NAME

ar – archive and library maintainer

SYNOPSIS

ar key afile name ...

DESCRIPTION

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

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

d means delete the named files from the archive file.

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

t prints a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

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

x will extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

v means verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. The following abbreviations are used:

c copy
a append
d delete
r replace
x extract

FILES

/tmp/vtm? temporary

SEE ALSO

ld(I), archive(V)

BUGS

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

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

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

NAME

as – assembler

SYNOPSIS

as [–] name ...

DESCRIPTION

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

)	Parentheses error
]	Parentheses error
<	String not terminated properly
*	Indirection used illegally
.	Illegal assignment to '**.**'
A	Error in address
B	Branch instruction is odd or too remote
E	Error in expression
F	Error in local ('f' or 'b') type symbol
G	Garbage (unknown) character
I	End of file inside an if
M	Multiply defined symbol as label
O	Word quantity assembled at odd address
P	'**.**' different in pass 1 and 2
R	Relocation error
U	Undefined symbol
X	Syntax error

BUGS

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

NAME

bas – basic

SYNOPSIS

bas [file]

DESCRIPTION

Bas is a dialect of Basic. If a file argument is provided, the file is used for input before the console is read. *Bas* accepts lines of the form:

statement
integer statement

Integer numbered statements (known as internal statements) are stored for later execution. They are stored in sorted ascending order. Non-numbered statements are immediately executed. The result of an immediate expression statement (that does not have '=' as its highest operator) is printed.

Statements have the following syntax:

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

done
Return to system level.

draw expression expression expression
A line is drawn on the Tektronix 611 display '/dev/vt0' from the current display position to the XY co-ordinates specified by the first two expressions. The scale is zero to one in both X and Y directions. If the third expression is zero, the line is invisible. The current display position is set to the end point.

display list
The list of expressions and strings is concatenated and displayed (i.e. printed) on the 611 starting at the current display position. The current display position is not changed.

erase
The 611 screen is erased.

for name = expression expression statement
for name = expression expression
...
next
The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression.

goto expression
The expression is evaluated, truncated to an integer and execution goes to the corresponding integer numbered statment. If executed from immediate mode, the internal statements are compiled first.

if expression statement
The statement is executed if the expression evaluates to non-zero.

list [expression [expression]]
is used to print out the stored internal statements. If no arguments are given, all internal statements are printed. If one argument is given, only that internal statement is listed. If two arguments are given, all internal statements inclusively between the arguments are printed.

print list
The list of expressions and strings are concatenated and printed. (A string is delimited by

" characters.)

return [expression]
The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run
The internal statements are compiled. The symbol table is re-initialized. The random number generator is reset. Control is passed to the lowest numbered internal statement.

Expressions have the following syntax:

name
A name is used to specify a variable. Names are composed of a letter followed by letters and digits. The first four characters of a name are significant.

number
A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an **e** followed by a possibly signed exponent.

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

expression operator expression
Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. A complete list of operators is given below.

expression **(** [expression [**,** expression] ...] **)**
Functions of an arbitrary number of arguments can be called by an expression followed by the arguments in parentheses separated by commas. The expression evaluates to the line number of the entry of the function in the internally stored statements. This causes the internal statements to be compiled. If the expression evaluates negative, a builtin function is called. The list of builtin functions appears below.

name **[** expression [**,** expression] ... **]**
Each expression is truncated to an integer and used as a specifier for the name. The result is syntactically identical to a name. **a[1,2]** is the same as **a[1][2].** The truncated expressions are restricted to values between 0 and 32767.

The following is the list of operators:

=
= is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

& |
& (logical and) has result zero if either of its arguments are zero. It has result one if both its arguments are non-zero. | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments are non-zero.

< <= > >= == <>
The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, <> not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: a>b>c is the same as a>b&b>c.

+ –
Add and subtract.

* /
Multiply and divide.

^
Exponentiation.

The following is a list of builtin functions:

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

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

log(x)
is the natural logarithm of *x*.

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

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

atn(x)
is the arctangent of *x* . its value is between $-\pi/2$ and $\pi/2$.

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

expr( )
is the only form of program input. A line is read from the input and evaluated as an expression. The resultant value is returned.

int(x)
returns *x* truncated to an integer.

FILES

/tmp/btm? temporary

DIAGNOSTICS

Syntax errors cause the incorrect line to be typed with an underscore where the parse failed. All other diagnostics are self explanatory.

BUGS

Has been known to give core images. Needs a way to *list* a program onto a file.

NAME

cat – concatenate and print

SYNOPSIS

cat file ...

DESCRIPTION

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

cat file

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

cat file1 file2 >file3

is about the easiest way to concatenate files.

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

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

SEE ALSO

pr(I), cp(I)

DIAGNOSTICS

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

BUGS

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

NAME

catsim – phototypesetter simulator

SYNOPSIS

catsim

DESCRIPTION

Catsim will interpret its standard input as codes for the phototypesetter (cat). The output of *catsim* is output to the display (vt).

About the only use of *catsim* is to save time and paper on the phototypesetter by the following command:

troff –t files | catsim

FILES

/dev/vt0

SEE ALSO

troff(I), cat(IV), vt(IV)

BUGS

Point sizes are not correct. The vt character set is restricted to one font of ASCII.

NAME

cc – C compiler

SYNOPSIS

cc [–**c**] [–**p**] file ...

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 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.

If the –**p** flag is used, only the macro prepass is run on all files whose name ends in **.c**. The expanded source is left on the file whose name is that of the source with **.i** substituted for **.c**.

FILES

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

SEE ALSO

'C reference manual', cdb(I), ld(I) for other flag arguments.

BUGS

NAME

cdb – C debugger

SYNOPSIS

cdb [core [a.out]]

DESCRIPTION

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

Even the present *cdb* has one useful feature: the command

$

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

SEE ALSO

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

BUGS

It has to be fixed to work with the new system.

NAME

chdir – change working directory

SYNOPSIS

chdir directory

DESCRIPTION

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

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

SEE ALSO

sh(I)

BUGS

NAME

chmod – change mode

SYNOPSIS

chmod octal file ...

DESCRIPTION

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

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

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

SEE ALSO

ls(I)

BUGS

NAME

chown – change owner

SYNOPSIS

chown owner file ...

