-

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** creates 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 are replaced by the modified version.

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

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

c copy
a append
d delete
r replace
x extract

FILES

/tmp/vtm? temporary

SEE ALSO

ld (I), archive (V)

BUGS

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

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

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

For the same reason, only the first 8 characters of file names are significant.

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

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

/lib/as2	pass 2 of the assembler
/tmp/atm[1-3]?	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.

comment ...
This statement is ignored. It is used to interject commentary in a program.

done
Return to system level.

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

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

dump
The name and current value of every variable is printed.

edit
The UNIX editor, *ed,* is invoked with the *file* argument. After the editor exits, this file is recompiled.

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
if expression
...
[**else**

...]
fi
The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. In the second form, an optional **else** allows for a group of statements to be executed when the first group is not.

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

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

prompt list
Prompt is the same as *print* except that no newline character is printed.

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

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

save [expression [expression]]
Save is like *list* except that the output is written on the *file* argument. If no argument is given on the command, **b.out** is used.

Expressions have the following syntax:

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

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

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

_ expression
The result is the negation of the expression.

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

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

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

The following is the list of operators:

=

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

& |

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

< <= > >= == <>

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

+ −

  Add and subtract.

* /

  Multiply and divide.

^

  Exponentiation.

The following is a list of builtin functions:

arg(i)

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

exp(x)

  is the exponential function of *x*.

log(x)

  is the natural logarithm of *x*.

sqr(x)

  is the square root 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.

abs(x)

  is the absolute value of *x*.

int(x)

  returns *x* truncated (towards 0) to an integer.

FILES

/tmp/btm?	temporary
b.out	save file

DIAGNOSTICS

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

-

BUGS

Has been known to give core images.

NAME

bc − arbitrary precision interactive language

SYNOPSIS

bc [**−l**] [file ...]

DESCRIPTION

Bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The '−l' argument stands for the name of a library of mathematical subroutines which contains sine (named 's'), cosine ('c'), arctangent ('a'), natural logarithm ('l'), and exponential ('e'). The syntax for *bc* programs is as follows; E means expression, S means statement.

Comments
    are enclosed in /* and */.

Names
    letters a−z
    array elements: letter[E]
    The words 'ibase', 'obase', and 'scale'

Other operands
    arbitrarily long numbers with optional sign and decimal point.
    (E)
    sqrt (E)
    <letter> (E , ... , E)

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

Statements
```
E
{ S ; ... ; S }
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
```

Function definitions are exemplified by
```
define <letter> ( <letter> ,..., <letter> ) {
        auto <letter>, ... , <letter>
        S; ... S
        return ( E )
}
```

All function arguments are passed by value.

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

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

For example

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

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

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

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

FILES

/usr/lib/lib.b mathematical library

SEE ALSO

dc (I), C Reference Manual, ‘‘BC – An Arbitrary Precision Desk-Calculator Language.’’

BUGS

No &&, | | yet.
for statement must have all three E’s
quit is interpreted when read, not when executed.

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

prints the file, and

cat file1 file2 >file3

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

If no input file is given, or if the argument '–' is encountered, *cat* reads from the standard input file.

SEE ALSO

pr(I), cp(I)

DIAGNOSTICS

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

BUGS

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

-

NAME

cc – C compiler

SYNOPSIS

cc [**–c**] [**–p**] [**–f**] [**–O**] [**–S**] [**–P**] file ...

DESCRIPTION

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

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

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

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

–p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls the *monitor* subroutine (III) at the start and arranges to write out a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof* (I).

–f In systems without hardware floating-point, use a version of the C compiler which handles floating-point constants and loads the object program with the floating-point interpreter. Do not use if the hardware is present.

–O Invoke an object-code optimizer.

