical user ID
default 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. 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, naturally, is inaccessible to anyone but the super-user.

This file resides in directory /etc.

FILES --

SEE ALSO /etc/init

DIAGNOSTICS --

BUGS --

OWNER super-user

NAME tap -- DEC/mag tape formats

SYNOPSIS --

DESCRIPTION The DECtape command tap and the magtape command mt dump and extract files to and from their respective tape media. The format of these tapes are the same.

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

Blocks 1 thru 24 contain a directory of the tape. There are 192 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 inodes (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 is zero.

Blocks 25 on are available for file storage.

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

FILES --

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

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME /etc/uids -- map user names to user IDs

SYNOPSIS --

DESCRIPTION This file allows programs to map user names into user numbers and vice versa. Anyone can read it. It resides in directory /etc, and should be updated along with the password file when a user is added or deleted.

The format is an ASCII name, followed by a colon, followed by a decimal ASCII user ID number.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER dmr, ken

NAME /tmp/utmp -- user information

SYNOPSIS --

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 (currently, '0' to '5' for /dev/tty0 to /dev/tty5). The next two words contain the user's login time. The last word is unused.

This file resides in directory /tmp.

FILES --

SEE ALSO /etc/init, which maintains the file.

DIAGNOSTICS --

BUGS --

OWNER ken, dmr

NAME	/tmp/wtmp -- user login history
SYNOPSIS	--
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.
FILES	--
SEE ALSO	init(VII), login(I), tacct(I), acct(I)
DIAGNOSTICS	--
BUGS	--
OWNER	ken, dmr

NAME	basic -- DEC supplied BASIC
SYNOPSIS	basic [file]
DESCRIPTION	Basic is the standard BASIC V000 distributed as a stand alone program. The optional file argument is read before the console. See DEC-11-AJPB-D manual. Since _bas_ is smaller and faster, _basic_ is not maintained on line.
FILES	--
SEE ALSO	bas
DIAGNOSTICS	See manual
BUGS	GOK
OWNER	dmr

NAME	bc -- B interpreter
SYNOPSIS	<u>bc</u> [<u>-c</u>] sfile$_1$<u>.b</u> ... ofile$_1$ ...

DESCRIPTION

<u>bc</u> 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 <u>bc</u> 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 <u>a.out</u>.

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 if 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.
c(I)

DIAGNOSTICS

see [1].

BUGS

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

OWNER

ken, dmr

NAME bj -- the game of black jack

SYNOPSIS /usr/games/bj

DESCRIPTION Black jack 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?'
Insureance?

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.

FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken

NAME	cal -- print calendar
SYNOPSIS	/usr/ken/cal year
DESCRIPTION	Cal will print a calendar for the given year. The year can be between 0 (really 1 BC) and 9999. For years when several calendars were in vogue in different countries, the calendar of England (and therefore her colonies) is printed. P.S. try cal of 1752.
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken

NAME chash -- precompile a hash table for <u>cref</u>

SYNOPSIS <u>chash</u> file1 file2

DESCRIPTION CHASH takes symbols (character sequences; one per line) from <u>file1</u> and compiles a hash table for the use of <u>cref</u>. The table is written on <u>file2</u>.

A subroutine suitable for searching such a hash table is available from the author.

FILES ---

SEE ALSO cref

DIAGNOSTICS ---

BUGS There can only be 199 symbols; they may total only 600 characters of text.

OWNER lem

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. Either _ignore_ or _only_ files must be made by _chash_ (q.v.). 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 chash(VI); 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 author

"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 eigth column.

OWNER

lem

NAME	das -- disassembler
SYNOPSIS	--
DESCRIPTION	A PDP-11 disassembler exists. Contact the owner for more information.
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken

NAME	dli -- load DEC binary paper tapes
SYNOPSIS	dli output [input]
DESCRIPTION	<u>dli</u> 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	--
OWNER	dmr

NAME dpt -- read DEC ASCII paper tape

SYNOPSIS dpt output [input]

DESCRIPTION _dpt_ reads the input file (/dev/ppt default) assuming the format is a DEC generated ASCII paper tape of an assembly language program. The output is a UNIX ASCII assembly program.

FILES /dev/ppt

SEE ALSO --

DIAGNOSTICS --

BUGS Almost always a hand pass is required to get a correct output.