DESCRIPTION

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

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

FILES

/etc/passwd

BUGS

NAME

cmp – compare two files

SYNOPSIS

cmp file1 file2

DESCRIPTION

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

SEE ALSO

proof (I), comm (I)

BUGS

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

NAME

comm – print lines common to two files

SYNOPSIS

comm [– [**123**]] file1 file2 [file3]

DESCRIPTION

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

If *file3* is given, the output will be placed there; otherwise it will be written on the standard output.

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

SEE ALSO

uniq(I), proof(I), cmp(I)

BUGS

NAME

cp – copy

SYNOPSIS

cp file1 file2

DESCRIPTION

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

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

SEE ALSO

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

BUGS

Copying a file onto itself destroys its contents.

NAME

cref – make cross reference listing

SYNOPSIS

cref [–**acilostux123**] name ...

DESCRIPTION

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

The output report is in four columns:

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

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

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

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

Options:

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

FILES

/tmp/crt??	temporaries
/usr/lib/aign	default assembler *ignore* file
/usr/lib/cign	default C *ignore* file
/usr/bin/crpost	post processor
/usr/bin/upost	post processor for –**u** option

/bin/sort used to sort temporaries

SEE ALSO

as(I), cc(I), sort(I)

BUGS

NAME

date – print and set the date

SYNOPSIS

date [mmddhhmm[yy]]

DESCRIPTION

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

date 10080045

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

BUGS

NAME

db – debug

SYNOPSIS

db [core [namelist]] [–]

DESCRIPTION

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

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

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

1. A name has the value assigned to it when the input file was assembled. It may be relocatable or not depending on the use of the name during the assembly.
2. An octal number is an absolute quantity with the appropriate value.
3. A decimal number immediately followed by '**.**' is an absolute quantity with the appropriate value.
4. An octal number immediately followed by **r** is a relocatable quantity with the appropriate value.
5. The symbol **.** indicates the current pointer of *db*. The current pointer is set by many *db* requests.
6. A ***** before an expression forms an expression whose value is the number in the word addressed by the first expression. A ***** alone is equivalent to '***.**'.
7. Expressions separated by + or blank are expressions with value equal to the sum of the components. At most one of the components may be relocatable.
8. Expressions separated by – form an expression with value equal to the difference to the components. If the right component is relocatable, the left component must be relocatable.
9. Expressions are evaluated left to right.

Names for registers are built in:

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

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

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

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

/ The addressed word is printed in octal.

\ The addressed byte is printed in octal.

" The addressed word is printed as two ASCII characters.

´ The addressed byte is printed as an ASCII character.

‘ The addressed word is printed in decimal.

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

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

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

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

% Exit.

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

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

There are two commands to interpret the value of expressions.

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

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

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

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

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

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

For some purposes, it is important to know how addresses typed by the user correspond with locations in the file being debugged. The mapping algorithm employed by *db* is non-trivial for two reasons: First, in an **a.out** file, there is a 20(8) byte header which will not appear when the file is loaded into core for execution. Therefore, apparent location 0 should correspond with actual file offset 20. Second, addresses in core images do not correspond with the addresses used by the program because in a core image there is a 512-byte header containing the system’s per-process data for the dumped process, and also because the stack is stored contiguously with the text and data part of the core image rather than at the highest possible locations. *Db* obeys the following rules:

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

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

If zero or two arguments are given, the mapping appropriate to a core image file is employed. This means that locations above the program break and below the stack effectively do not exist (and are not, in fact, recorded in the core file). Locations above the user's stack pointer are mapped, in looking at the core file, to the place where they are really stored. The per-process data kept by the system, which is stored in the first 512(10) bytes of the core file, cannot currently be examined (except by **$**).

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

SEE ALSO

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

DIAGNOSTICS

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

BUGS

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

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

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

number	The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore _ to input a negative number.
+ – * / % ^	The top two values on the stack are added (+), subtracted (–), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place.
s*x*	The top of the stack is popped and stored into a register named *x*, where *x* may be any character.
l*x*	The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value.
d	The top value on the stack is pushed on the stack. Thus the top value is duplicated.
p	The top value on the stack is printed. The top value remains unchanged.
f	All values on the stack and in registers are printed.
q	exits the program. If executing a string, the nesting level is popped by two.
x	treats the top element of the stack as a character string and executes it as a string of dc commands.
[...]	puts the bracketed ascii string onto the top of the stack.
<x =x >x	The top two elements of the stack are popped and compared. Register *x* is executed if they obey the stated relation.
v	replaces the top element on the stack by its square root.
!	interprets the rest of the line as a UNIX command.
c	All values on the stack are popped.
i	The top value on the stack is popped and used as the number radix for further input.
o	The top value on the stack is popped and used as the number radix for further output.
z	The stack level is pushed onto the stack.
?	A line of input is taken from the input source (usually the console) and executed.
new-line	ignored except as the name of a register or to end the response to a **?.**
space	ignored except as the name of a register or to terminate a number.

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

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

FILES

/etc/msh	to implement '!'

DIAGNOSTICS

(x) ? for unrecognized character x.
(x) ? for not enough elements on the stack to do what was asked by command x.
'Out of space' when the free list is exhausted (too many digits).
'Out of headers' for too many numbers being kept around.
'Out of pushdown' for too many items on the stack.
'Nesting Depth' for too many levels of nested execution.

BUGS

NAME

dsw – delete interactively

SYNOPSIS

dsw [directory]

DESCRIPTION

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

SEE ALSO

rm(I)

BUGS

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

NAME

du – summarize disk usage

SYNOPSIS

du [–**s**] [–**a**] [name ...]

DESCRIPTION

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

The optional argument –**s** causes only the grand total to be given. The optional argument –**a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

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

BUGS

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

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

NAME

echo – echo arguments

SYNOPSIS

echo [arg ...]

DESCRIPTION

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

BUGS

Echo with no arguments does not print a blank line.

NAME

ed – editor

SYNOPSIS

ed [–] [name]

DESCRIPTION

Ed is the standard text editor.

If a *name* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed's* buffer so that it can be edited. The optional – simulates an **os** command (see below) which suppresses the printing of characters counts by *e*, *r*, and *w* commands.