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

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

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

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/lib/c[01]	compiler
/lib/fc[01]	floating-point compiler
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	runtime startoff of profiling
/lib/fcrt0.o	runtime startoff for floating-point interpretation
/lib/libc.a	C library; see section III.
/lib/liba.a	Assembler library used by some routines in libc.a

SEE ALSO

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

DIAGNOSTICS

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

BUGS

-

NAME

cdb – C debugger

SYNOPSIS

cdb [a.out [core]]

DESCRIPTION

Cdb is a debugger for use with C programs. It is useful for both post-mortem and interactive debugging. An important feature of *cdb* is that even in the interactive case no advance planning is necessary to use it; in particular it is not necessary to compile or load the program in any special way nor to include any special routines in the object file.

The first argument to *cdb* is an object program, preferably containing a symbol table; if not given ‘‘a.out’’ is used. The second argument is the name of a core-image file; if it is not given, ‘‘core’’ is used. The core file need not be present.

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

Addresses are expressions composed of names, decimal numbers, and octal numbers (which begin with ‘‘0’’) and separated by ‘‘+’’ and ‘‘−’’. Evaluation proceeds left-to-right.

Names of external variables are written just as they are in C. For various reasons the external names generated by C all begin with an underscore, which is automatically tacked on by *cdb.* Currently it is not possible to suppress this feature, so symbols (defined in assembly-language programs) which do not begin with underscore are inaccessible.

Variables local to a function (automatic, static, and arguments) are accessible by writing the name of the function, a colon ‘‘:’’, and the name of the local variable (e.g. ‘‘main:argc’’). There is no notion of the ‘‘current’’ function; its name must always be written explicitly.

A number which begins with ‘‘0’’ is taken to be octal; otherwise numbers are decimal, just as in C. There is no provision for input of floating numbers.

The construction ‘‘name[expression]’’ assumes that *name* is a pointer to an integer and is equivalent to the contents of the named cell plus twice the expression. Notice that *name* has to be a genuine pointer and that arrays are not accessible in this way. This is a consequence of the fact that types of variables are not currently saved in the symbol table.

The command characters are:

/m print the addressed words. *m* indicates the mode of printout; specifying a mode sets the mode until it is explicitly changed again:

- **o** octal (default)
- **i** decimal
- **f** single-precision floating-point
- **d** double-precision floating-point

\ Print the specified bytes in octal.

= print the value of the addressed expression in octal.

´ print the addressed bytes as characters. Control and non-ASCII characters are escaped in octal.

" take the contents of the address as a pointer to a sequence of characters, and print the characters up to a null byte. Control and non-ASCII characters are escaped as octal.

& Try to interpret the contents of the address as a pointer, and print symbolically where the pointer points. The typeout contains the name of an external symbol and, if required, the smallest possible positive offset. Only external symbols are considered.

-

? Interpret the addressed location as a PDP-11 instruction.

$*m* If no *m* is given, print a stack trace of the terminated or stopped program. The last call made is listed first; the actual arguments to each routine are given in octal. (If this is inappropriate, the arguments may be examined by name in the desired format using "/".) If *m* is "r", the contents of the PDP-11 general registers are listed. If *m* is "f", the contents of the floating-point registers are listed. In all cases, the reason why the program stopped or terminated is indicated.

%*m* According to *m,* set or delete a breakpoint, or run or continue the program:

b An address within the program must be given; a breakpoint is set there. Ordinarily, breakpoints will be set on the entry points of functions, but any location is possible as long as it is the first word of an instruction. (Labels don't appear in the symbol table yet.) Stopping at the actual first instruction of a function is undesirable because to make symbolic printouts work, the function's save sequence has to be completed; therefore *cdb* automatically moves breakpoints at the start of functions down to the first real code.

It is impossible to set breakpoints on pure-procedure programs (−n flag on cc or ld) because the program text is write-protected.

d An address must be given; the breakpoint at that address is removed.

r Run the program being debugged. Following the "%r", arguments may be given; they cannot specify I/O redirection (">", "<") or filters. No address is permissible, and the program is restarted from scratch, not continued. Breakpoints should have been set if any were desired. The program will stop if any signal is generated, such as illegal instruction (including simulated floating point), bus error, or interrupt (see signal(II)); it will also stop when a breakpoint occurs and in any case announce the reason. Then a stack trace can be printed, named locations examined, etc.

c Continue after a breakpoint. It is possible but probably useless to continue after an error since there is no way to repair the cause of the error.