OWNER ken, dmr

NAME	moo -- a game
SYNOPSIS	/usr/games/moo
DESCRIPTION	<u>moo</u> is a guessing game imported from England.
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken

NAME ptx — permuted index

SYNOPSIS

ptx1 input temp1
sort temp1 temp2
ptx2 temp2 output

DESCRIPTION

ptx generates a permuted index from file _input_ on file _output_. It is in two pieces: 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 must then be sorted. _ptx2_ then rotates each line around 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

DIAGNOSTICS --

BUGS --

OWNER dmr

NAME	tmg -- compiler compiler
SYNOPSIS	*tmg* name

DESCRIPTION

tmg produces a translator for the language whose syntactic 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.

The tmg language is described in (Reference).

FILES

/etc/tmg -- the compiler-compiler
/etc/tmga,/etc/tmgb,/etc/tmgc -- libraries
/etc/tmg0.s -- global definitions

SEE ALSO --

DIAGNOSTICS

??? -- illegal input, offending line follows
fatal error codes, appear in tmg and a.out:

ad	address out of bounds
so	stack overflow
ga	address out of bounds while generating
ko	too much parse without output
to	symbol table overflow
gn	getnam on symbol not in table
co	character string overflow

BUGS --

OWNER doug

NAME	ttt -- tic-tac-toe
SYNOPSIS	/usr/games/ttt
DESCRIPTION	_ttt_ is the X's and O's game popular in 1st grade. This is a learning program that never makes the same mistake twice.
FILES	ttt.k -- old mistakes
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken

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

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER jfo

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

SYNOPSIS —

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 UNIX read only memory (ROM) is home cut with 2 programs of 16 words each. The first (address 173000) reads <u>bos</u> from the RF disk into core location 154000 and transfers to 154000. The other ROM program (address 173040) reads a DECtape sitting in the end-zone on drive 0 into core location 0 and transfers to 0. This latter operation is compatible with part of DEC's standard ROM. The disassembled code for the UNIX ROM follows:

```
173000: mov     $177472,r0                12700;177472
        mov     $3,-(r0)                  12740;3
        mov     $140000,-(r0)             12740;140000
        mov     $154000,-(r0)             12740;154000
        mov     $-2000,-(r0)              12740;176000
        mov     $5,-(r0)                  12740;5
        tstb    (r0)                      105710
        bge     .-2                       2376
        jmp     *$154000                  137;154000

173040: mov     $177350,r0                12700;177350
        clr     -(r0)                     5040
        mov     r0,-(r0)                  10040
        mov     $3,-(r0)                  12740;3
        tstb    (r0)                      105710
        bge     .-2                       2376
        tst     *$177350                  5737;177350
        bne     .                         1377
        movb    $5,(r0)                   112710;5
        tstb    (r0)                      105710
        bge     .-2                       2376
        clr     pc                        5007
```

The program <u>bos</u> (Bootstrap Operating System) examines the console switchs 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.

Thus we come to the UNIX warm boot procedure: put 173000 into the switches, push _load address_ and then push _start_. The alternate switch setting of 173030 that will load warm UNIX is used as a signal to bring up a single user system for special purposes. See init(VII). For systems without a rom, UNIX (both warm and cold) have a copy of the disk boot program at location 602. This is probably a better warm boot procedure because the program at 602 also attempts to complete outstanding I/O.

Cold boots can be accomplished with the Cold UNIX program, but they're not. Thus the Cold UNIX slot on the RF may have any program desired. This slot is, however, used during a cold boot. Mount the UNIX INIT DECtape on drive 0 positioned in the end-zone. Put 173040 into the switches. Push _load address_. Put 1 into the switches. Push _start_. This reads a program called _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_ then reads in Cold UNIX and executes that. Cold UNIX halts for a last chance before it completely initializes the RF file system. Push _continue_, and Cold UNIX will initialize the RF. It then sets into execution a user program that reads the DECtape for initialization files starting from block 33. When this is done, the program executes /etc/init which should have been on the tape.

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
```

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_. _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 characacter _char_. Char is taken from the following set:

_b_	bos
_u_	Warm UNIX
_1_	Cold UNIX

Due to their rarity of use, _maki_ and _msys_ are maintained off line and must be reassembled before used.

FILES /dev/rf0, /dev/tap?

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

DIAGNOSTICS --

BUGS This section is very configuration dependent. Thus, it does not describe the boot procedure for any one machine.

OWNER ken

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

SYNOPSIS --

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, 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	/etc/getty
SEE ALSO	init(VII), login(I), stty(II)
DIAGNOSTICS	--
BUGS	--
OWNER	dmr, ken, jfo

NAME	glob -- generate command arguments
SYNOPSIS	--
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.
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.
OWNER	dmr

NAME init -- process control initialization

SYNOPSIS --

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 dataphone 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 kept in /etc/init; uses /dev/tap, /dev/tty, /dev/tty?, /tmp/utmp, /tmp/wtmp

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

DIAGNOSTICS	none possible
BUGS	none possible
OWNER	ken, dmr

NAME kbd -- keyboard map

SYNOPSIS cat /usr/pub/kbd

DESCRIPTION *kbd* contains a map to the keyboard for model 37 Teletype terminals with the extended character set feature. If *kbd* is printed on such a terminal, the following will appear:

```
<[1234567890-_]^\ >qwertyuiop@ asdfghjkl;: zxcvbnm,./

<▽1234567890-¬∂∫ϒ >          ν          ;:        ,./


<{!"#$%&'() =_}~  >QWERTYUIOP` ASDFGHJKL+* ZXCVBNM,.?

< !"#$%&'() =¬     >ζΔΛΣθσφτΘΠ  αεδΦΓΨπρλ+* Ωξωψβημ,.?
```

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER jfo

NAME logging in and logging out

SYNOPSIS --

DESCRIPTION UNIX must be called from an appropriate terminal. UNIX supports ASCII terminals typified by the Teletype M37, the GE Terminet 300, the Memorex 1240, and various graphical terminals on the one hand, and IBM 2741-type terminals on the other.

Not all installations support all these terminals. Often the M33/35 Teletype is supported instead of the 2741. Depending on the hardware installed, most terminals operating at 110, 134.5, 150, or 300 baud can be accommodated.

To use UNIX, it is also necessary to have a valid UNIX user ID and (if desired) password. These may be obtained, together with the telephone number, from the system administrators.

The same telephone number serves terminals operating at all the standard speeds. The discussion below applies when the standard speeds of 134.5 (2741's) 150 (TTY 37's) and 300 (Terminet 300's) are available.

When a connection is established via a 150-baud terminal (e.g. TTY 37) UNIX types out "login:"; you respond with your user name, and, if requested, with a password. (The printer is turned off while you type the password.) If the login was successful, the "@" character is typed by the Shell to indicate login is complete and commands may be issued. A message of the day may be typed if there are any announcements. Also, if there is a file called "mailbox", you are notified that someone has sent you mail. (See the mail command.)

From a 300-baud terminal, the procedure is slightly different. Such terminals often have a full-duplex switch, which should be turned on (or conversely, half-duplex should be turned off). When a connection with UNIX is established, a few garbage characters are typed (these are the "login:" message at the wrong speed). You should depress the "break" key; this is a speed-independent signal to UNIX that a 300-baud terminal is in use. It will type "login:" (at the correct speed this time) and from then on the procedure is the same as described above.

From a 2741, no message will appear. After the telephone connection is established, press the "ATTN" button. UNIX should type "login:" as described above. If the greeting does not appear after a few seconds, hang up and try again; something has gone wrong. If a password is required, the printer cannot be turned off, so it will appear on the paper when you type it.

For more information, consult getty(VII), which discusses the login sequence in more detail, and tty0(IV), which discusses typewriter I/O.

Logging out is simple by comparison (in fact, sometimes too simple). Simply generate an end-of-file at Shell level by using the EOT character; the "login:" message will appear again to indicate that you may log in again.

It is also possible to log out simply by hanging up the terminal; this simulates an end-of-file on the typewriter.

FILES /etc/motd may contain a message-of-the-day.

SEE ALSO init(VII), getty(VII), tty0(IV)

DIAGNOSTICS --

BUGS Hanging up on programs which never read the typewriter or which ignore end-of-files is very dangerous; in the worst cases, the programs can only be halted by restarting the system.

OWNER ken, dmr

NAME msh -- mini-shell

SYNOPSIS --

DESCRIPTION <u>msh</u> 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 <u>glob</u> is called. The main use of <u>msh</u> is to provide a command-executing facility for various interactive sub-systems.

FILES found in /etc/msh

SEE ALSO sh, glob

DIAGNOSTICS "?"

BUGS --

OWNER ken, dmr

NAME	tabs -- tab stop set
SYNOPSIS	cat /usr/pub/tabs
DESCRIPTION	When printed on a suitable terminal, this file will set tab stops at columns 8, 16, 24, 32, ... Suitable terminals include the Teletype model 37 and the GE TermiNet 300. These tabs stop settings are desirable because UNIX assumes them in calculating delays.
FILES	--
SEE ALSO	--
DIAGNOSTICS	--
BUGS	--
OWNER	ken