Ed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

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

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

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

1. An ordinary character (not one of those discussed below) is a regular expression and matches that character.
2. A circumflex '^' at the beginning of a regular expression matches the null character at the beginning of a line.
3. A currency symbol '$' at the end of a regular expression matches the null character at the end of a line.
4. A period '**.**' matches any character but a new-line character.
5. A regular expression followed by an asterisk '*' matches any number of adjacent occurrences (including zero) of the regular expression it follows.
6. A string of characters enclosed in square brackets '[ ]' matches any character in the string but no others. If, however, the first character of the string is a circumflex '^' the regular expression matches any character but new-line and the characters in the string.
7. The concatenation of regular expressions is a regular expression which matches the concatenation of the strings matched by the components of the regular expression.
8. The null regular expression standing alone is equivalent to the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced.

If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

Addresses are constructed as follows. To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line af-

fected 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 *n*-th line of the buffer.
5. ''*x*' addresses the line associated (marked) with the mark name character *x* which must be a printable character. Lines are marked with the *k* command described below.
6. A regular expression enclosed in slashes '/' addresses the first line found by searching toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the beginning of the buffer.
7. A regular expression enclosed in queries '?' addresses the first line found by searching toward the beginning of the buffer and stopping at the first line found containing a string matching the regular expression. If necessary the search wraps around to the end of the buffer.
8. An address followed by a plus sign '+' or a minus sign '–' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.

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

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

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

As mentioned, it is generally illegal for more than one command to appear on a line. However, any command may be suffixed by 'p' (for 'print'). In that case, the current line is printed after the command is complete.

(**.**)a
<text>
.

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

(**.**,**.**)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '**.**' is left at the last line input; if there were none, it is left at the first line not changed.

(**.**,**.**) d

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

e filename

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in. '**.**' is set to the last line of the buffer. The number of characters read is typed. 'filename' is remembered for possible use as a default file name in a subsequent *r* or *w* command.

f filename

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

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

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

(**.**) i
<text>
.

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

(**.**) k*x*

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

(**.**,**.**) m*a*

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

n

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

os
ov

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

(**.**,**.**) p

The print command prints the addressed lines. '**.**' is left at the last line printed. The *p* command *may* be placed on the same line after any command.

q

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

($) r filename

The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). The remembered file name is not changed unless 'filename' is the very first file name mentioned. Address '0' is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '**.**' is left at the last line read in from the file.

(**.**,**.**) s/regular expression/replacement/ or,
(**.**,**.**) s/regular expression/replacement/g

The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings

are replaced by the replacement specified, if the global replacement indicator 'g' appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of '/' to delimit the regular expression and the replacement. '**.**' is left at the last line substituted.

An ampersand '&' appearing in the replacement is replaced by the regular expression that was matched. The special meaning of '&' in this context may be suppressed by preceding it by '\'.

(1,$) v/regular expression/command list
This command is the same as the global command except that the command list is executed with '**.**' initially set to every line *except* those matching the regular expression.

(1,$) w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writeable by everyone). The remembered file name is *not* changed unless 'filename' is the very first file name mentioned. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). '**.**' is unchanged. If the command is successful, the number of characters written is typed.

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

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

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

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

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

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

FILES

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

DIAGNOSTICS

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

BUGS

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

-

NAME

exit – terminate command file

SYNOPSIS

exit

DESCRIPTION

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

SEE ALSO

if(I), goto(I), sh(I)

BUGS

NAME

fc – fortran compiler

SYNOPSIS

fc [−**c**] sfile1.f ... ofile1 ...

DESCRIPTION

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

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

Other arguments (except for −**c**) are assumed to be either loader flags, or object programs, typically produced by an earlier *fc* run, or perhaps libraries of Fortran-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out.**

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

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

1. Arbitrary combination of types is allowed in expressions. Not all combinations are expected to be supported at runtime. All of the normal conversions involving integer, real, double precision and complex are allowed.
2. DEC's **implicit** statement is recognized. E.g.: **implicit integer /i–n/**
3. The types doublecomplex, logical*1, integer*1, integer*2 and real*8 (double precision) are supported.
4. **&** as the first character of a line signals a continuation card.
5. **c** as the first character of a line signals a comment.
6. All keywords are recognized in lower case.
7. The notion of 'column 7' is not implemented.
8. G-format input is free form– leading blanks are ignored, the first blank after the start of the number terminates the field.
9. A comma in any numeric or logical input field terminates the field.
10. There is no carriage control on output.
11. A sequence of *n* characters in double quotes '"' is equivalent to *n* **h** followed by those characters.
12. In **data** statements, a hollerith string may initialize an array or a sequence of array elements.
13. The number of storage units requested by a binary **read** must be identical to the number contained in the record being read.

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

FILES

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

/lib/libf.a	builtin functions, etc.
/lib/liba.a	system library

SEE ALSO

ANSI standard, ld(I) for loader flags
Also see the writeups on the precious few non-standard Fortran subroutines, ierror and setfil (III)

DIAGNOSTICS

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

1 invalid log argument
2 bad arg count to amod
3 bad arg count to atan2
4 excessive argument to cabs
5 exp too large in cexp
6 bad arg count to cmplx
7 bad arg count to dim
8 excessive argument to exp
9 bad arg count to idim
10 bad arg count to isign
11 bad arg count to mod
12 bad arg count to sign
13 illegal argument to sqrt
14 assigned/computed goto out of range
15 subscript out of range
16 real**real overflow
17 (negative real)**real

100 illegal I/O unit number
101 inconsistent use of I/O unit
102 cannot create output file
103 cannot open input file
104 EOF on input file
105 illegal character in format
106 format does not begin with (
107 no conversion in format but non-empty list
108 excessive parenthesis depth in format
109 illegal format specification
110 illegal character in input field
111 end of format in hollerith specification
999 unimplemented input conversion

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

BUGS

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

arithmetic statement functions
scale factors on input

Backspace statement.

NAME

fed – edit associative memory for form letter

SYNOPSIS

fed

DESCRIPTION

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

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

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

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

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

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

q
returns to the system.

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

p causes any unaccounted-for string to be printed.

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

FILES

/tmp/ftmp?	temporary
form.m	associative memory

SEE ALSO

form(I), ed(I), sh(I)

WARNING

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

BUGS

NAME

file – determine format of file

SYNOPSIS

file files

DESCRIPTION

File will examine each of its arguments and give a guess as to the contents of the file. It is the only program that will give device numbers of special files.