SEE ALSO

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

BUGS

Use caution in believing values of register variables at the lowest levels of the call stack; the value of a register is found by looking at the place where it was supposed to have been saved by the callee.

Some things are still needed to make *cdb* uniformly better than *db:* non-C symbols, patching files, patching core images of programs being run. It would be desirable to have the types of variables around to make the correct style printout more automatic. Structure members should be available.

Naturally, there are all sorts of neat features not handled, like conditional breakpoints, optional stopping on certain signals (like illegal instructions, to allow breakpointing of simulated floating-point programs).

-

NAME

chdir – change working directory

SYNOPSIS

chdir directory

DESCRIPTION

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

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

SEE ALSO

sh (I), pwd (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
1000	sticky bit for shared, pure-procedure programs (see below)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

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

If an executable file is set up for sharing (‘‘−n’’ option of *ld (I)*), then mode 1000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time. Ability to set this bit is restricted to the super-user since swap space is consumed by the images; it is only worth while for heavily used commands.

SEE ALSO

ls (I), chmod (II)

BUGS

NAME

cmp – compare two files

SYNOPSIS

cmp [–**l**] [–**s**] file1 file2

DESCRIPTION

The two files are compared. (If *file1* is '–', the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted. Moreover, return code 0 is yielded for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

Options:

–l Print the byte number (decimal) and the differing bytes (octal) for each difference.

–s Print nothing for differing files; return codes only.

SEE ALSO

diff (I), comm (I)

BUGS

-

NAME

comm − print lines common to two files

SYNOPSIS

comm [− [**123**]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2,* which should be in sort, and produces a three column output: lines only in *file1;* lines only in *file2;* and lines in both files. The filename '−' means the standard input.

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

SEE ALSO

cmp (I), diff (I)

BUGS

NAME

cp – copy

SYNOPSIS

cp 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*.

It is forbidden to copy a file onto itself.

SEE ALSO

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

BUGS

-

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, the next argument is taken to be an *ignore* file; if the **−o** option is given, the next argument is taken to be an *only* file. *Ignore* and *only* files are lists of symbols separated by new lines. All symbols in an *ignore* file are ignored in columns (1) and (3) of the output. If an *only* file is given, only symbols in that file appear in column (1). At most one of **−i** and **−o** may be used. The default setting is **−i.** Assembler predefined symbols or C keywords are ignored.

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

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

Options:

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

FILES

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

/bin/sort used to sort temporaries

SEE ALSO

as (I), cc (I)

BUGS

-

NAME

date − print and set the date

SYNOPSIS

date [**s**] [mmddhhmm[yy]]

DESCRIPTION

If no argument is given, the current date and time are printed. 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.

If the argument is ''s,'' *date* calls the network file store via the TIU interface (if present) and sets the clock to the time thereby obtained.

DIAGNOSTICS

''No permission'' if you aren't the super-user and you try to change the date; ''bad conversion'' if the date set is syntactically incorrect.

FILES

/dev/tiu/d0

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.

ˆ 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 ˆ requests is by one or two depending on whether the last command was word or byte oriented.

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

There are two commands to interpret the value of expressions.

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

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

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

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

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

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

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

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

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

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

-

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

SEE ALSO

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

DIAGNOSTICS

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

BUGS

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

-

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

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

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

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

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

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

d
: The top value on the stack is duplicated.

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

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

q
: exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

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

[...]
: puts the bracketed ascii string onto the top of the stack.

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

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

!
: interprets the rest of the line as a 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.

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

z
: The stack level is pushed onto the stack.

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

-

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

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

SEE ALSO

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

DIAGNOSTICS

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

BUGS

NAME

dd − convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=	input block size (default 512)
obs=	output block size (default 512)
bs=	set both input and output block size, superseding *ibs* and *obs;* also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

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

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

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

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

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

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

SEE ALSO

cp (I)

BUGS

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

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

NAME

diff − differential file comparator

SYNOPSIS

diff [−] name1 name2

DESCRIPTION

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

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

These lines resemble *ed* commands to convert file *name1* into file *name2.* The numbers after the letters pertain to file *name2.* In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert file *name2* into *name1.* As in *ed,* identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by '*', then all the lines that are affected in the second file flagged by '**.**'.

Under the − option, the output of *diff* is a script of *a, c* and *d* commands for the editor *ed,* which will change the contents of the first file into the contents of the second. In this connection, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

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

Except for occasional 'jackpots', *diff* finds a smallest sufficient set of file differences.

SEE ALSO

cmp (I), comm (I), ed (I)

DIAGNOSTICS

'jackpot' − To speed things up, the program uses hashing. You have stumbled on a case where there is a chance that this has resulted in a difference being called where none actually existed. Sometimes reversing the order of files will make a jackpot go away.

BUGS

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

NAME

dsw − delete interactively

SYNOPSIS

dsw [directory]

DESCRIPTION

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

SEE ALSO

rm (I)

BUGS

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

-

NAME

du − summarize disk usage

SYNOPSIS

du [−**s**] [−**a**] [name ...]

DESCRIPTION

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

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

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

BUGS

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

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

-

NAME

echo − echo arguments

SYNOPSIS

echo [arg ...]

DESCRIPTION

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

BUGS

-

NAME

ed − text editor

SYNOPSIS

ed [−] [name]

DESCRIPTION

Ed is the standard text editor.

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

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

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

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

Ed supports a limited form of *regular expression* notation. A regular expression 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 empty string 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 except 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 except 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. A regular expression enclosed between the sequences '\(' and '\)'is identical to the unadorned expression; the construction has side effects discussed under the *s* command.
9. 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.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line.* Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

-

1. The character '**.**' addresses the current line.
2. The character '$' addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '´*x*' addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A regular expression enclosed in slashes '/' addresses the 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.
6. 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 containing a string matching the regular expression. If necessary the search wraps around to the end of the buffer.
7. An address followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with '+' or '−' the addition or subtraction is taken with respect to the current line; e.g. '−5' is understood to mean '**.**−5'.
9. If an address ends with '+' or '−', then 1 is added (resp. subtracted). As a consequence of this rule and rule 8, the address '−' refers to the line before the current line. Moreover, trailing '+' and '−' characters have cumulative effect, so '−−' refers to the current line less 2.
10. To maintain compatibility with earlier version of the editor, the character '^' in addresses is entirely equivalent to '−'.

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

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

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

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

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

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

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

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

(**.** , **.**) 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 marks the addressed line with name *x,* which must be a lower-case letter. The address form '´*x*' then addresses this line.

(**.** , **.**) l

The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in octal, and long lines are folded. An *l* command may follow any other on the same line.

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

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

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

The print command prints the addressed lines. '**.**' is left at the last line printed. The *p* command may be placed on the same line after any 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 string matching the regular expression. The special meaning of '&' in this context may be suppressed by preceding it by '\'. As a more general feature, the characters '*n*', where *n* is a digit, are replaced by the text matched by the *n*-

-

th regular subexpression enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

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

(**.** , **.**) t *a*
This command acts just like the *m* command, except that a copy of the addressed lines is placed after address *a* (which may be 0). '**.**' is left on the last line of the copy.

(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.

(**.**+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* prints a '?' and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the temporary file. The limit on the number of lines depends on the amount of core: each line takes 1 word.

FILES

/tmp/#, temporary; '#' is the process number (in octal).