BUGS

If the file is not instantly recognized, its type is given as ‘unknown’. There should be some heuristic to recognize source file ‘signatures’ in each of the standard languages.

NAME

form – form letter generator

SYNOPSIS

form proto arg ...

DESCRIPTION

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

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

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

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

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

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

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

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

FILES

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

SEE ALSO

fed(I), type(I), roff(I)

BUGS

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

NAME

goto – command transfer

SYNOPSIS

goto label

DESCRIPTION

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

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

SEE ALSO

sh(I)

BUGS

NAME

grep – search a file for a pattern

SYNOPSIS

grep [–**v**] [–**l**] [–**n**] expression [input] [output]

DESCRIPTION

Grep will search the input file (standard input default) for each line containing the regular expression. Normally, each line found is printed on the output file (standard output default). If the –**v** flag is used, all lines but those matching are printed. If the –**l** flag is used, each line printed is preceded by its line number. If the –**n** flag is used, no lines are printed, but the number of lines that would normally have been printed is reported. If interrupt is hit, the number of lines searched is printed.

For a complete description of the regular expression, see ed(I). Care should be taken when using the characters $ * [^ | ( ) and \ in the regular expression as they are also meaningful to the shell. (Precede them by \)

SEE ALSO

ed(I), sh(I)

BUGS

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

NAME

if – conditional command

SYNOPSIS

if expr command [arg ...]

DESCRIPTION

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

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

–**r** file	true if the file exists and is readable.
–**w** file	true if the file exists and is writable
s1 **=** s2	true if the strings *s1* and *s2* are equal.
s1 **!=** s2	true if the strings *s1* and *s2* are not equal.

These primaries may be combined with the following operators:

!	unary negation operator
–**a**	binary *and* operator
–**o**	binary *or* operator
(expr **)**	parentheses for grouping.

–**a** has higher precedence than –**o.** Notice that all the operators and flags are separate arguments to *if* and hence must be surrounded by spaces. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO

sh(I)

BUGS

NAME

kill – do in an unwanted process

SYNOPSIS

kill processid ...

DESCRIPTION

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

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

SEE ALSO

ps(I), sh(I)

BUGS

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

NAME

ld – link editor

SYNOPSIS

ld [–**sulxrnd**] name ...

DESCRIPTION

Ld combines several object programs into one; resolves external references; and searches libraries. In the simplest case the names of several object programs are given, and *d* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the –**r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out.** This file is executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

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

–**s** 'squash' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*.

–**u** take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

–**l** This option is an abbreviation for a library name. –**l** alone stands for '/lib/liba.a', which is the standard system library for assembly language programs. –**l***x* stands for '/lib/lib*x*.a' where *x* is any character. There are libraries for Fortran (*x* = **f**), and C (*x* = **c**). A library is searched when its name is encountered, so the placement of a –**l** is significant.

–**x** do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

–**r** generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols.

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

–**n** Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up the the first possible 4K word boundary following the end of the text.

FILES

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

SEE ALSO

as(I), ar(I)

BUGS

NAME

ln – make a link

SYNOPSIS

ln name1 [name2]

DESCRIPTION

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

Ln creates a link to an existing file *name1*. If *name2* is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of *name1*.

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

SEE ALSO

rm(I)

BUGS

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

NAME

login – sign onto UNIX

SYNOPSIS

login [username]

DESCRIPTION

The *login* command is used when a user initially signs onto UNIX, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See 'How to Get Started' for how to dial up initially.

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

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

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

FILES

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

SEE ALSO

init(VII), getty(VII), mail(I)

DIAGNOSTICS

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

BUGS

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

NAME

ls – list contents of directory

SYNOPSIS

ls [–**ltasdru**] name ...

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. There are several options:

–l list in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.)

–t sort by time modified (latest first) instead of by name, as is normal

–a list all entries; usually those beginning with '**.**' are suppressed

–s give size in blocks for each entry

–d if argument is a directory, list only its name, not its contents (mostly used with **–l** to get status on directory)

–r reverse the order of sort to get reverse alphabetic or oldest first as appropriate

–u use time of last access instead of last modification for sorting (**–t**) or printing (**–l**)

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

d if the entry is a directory;
b if the entry is a block-type special file;
c if the entry is a character-type special file;
– if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r if the file is readable
w if the file is writable
x if the file is executable
– if the indicated permission is not granted

Finally, the group-execute permission character is given as **s** if the file has set-group-ID mode; likewise the user-execute permission character is given as **s** if the file has set-user-ID mode.

FILES

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

BUGS

NAME

mail – send mail to another user

SYNOPSIS

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

DESCRIPTION

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

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

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

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

FILES

/etc/passwd	to identify sender and locate persons
mailbox	input mail
mbox	saved mail

SEE ALSO

login(I)

BUGS

The mail should be prepended rather than appended to the mailbox. The old mbox should not be destroyed when new mail is saved.

NAME

man – run off section of UNIX manual

SYNOPSIS

man [section] [title ...]

DESCRIPTION

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

man man

would reproduce this page.

FILES

/usr/man/man?/*

BUGS

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

NAME

merge – merge several files

SYNOPSIS

merge [–**anr**] [–*n*] [+*n*] [name ...]

DESCRIPTION

Merge merges several files together and writes the result on the standard output. If a file is designated by an unadorned '–', the standard input is understood.

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

Merge understands several flag arguments.

–**a** Use strict ASCII collating sequence.

–**n** An initial numeric string, possibly preceded by '–', is sorted by numerical value.

–**r** Data is in reverse order.

–*n* The first *n* fields in each line are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields (with –*n*) are skipped before characters.

SEE ALSO

sort(I)

BUGS

Only 8 files can be handled; any further files are ignored.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [**n**] [**y**]

DESCRIPTION

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

FILES

/dev/tty?

SEE ALSO

write(I)

DIAGNOSTICS

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

BUGS

NAME

mkdir – make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

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

SEE ALSO

rmdir(I)

BUGS

NAME

mv – move or rename a file

SYNOPSIS

mv name1 name2

DESCRIPTION

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

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

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

BUGS

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

NAME

nice – run a command at low priority

SYNOPSIS

nice command [arguments]

DESCRIPTION

Nice executes *command* at low priority.