DIAGNOSTICS

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

SEE ALSO

A Tutorial Introduction to the ED Text Editor (B. W. Kernighan)

BUGS

The *s* command causes all marks to be lost on lines changed.

-

NAME

eqn − typeset mathematics

SYNOPSIS

eqn [file] ...

DESCRIPTION

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

eqn file ... | troff

If no files are specified, *eqn* reads from the standard input. A line beginning with ‘‘.EQ’’ marks the start of an equation; the end of an equation is marked by a line beginning with ‘‘.EN’’. Neither of these lines is altered or defined by *eqn,* so you can define them yourself to get centering, numbering, etc. All other lines are treated as comments, and passed through untouched.

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

Subscripts and superscripts are produced with the keywords **sub** and **sup.** Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$. Fractions are made with **over.** *a over b* is $\frac{a}{b}$ and *1 over sqrt {ax sup 2 +bx+c}* is $\frac{1}{\sqrt{ax^2+bx+c}}$. **sqrt** makes square roots.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n->\infty} \sum_0^n x_i$ is made with *lim from {n-> inf} sum from 0 to n x sub i.* Left and right brackets, braces, etc., of the right height are made with **left** and **right:** *left [x sup 2 + y sup 2 over alpha right] ˜=˜1* produces $\left[x^2 + \frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional.

Vertical piles of things are made with **pile, lpile, cpile,** and **rpile:** *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

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

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

SEE ALSO

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

BUGS

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

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. Two forms of "implicit" statements are recognized: **implicit integer /i−n/** or **implicit integer (i−n).**
3. The types doublecomplex, logical*1, integer*1, integer*2, integer*4 (same as integer), real*4 (real), and real*8 (double precision) are supported.
4. **&** as the first character of a line signals a continuation card.
5. **c** as the first character of a line signals a comment.
6. All keywords are recognized in lower case.
7. The notion of 'column 7' is not implemented.
8. G-format input is free form− leading blanks are ignored, the first blank after the start of the number terminates the field.
9. A comma in any numeric or logical input field terminates the field.
10. There is no carriage control on output.
11. A sequence of *n* characters in double quotes '"' is equivalent to *n* **h** followed by those characters.
12. In **data** statements, a hollerith string may initialize an array or a sequence of array elements.
13. The number of storage units requested by a binary **read** must be identical to the number contained in the record being read.
14. If the first character in an input file is "#", a preprocessor identical to the C preprocessor is called, which implements "#define" and "#include" preprocessor statements. (See the C reference manual for details.) The preprocessor does not recognize Hollerith strings written with *n* **h**.

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

FILES

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

rc (I), which announces a more pleasant Fortran dialect; the ANSI standard; ld (I) for loader flags. For some subroutines, try ierror, getarg, setfil (III).

DIAGNOSTICS

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

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

BUGS

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

NAME

file − determine file type

SYNOPSIS

file file ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ascii, *file* examines the first 512 bytes and tries to guess its language.

BUGS

-

NAME

find − find files

SYNOPSIS

find pathname expression

DESCRIPTION

Find recursively descends the directory hierarchy from *pathname* seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n,* −*n* means less than *n* and *n* means exactly *n.*

−**name** filename — True if the *filename* argument matches the current file name. Normal *Shell* argument syntax may be used if escaped (watch out for '[', '?' and '*').

−**perm** onum — True if the file permission flags exactly match the octal number *onum* (see chmod(I)). If *onum* is prefixed by a minus sign, more flag bits (017777, see stat(II)) become significant and the flags are compared: *(flags&onum)==onum.*

−**type** *c* — True if the type of the file is *c,* where *c* is **b, c, d** or **f** for block special file, character special file, directory or plain file.

−**links** *n* — True if the file has *n* links.

−**user** uname — True if the file belongs to the user *uname.*

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

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

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

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

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

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

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

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

! — prefix *not*

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

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

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

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

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

FILES

/etc/passwd

SEE ALSO

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

BUGS

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

-

NAME

goto – command transfer

SYNOPSIS

goto label

DESCRIPTION

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

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

SEE ALSO

sh (I)

BUGS

-

NAME

grep − search a file for a pattern

SYNOPSIS

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

DESCRIPTION

Grep searches the input files (standard input default) for lines matching the regular expression. Normally, each line found is copied to the standard output. If the −**v** flag is used, all lines but those matching are printed. If the −**c** flag is used, only a count of matching lines is printed. If the −**n** flag is used, each line is preceded its relative line number in the file. If the −**b** flag is used, each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

In all cases the file name is shown if there is more than one input file.

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. It is generally necessary to enclose the entire *expression* argument in quotes.

SEE ALSO

ed (I), sh (I)

BUGS

Lines are limited to 256 characters; longer lines are truncated.

NAME

if − conditional command

SYNOPSIS

if expr command [arg ...]

DESCRIPTION

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

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

−**r** file true if the file exists and is readable.

−**w** file true if the file exists and is writable.

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

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

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

These primaries may be combined with the following operators:

! unary negation operator

−**a** binary *and* operator

−**o** binary *or* operator

(expr) parentheses for grouping.

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

SEE ALSO

sh (I), find (I)

BUGS

-

NAME

kill − terminate a process

SYNOPSIS

kill [−signo] processid ...

DESCRIPTION

Kills the specified processes. The process number of each asynchronous process started with ‘&’ is reported by the Shell. Process numbers can also be found by using *ps* (I).

If process number 0 is used, then all processes belonging to the current user and associated with the same control typewriter are killed.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by ‘‘−’’ is given as first argument, that signal is sent instead of *kill* (see *signal (II))*.

SEE ALSO

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

BUGS

-

NAME

ld − link editor

SYNOPSIS

ld [−**sulxrdni**] 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 *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the −**r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out.** This file is made executable only if no errors occurred during the load.

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

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, 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. 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.

−**X** Save local symbols except for those whose names begin with 'L'. This option is used by *cc* to discard internally generated labels while retaining symbols local to routines.

−**r** generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

−**d** force definition of common storage even if the −**r** flag is present.

−**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.

−**i** When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and −**n** is that here the data starts at location 0.

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 asks 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 and message-of-the-day files. *Login* initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh* (I)) according to specifications found in a password file.

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

BUGS

-

NAME

ls − list contents of directory

SYNOPSIS

ls [**−ltasdruifg**] name ...

DESCRIPTION

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

−l list in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers.

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

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

−s give size in blocks for each entry

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

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

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

−i print i-number in first column of the report for each file listed

−f force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l, −t, −s,** and **−r,** and turns on **−a;** the order is the order in which entries appear in the directory.

−g Give group ID instead of owner ID in long listing.

The mode printed under the **−l** option contains 11 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

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

The last character of the mode is normally blank but is printed as ''t'' if the 1000 bit of the mode is on. See *chmod (I)* for the current meaning of this mode.

FILES

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

-

BUGS

-

NAME

mail − send mail to designated users

SYNOPSIS

mail [**−yn**] [person ...]

DESCRIPTION

Mail with no argument searches for a file called prints it if it is nonempty, then asks if it should be saved. If the answer is **y,** the mail is added to *mbox.* Finally is truncated to zero length. To leave the mailbox untouched, hit ‘delete.’ The question can be answered on the command line with the argument ‘−y’ or ‘−n’.

When *persons* are named, *mail* takes the standard input up to an end of file and adds it to each *person's* file. The message is preceded by the sender's name and a postmark.

A *person* is either a user name recognized by *login* (I), 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 in that directory is used.

When a user logs in he is informed of the presence of mail. No mail will be received from a sender to whom is inaccessible or unwritable.

FILES

/etc/passwd	to identify sender and locate persons
/etc/utmp	to identify sender
.mail	input mail
mbox	saved mail
/tmp/m#	temp file

SEE ALSO

write (I)

BUGS

-

NAME

man − run off section of UNIX manual

SYNOPSIS

man [section] [title ...]