SEE ALSO

nohup(I), nice(II)

BUGS

NAME

nm – print name list

SYNOPSIS

nm [–**cjnru**] [name]

DESCRIPTION

Nm prints the symbol table from the output file of an assembler or loader run. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined) **A** (absolute) **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), or **C** (common symbol). Global symbols have their first character underlined. Normally, the output is sorted alphabetically and symbols consisting of a letter followed by one or more digits are not printed (that is, symbols which look like C internal symbols).

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

Options are:

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

–**j** list symbols consisting of a letter followed by digits, which are normally suppressed.

–**n** sort by value instead of by name

–**r** sort in reverse order

–**u** print only undefined symbols.

FILES

a.out

BUGS

-

NAME

nohup – run a command immune to hangups

SYNOPSIS

nohup command [arguments]

DESCRIPTION

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

SEE ALSO

nice(I), signal(II)

BUGS

NAME

nroff – format text

SYNOPSIS

nroff [+*n*] [–*n*] [–**s**] [–**h**] [–**q**] [–**i**] files

DESCRIPTION

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

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

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

–**s** Stop prior to each page to permit paper loading. Printing is restarted by typing a ‘newline’ character.

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

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

–**i** Causes the standard input to be read after the files.

Nroff is more completely described in [1]. A condensed Request Summary is included here.

FILES

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

SEE ALSO

[1] NROFF User’s Manual, internal memorandum.

BUGS

REQUEST REFERENCE AND INDEX

Request Form	Initial Value	If no Argument	Cause Break	Explanation
I. Page Control				
.pl +N	N=66	N=66	no	Page Length.
.bp +N	N=1	-	yes	Begin Page.
.pn +N	N=1	ignored	no	Page Number.
.po +N	N=0	N=prev	no	Page Offset.
.ne N	-	N=1	no	NEed N lines.
II. Text Filling, Adjusting, and Centering				
.br	-	-	yes	BReak.
.fi	fill	-	yes	FIll output lines.
.nf	fill	-	yes	NoFill.
.ad c	adj,norm adjust		no	ADjust mode on.
.na	adjust	-	no	NoAdjust.
.ce N	off	N=1	yes	CEnter N input text lines.
III. Line Spacing and Blank Lines				
.ls +N	N=1	N=prev	no	Line Spacing.
.sp N	-	N=1	yes	SPace N lines
.lv N	-	N=1	no	LeaVe N lines
.sv N	-	N=1	no	SaVe N lines.
.os	-	-	no	Output Saved lines.
.ns	space	-	no	No-Space mode on.
.rs	-	-	no	Restore Spacing.
.xh	off	-	no	EXtra-Half-line mode on.
IV. Line Length and Indenting				
.ll +N	N=65	N=prev	no	Line Length.
.in +N	N=0	N=prev	yes	INdent.
.ti +N	-	N=1	yes	Temporary Indent.
V. Macros, Diversion, and Line Traps				
.de xx	-	ignored	no	DEfine or redefine a macro.
.ds xx	-	ignored	no	Define or redefine String.
.rm xx	-	-	no	ReMove macro name.
.di xx	-	end	no	DIvert output to macro "xx".
.wh -N xx	-	-	no	WHen; set a line trap.
.ch xx y	-	-	no	CHange trap line.
.ch -N -M	-	-	no	"
.ch xx -M	-	-	no	"
.ch -N y	-	-	no	"
VI. Number Registers				
.nr ab +N -M	-	no	Number Register.	
.nr a +N -M	-	no		"
.nc c	\n	\n	no	Number Character.
.ar	arabic	-	no	Arabic numbers.
.ro	arabic	-	no	Roman numbers.
.RO	arabic	-	no	ROMAN numbers.
VII. Input and Output Conventions and Character Translations				
.ta N,M,...	-	none	no	PseudoTAbs setting.
.tc c	space	space	no	Tab replacement Character.
.lc c	.	.	no	Leader replacement Character.
.ul N	-	N=1	no	UNderline input text lines.

.cc c	.	.	no	Basic Control Character.
.c2 c	'	'	no	Nobreak control character.
.ec c	-	\	no	Escape Character.
.li N	-	N=1	no	Accept input lines LIterally.
.tr abcd....	-	-	no	TRanslate on output.

VIII. Hyphenation.

.nh	on	-	no	No Hyphen.
.hy	on	-	no	HYphenate.
.hc c	none	none	no	Hyphenation indicator Character.

IX. Three Part Titles.

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

X. Output Line Numbering.

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

XI. Conditional Input Line Acceptance

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

XII. Environment Switching.

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

XIII. Insertions from the Standard Input Stream

.rd prompt	-	bell	no	ReaD insert.
.ex	-	-	no	EXit.

XIV. Input File Switching

.so filename	-	-	no	Switch SOurce file (push down).
.nx filename	-	no	NeXt file.	

XV. Miscellaneous

.tm mesg	-	-	no	Typewriter Message
.ig	-	-	no	IGnore.
.fl	-	-	no	FLush output buffer.
.ab	-	-	no	ABort.

NAME

od – octal dump

SYNOPSIS

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

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing –**o** is default. The meanings of the format argument characters are:

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

b interprets bytes in octal.

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

d interprets words in decimal.

h interprets words in hex.

o interprets words in octal.

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

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If '**.**' is appended, the offset is interpreted in decimal. If '**b**' is appended, the offset is interpreted in blocks. (A block is 512 bytes.) If the file argument is omitted, the offset argument must be preceded by '**+**'.

Dumping continues until end-of-file.

SEE ALSO

db(I)

BUGS

NAME

opr – off line print

SYNOPSIS

opr [––] [–] [+] [+–]file ...

DESCRIPTION

Opr will arrange to have the 201 data phone daemon submit a job to the Honeywell 6070 to print the file arguments. Normally, the output appears at the GCOS central site. If the first argument is ––**,** the output is remoted to station R1, which has an IBM 1403 printer.

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

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

FILES

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

SEE ALSO

dpd(I), passwd(V)

BUGS

There should be a way to specify a general remote site.

NAME

passwd – set login password

SYNOPSIS

passwd name password

DESCRIPTION

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

FILES

/etc/passwd

SEE ALSO

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

BUGS

NAME

pfe – print floating exception

SYNOPSIS

pfe

DESCRIPTION

Pfe will examine the floating point exception register and print a diagnostic for the last floating point exception.

SEE ALSO

signal(II)

BUGS

Since there is but one floating point exception register and it cannot be saved and restored by the system, the floating exception that is printed is the one that occured system wide. Floating exceptions are therefore volatile.

NAME

plot – make a graph

SYNOPSIS

plot [option] ...

DESCRIPTION

Plot takes pairs of numbers from the standard input as abscissas and ordinates of a graph. The graph is plotted on the storage scope, /dev/vt0.

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

a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

c Place character string given by next argument at each point.

d Omit connections between points. (Disconnect.)

g*n* Grid style:
n=0, no grid
n=1, axes only
n=2, complete grid (default).

s Save screen, don't erase before plotting.

x Next 1 (or 2) arguments are lower (and upper) *x* limits.

y Next 1 (or 2) arguments are lower (and upper) *y* limits.

Points are connected by straight line segments in the order they appear in input. If a specified lower limit exceeds the upper limit, or if the automatic increment is negative, the graph is plotted upside down. Automatic abscissas begin with the lower *x* limit, or with 0 if no limit is specified. Grid lines and automatically determined limits fall on round values, however roundness may be subverted by giving an inappropriately rounded lower limit. Plotting symbols specified by **c** are placed so that a small initial letter, such as + o x, will fall approximately on the plotting point.

FILES

/dev/vt0

SEE ALSO

spline(VI)

BUGS

A limit of 1000 points is enforced silently.

NAME

pr − print file

SYNOPSIS

pr [−**h** name] [−*n*] [+*n*] [file ...]

DESCRIPTION

Pr produces a printed listing of one or more files. The output is separated into pages headed by a date, the name of the file or a header (if any), and the page number. If there are no file arguments, *pr* prints the standard input file, and is thus usable as a filter.

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

−*n* produce *n*-column output

+*n* begin printing with page *n*.

−**h** treat the next argument as a header

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

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(I), cp(I)

DIAGNOSTICS

none (files not found are ignored)

BUGS

It would be nice to be able to set the number of lines per page.

NAME

proof – compare two text files

SYNOPSIS

proof oldfile newfile

DESCRIPTION

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

SEE ALSO

cmp(I), comm(I)

DIAGNOSTICS

yes, but they are undecipherable, e.g. '?1'.

BUGS

This program has a long way to go before even a list of specific bugs is appropriate.

NAME

ps – process status

SYNOPSIS

ps [**alx**]

DESCRIPTION

Ps prints certain indicia about active processes. The **a** flag asks for information about all processes with teletypes (ordinarily only one's own processes are displayed); **x** asks even about processes with no typewriter; **l** asks for a long listing. Ordinarily only the typewriter number (if not one's own) and the process number are given.

The long listing is columnar and contains

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

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

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

The last character of the control typewriter of the process.

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

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

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

Unfortunately if you have forgotten the number of a process you will have to guess which one it is. Plain *ps* will tell you only a list of numbers.

FILES

/usr/sys/unix	system namelist
/dev/mem	resident system

SEE ALSO

kill(I)

BUGS

The ability to see, even if dimly, the name by which the process was invoked would be welcome.

NAME

rew – rewind tape

SYNOPSIS

rew [[**m**]digit]

DESCRIPTION

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

FILES

/dev/tap?
/dev/mt?

BUGS

NAME

rm – remove (unlink) files

SYNOPSIS

rm [–**f**] [–**r**] name ...

DESCRIPTION

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

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

If a designated file is a directory, an error comment is printed unless the optional argument –**r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory. To remove directories *per se* see rmdir(I).

FILES

/etc/glob to implement the –**r** flag

SEE ALSO

rmdir(I)

BUGS

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

NAME

rmdir – remove directory

SYNOPSIS

rmdir dir ...

DESCRIPTION

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

BUGS

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

NAME

roff – format text

SYNOPSIS

roff [+*n*] [–*n*] [–**s**] [–**h**] file ...

DESCRIPTION

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

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

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

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

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

A Request Summary is attached.

FILES

/usr/lib/suftabsuffix hyphenation tables
/tmp/rtm?temporary

SEE ALSO

nroff (I), troff (I)

BUGS

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

REQUEST SUMMARY

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

NAME

sh – shell (command interpreter)

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

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

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

For example, either of the command lines

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

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

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

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

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

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

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

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

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

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

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

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

```
as /usr/dmr/a?.s
```

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

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

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

```
ls | pr –h "My directory"
```

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

to *pr*.

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

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

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

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

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

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

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

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

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

shift is done by manipulating the arguments to the Shell.

'**:**' is simply ignored.

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

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

FILES

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

SEE ALSO

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

BUGS

When output is redirected, particularly to make a multicommand pipeline, diagnostics tend to be sent down the pipe and are sometimes lost altogether. Not all components of a pipeline swawned with '&' ignore interrupts.

NAME

shift – adjust Shell arguments

SYNOPSIS

shift

DESCRIPTION

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

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

prints each of its arguments in 3-column format.

Shift is executed within the Shell.

SEE ALSO

sh (I)

BUGS

NAME

size – size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

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

BUGS

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

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

```
(sleep 105; command)&
```

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

```
: loop
command
sleep 37
goto loop
```

SEE ALSO

sleep(II)

BUGS

Time must be less than 65536 seconds.

NAME

sno – Snobol interpreter

SYNOPSIS

sno [file]

DESCRIPTION

Sno is a Snobol III (with slight differences) compiler and interpreter. *Sno* obtains input from the concatenation of *file* and the standard input. All input through a statement containing the label 'end' is considered program and is compiled. The rest is available to 'syspit'.

Sno differs from Snobol III in the following ways.

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

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

There is no back referencing.

```
x = "abc"
a *x* x         is an unanchored search for 'abc'
```

Function declaration is different. The function declaration is done at compile time by the use of the label 'define'. Thus there is no ability to define functions at run time and the use of the name 'define' is preempted. There is also no provision for automatic variables other than the parameters. For example:

definef( )

or

define f(a,b,c)

All labels except 'define' (even 'end') must have a non-empty statement.

If 'start' is a label in the program, program execution will start there. If not, execution begins with the first executable statement. 'define' is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators '/' and '*' must be set off by space.

The right side of assignments must be non-empty.

Either ´ or " may be used for literal quotes.

The pseudo-variable 'sysppt' is not available.