DESCRIPTION

Man is a shell command file which locates and prints 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

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

neqn − typeset mathematics on terminal

SYNOPSIS

neqn [file] ...

DESCRIPTION

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

```
neqn file ... | nroff
```

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

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

SEE ALSO

eqn (I), gsi (VI)

BUGS

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

-

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp group

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to *login.* The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

A password is demanded if the group has a password and the user himself does not.

When most users log in, they are members of the group named ‘other.’

FILES

/etc/group, /etc/passwd

SEE ALSO

login (I), group (V)

BUGS

-

NAME

nice − run a command at low priority

SYNOPSIS

nice [**−***number*] command [arguments]

DESCRIPTION

Nice executes *command* with low scheduling priority. If a numerical argument is given, that priority (in the range 1-20) is used; if not, priority 4 is used.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. '−−10'.

SEE ALSO

nohup (I), nice (II)

BUGS

-

NAME

nm − print name list

SYNOPSIS

nm [**−cnrupg**] [name]

DESCRIPTION

Nm prints the symbol table from the output file of an assembler or loader run. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined) **A** (absolute) **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

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

Options are:

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

−g print only global (external) symbols

−n sort by value instead of by name

−p don't sort; print in symbol-table order

−r sort in reverse order

−u print only undefined symbols.

FILES

a.out

BUGS

NAME

nohup – run a command immune to hangups

SYNOPSIS

nohup command [arguments]

DESCRIPTION

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

SEE ALSO

nice (I), signal (II)

BUGS

-

NAME

nroff − format text

SYNOPSIS

nroff [**+***n*] [−*n*] [**−n***n*] [**−r***an*] [**−m***x*] [−**s**] [−**h**] [−**q**] files

DESCRIPTION

Nroff formats text according to control lines embedded in the text files. *Nroff* reads the standard input if no file arguments are given. An argument of just ‘‘−’’ refers to the standard input. The non-file option arguments are interpreted as follows:

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

−*n* Printing stops after page *n.*

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

−r*an* Set number register to the value *n.*

−m*name* Prepends a standard macro file; simulates ‘‘.so /usr/lib/tmac.*name*’’.

−**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.

FILES

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

SEE ALSO

NROFF User’s Manual (internal memorandum).
neqn (I), col (I)

BUGS

-

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 [–destination] [–**crm**] [name ...]

DESCRIPTION

Opr causes the named files to be printed off line at the specified destination. If no names appear the standard input is assumed.

At the mother system the following destinations are recognized. The default destination is **mh.**

lp Local line printer.

mh GCOS at Murray Hill Comp Center. GCOS identification must be registered in the UNIX password file (see passwd (V)).

sp Spider network printer.

xx The two-character code *xx* is taken to be a Murray Hill GCOS station id. Useful codes are ‘r1’ for quality print and ‘q1’ for quality print with special ribbon.

Opr uses spooling daemons that do the job when facilities become available. Flag –**r** causes the named files to be removed when spooled. Flag –**c** causes copies to be made so as to insulate the daemons from any intervening changes to the files.

Flag –**m** causes mail to be sent when UNIX is finished transmitting the file. For GCOS jobs the mail includes the snumb.

FILES

/etc/passwd	personal ident cards
/lib/dpr	dataphone spooler
/etc/dpd	dataphone daemon
/usr/dpd/*	spool area
/lib/lpr	line printer spooler
/etc/lpd	line printer daemon
/usr/lpd/*	spool area
/lib/npr	spider network spooler

SEE ALSO

fsend (I), dpd (VIII), lpd (VIII)

BUGS

Line printer spooler doesn’t handle flags.
Spider network spooler doesn’t spool.

NAME

passwd − change login password

SYNOPSIS

passwd name password

DESCRIPTION

The *password* becomes associated with the given login name. This can only be done by corresponding user or by the super-user. An explicit null argument ("") for the password argument removes any password.