SEE ALSO

Snobol III manual. (JACM; Vol. 11 No. 1; Jan 1964; pp 21)

BUGS

NAME

sort – sort a file

SYNOPSIS

sort [–**anr**] [+*n*] [–*n*] [input [output]]

DESCRIPTION

Sort sorts *input* and writes the result on *output.* If the output file is not given, the standard output is used. If the input file is missing, the standard input is used. Thus *sort* may be used as a filter. The input and output file may be the same.

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

Sort understands several flag arguments.

–**a** Use strict ASCII collating sequence.

–**n** An initial numeric string is sorted by numerical value.

–**r** Output is in reverse order.

–*n* The first *n* fields in each line are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored in the sort. Fields (with –*n*) are skipped before characters.

FILES

/tmp/stm?

BUGS

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

NAME

speak – word to voice translator

SYNOPSIS

speak [–**epsv**] [vocabulary [output]]

DESCRIPTION

Speak turns a stream of words into utterances and outputs them to a voice synthesizer, or to a specified output file. It has facilities for maintaining a vocabulary. It receives, from the standard input

- working lines: text of words separated by blanks
- phonetic lines: strings of phonemes for one word preceded and separated by commas. The phonemes may be followed by comma-percent then a 'replacement part' – an ASCII string with no spaces. The phonetic code is given in vsp(VII).
- empty lines
- command lines: beginning with **!.** The following command lines are recognized:

!r file	replace coded vocabulary from file
!w file	write coded vocabulary on file
!p	print parsing for working word
!l	list vocabulary on standard output with phonetics
!c word	copy phonetics from working word to specified word
!d	print phonetics for working word

Each working line replaces its predecessor. Its first word is the 'working word'. Each phonetic line replaces the phonetics stored for the working word. In particular, a phonetic line of comma only deletes the entry for the working word. Each working line, phonetic line or empty line causes the working line to be uttered. The process terminates at the end of input.

Unknown words are pronounced by rules, and failing that, are spelled. Spelling is done by taking each character of the word, prefixing it with *, and looking it up. Unspellable words burp.

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

A set of single letter options may appear in any order preceded by **–.** Their meanings are:

–e	suppress English steps (4–8) below
–p	suppress pronunciation by rule
–s	suppress spelling
–v	suppress voice output

The steps of pronunciation by rule are:

(1) If there were no lower case letters in the working line, fold all upper case letters to lower.
(2) Fold an initial cap to lower case, and try again.
(3) If word has only one letter, or has no lower case vowels, quit.
(4) If there is a final *s*, strip it.
(5) Replace final –ie by –y.
(6) If any changes have been made, try whole word again.
(7) Locate probable long vowels and capitalize them. Mark probable silent *e*'s.
(8) Put back the *s* stripped in (4), if any.
(9) Place # before and after word.
(10) Prefix word with *%*, and look up longest initial match in the stored table of words; if none, quit.
(11) Use phonemes from the stored phonetic string as pronunciation, and replace the matched stuff by the replacement part of the phonetic string.
(12) If anything remains, go to (10).

Long vowels are located this way in step (7):

(1) A *u* appearing in context [ˆaeiou]u[ˆaeiouwxy][aieouy]. (The notation is just a regular expression à la ed(I).) *(pustUlous)*

(2) One of [aeo] appearing in the context [aeo][ˆaehiouwxy][ie][aou] or in the context [aeo][ˆaehiouwxy]ien is assumed long. The digram *th* behaves as a single letter in this test. *(rAdium, facEtious, quOtient, carpAthian)*

(3) If the first vowel in the word is *i* followed by one of *aou*, it is assumed long. *(Iodine, dIameter, trIumph)*

(4) If the only vowel in the word is final *e*, the vowel is assumed long. *(bE, shE)*

(5) If the only vowels in the word appear in the pattern [aeiouy][ˆaeiouwxy]S, where S is one of the suffixes

–al	–le	–re	–y

then the first vowel is assumed long. *(glObal, tAble, lUcre, lAdy)*

(6) If no suffix was found in (5), as many of these suffixes as possible are isolated from right to left. Stripping stops when *e* has been stripped, nor is *e* stripped before a suffix beginning with *e*. Each suffix is marked by inserting ⏐ just before the first letter, or just after *e* in those suffixes that begin with *e*.

–able	–ably	–e	–ed
–er	–ery	–est	–ful
–ing	–less	–ment	–ness

(care⏐ful⏐ly, maj⏐or, fine⏐ry, state⏐, caree⏐r)

(7) If the word, exclusive of suffixes, ends in *i* or *y*, and contains no earlier vowel, then *i* or *y* is assumed long. *(pY* (from pie), *crY⏐ing, lIe⏐d)*

(8) If the first suffix begins with one of [aeio], then the vowel [aeiouy] in an immediately preceding pattern [ˆaeo][aeiouy][ˆaeiouwxy] is assumed long. The digram *th* behaves as a single letter in this test. *(cAre⏐ful⏐ly, bAthe⏐d, mAj⏐or, pOt⏐able, port⏐able)*

(9) In these exceptional cases no long letter is assumed in the preceding step:
(i) before *g*, if there are any earlier vowels *(postage⏐, stAge⏐, college⏐)*
(ii) *e* is not long before *l (travele⏐d)*

(10) If the first suffix begins with one of [aeio], and the word exclusive of suffixes ends in [aeiouyAEIOUY]th, then digram *th* is capitalized. *(breaTH⏐ing, blITHe⏐ly)*

(11) An attempt is made to recognize silent *e* in the middle of compound words. Such an *e* is marked by a following ⏐, and preceding vowels, other than *e*, are assumed long as in step (8). Silent *e* is marked in the context [bdgmnprst][bdgpt]le[ˆaeioruy⏐]S, where S is any string that contains [aeiouy] but does not contain ⏐ or the end of the word. Silent *e* is also marked in the context [ˆaeiu][aiou][ˆaeiouwxy]e[ˆaeinoruy]S. *(simple⏐ton, fAce⏐guard, cAve⏐man, cavernous)*

FILES

/usr/lib/speak.m

SEE ALSO

vs(VII), vs(IV)

DIAGNOSTICS

'?' for unknown command with **!,** or for unreadable or unwritable vocabulary file

BUGS

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

NAME

split – split a file into pieces

SYNOPSIS

split [file1 [file2]]

DESCRIPTION

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

If no input file is given, or the first argument is '–', then the standard input file is used.

BUGS

Watch out for 14-character file names. The number of lines per file should be an argument.

NAME

strip – remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

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

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

FILES

/tmp/stm? temporary file

SEE ALSO

ld(I), as(I)

BUGS

NAME

stty – set teletype options

SYNOPSIS

stty option ...

DESCRIPTION

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

even	allow even parity
–even	disallow even parity
odd	allow odd parity
–odd	disallow odd parity
raw	raw mode input (no erase, kill, interrupt, quit, EOT; parity bit passed back)
–raw	negate raw mode
–nl	allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl	accept only new-line to end lines
echo	echo back every character typed
–echo	do not echo characters
lcase	map upper case to lower case
–lcase	do not map case
–tabs	replace tabs by spaces in output
tabs	preserve tabs
delay	calculate cr, tab, and form-feed delays
–delay	no cr/tab/ff delays
tdelay	calculate tab delays
–tdelay	no tab delays

SEE ALSO

stty(II)

BUGS

There should be ‘package’ options such as **execuport, 33,** or **terminet.**

NAME

sum – sum file

SYNOPSIS

sum name ...

DESCRIPTION

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

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

BUGS

NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

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

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

BUGS

Notice that **time x >y** puts the timing information into *y*. One can get around this by **time sh** followed by **x >y.**

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tp − manipulate DECtape and magtape

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

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

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

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

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

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

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

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

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

m Specifies magtape as opposed to DECtape.

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

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

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

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

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

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

FILES

/dev/tap?
/dev/mt?

-

DIAGNOSTICS

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

BUGS

NAME

tr – transliterate

SYNOPSIS

tr [–**cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. If *string2* is short, it is padded with corresponding characters from *string1*. Any combination of the options –**cds** may be used. –**c** complements the set of characters in *string1* with respect to the universe of characters whose ascii codes are 001 through 377 octal. –**d** deletes all input characters not in *string1*. –**s** squeezes all strings of repeated output characters that are in *string2* to single characters.

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

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

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

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

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

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

SEE ALSO

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

BUGS

Won't handle ascii NUL.
Also, Kernighan's Lemma can really bite you; try looking for strings which have \ and * in them.

NAME

troff – format text

SYNOPSIS

troff [+*n*] [–*n*] [–**t**] [–**f**] [–**w**] [–**i**] [–**a**] files

DESCRIPTION

Troff formats text for a Graphic Systems phototypesetter according to control lines embedded in the text files. *Troff* is based on nroff(I). The non-file option arguments are interpreted as follows:

- **+***n* Commence typesetting at the first page numbered *n* or larger.
- **–***n* Stop after page *n.*
- **–t** Place output on standard output instead of the phototypesetter.
- **–f** Refrain from feeding out paper and stopping the phototypesetter at the end.
- **–w** Wait until phototypsetter is available, if currently busy.
- **–i** Read from standard input after the files have been exhausted.
- **–a** Send a printable approximation of the results to the standard output.

A TROFF Guide is available [1] which can be used in conjunction with an NROFF Manual [2].

FILES

/usr/lib/suftabsuffix hyphenation tables
/tmp/rtm?temporary

SEE ALSO

[1] Preliminary TROFF Guide (unpublished).
[2] NROFF User's Manual (internal memorandum).
TROFF Made Trivial (unpublished).
nroff(I), roff(I)

BUGS

NAME

tss – interface to MH-TSS

SYNOPSIS

tss

DESCRIPTION

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

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

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

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

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

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

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

FILES

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

DIAGNOSTICS

Most often, 'Transmission error on last message.'

BUGS

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

-

NAME

tty – get typewriter name

SYNOPSIS

tty

DESCRIPTION

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

DIAGNOSTICS

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

BUGS

NAME

type – type on 2741

SYNOPSIS

type file ...

DESCRIPTION

Type copies its input files to the fixed output port **ttyc** converting to 2741 EBCDIC output code. Before each new page (66 lines) and before each new file, *type* stops and reads the 2741 before continuing. This allows time for insertion of single sheet paper. To continue, push the ATTN key on the 2741.

FILES

/dev/ttyc

BUGS

Since it is impossible to second guess a 2741, quite often it is necessary to print a # to put this device in a state it might already be in.
The value of padding out a page with up to 66 carriage returns is doubtful.

NAME

typo – find possible typos

SYNOPSIS

typo [–] $file_1$ ...

DESCRIPTION

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

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

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

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

FILES

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

BUGS

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

The expanded escape sequences of *troff* are not correctly recognized.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [–**udc** [+n] [–n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see sort(I). If the –**u** flag is used, just the lines that are not repeated in the original file are output. The –**d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the –**u** and –**d** mode outputs.

The –**c** option supersedes –**u** and –**d** and generates an output report in the style of –**ud** but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(I), comm(I)

BUGS

NAME

wait – await completion of process

SYNOPSIS

wait

DESCRIPTION

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

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

SEE ALSO

sh(I)

BUGS

After executing *wait* there is no way to interrupt the processes waited on. This is because interrupts were set to be ignored when the process was created. The only out (if the process does not terminate) is to *kill* the process from another terminal or to hangup.

NAME

wc – get (English) word count

SYNOPSIS

wc files

DESCRIPTION

Wc provides a count of the words, text lines, and control lines for each argument file. If no files are provided, *wc* reads the standard input.

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

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

DIAGNOSTICS

none; arguments not found are ignored.

BUGS

NAME

who – who is on the system

SYNOPSIS

who [who-file]

DESCRIPTION

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

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

Each login is listed with user name, typewriter name (with '/dev/' suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in the place of the device name, and a fossil time indicative of when the system went down.

FILES

/tmp/utmp

SEE ALSO

login(I), init(VII)

BUGS

NAME

write – write to another user

SYNOPSIS

write user [ttyno]

DESCRIPTION

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

message from yourname...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the typewriter or an interrupt is sent. At that point *write* writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *ttyno* argument may be used to indicate the last character of the appropriate typewriter name.

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

If the character '!' is found at the beginning of a line, *write* calls the mini-shell *msh* to execute the rest of the line as a command.

The following protocol is suggested for using *write:* when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a distinctive signal (**(o)** for 'over' is conventional) that the other may reply. **(oo)** (for 'over and out') is suggested when conversation is about to be terminated.

FILES

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

SEE ALSO

mesg(I), who(I)

BUGS