FILES

/etc/passwd

SEE ALSO

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

BUGS

NAME

pfe − print floating exception

SYNOPSIS

pfe

DESCRIPTION

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

SEE ALSO

signal (II)

BUGS

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

-

NAME

pr − print file

SYNOPSIS

pr [−**h** *header*] [−*n*] [**+***n*] [−**w***n*] [−**l***n*] [−**t**] [−**s***c*] [−**m**] name . . .

DESCRIPTION

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

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

−*n* produce *n*-column output

+*n* begin printing with page *n*

−**h** treat the next argument as a header to be used instead of the file name

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

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

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

−**s***c* separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.

−**m** print all files simultaneously, each in one column

Interconsole messages via write(I) are forbidden during a *pr*.

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat (I), cp (I)

DIAGNOSTICS

none; files not found are ignored

BUGS

-

NAME

prof − display profile data

SYNOPSIS

prof [−**v**] [−**a**] [−**l**] [file]

DESCRIPTION

Prof interprets the file *mon.out* produced by the *monitor* subroutine. Under default modes, the symbol table in the named object file *(a.out* default) is read and correlated with the *mon.out* profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the −**a** option is used, all symbols are reported rather than just external symbols. If the −**l** option is used, the output is listed by symbol value rather than decreasing percentage. If the −**v** option is used, all printing is suppressed and a profile plot is produced on the 611 display.

In order for the number of calls to a routine to be tallied, the −**p** option of *cc* must have been given when the file containing the routine was compiled. This option also arranges for the *mon.out* file to be produced automatically.

FILES

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

SEE ALSO

monitor (III), profil (II), cc (I)

BUGS

Beware of quantization errors.

-

NAME

ps – process status

SYNOPSIS

ps [**aklx**] [namelist]

DESCRIPTION

Ps prints certain indicia about active processes. The **a** flag asks for information about all processes with typewriters (ordinarily only one's own processes are displayed); **x** asks even about processes with no typewriter; **l** asks for a long listing. Ordinarily only the typewriter number (if not one's own), the process number, and an approximation to the command line are given. If the **k** flag is specified, the file */usr/sys/core* is used in place of */dev/mem.* This is used for postmortem system debugging. If a second argument is given, it is taken to be the file containing the system's namelist.

The long listing is columnar and contains

The name of the process's control typewriter.

Flags associated with the process. 01: in core; 02: system process; 04: locked in code (e.g. for physical I/O); 10: being swapped; 20: being traced by another process.

The state of the process. 0: nonexistent; S: sleeping; W: waiting; R: running; Z: terminated; T: stopped.

The user ID of the process owner.

The process ID of the process; as in certain cults it is possible to kill a process if you know its true name.

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

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

The event for which the process is waiting or sleeping; if blank, the process is running.

The command and its arguments.

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

FILES

/unix	system namelist
/dev/mem	core memory
/usr/sys/core	alternate core file
/dev	searched to find swap device and typewriter names

SEE ALSO

kill (I)

BUGS

-

NAME

pwd − working directory name

SYNOPSIS

pwd

DESCRIPTION

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

SEE ALSO

chdir (I)

BUGS

NAME

rc − Ratfor compiler

SYNOPSIS

rc [−**c**] [−**r**] [−**f**] [−**v**] file ...

DESCRIPTION

Rc invokes the Ratfor preprocessor on a set of Ratfor source files. It accepts three types of arguments:

Arguments whose names end with '.r' are taken to be Ratfor source programs; they are preprocessed into Fortran and compiled. Each subroutine or function 'name' is placed on a separate file *name.f,* and its object code is left on *name.o.* The main routine is on *MAIN.f* and *MAIN.o;* block data subprograms go on *blockdata?.f* and *blockdata?.o.* The files resulting from a '.r' file are loaded into a single object file *file.o,* and the intermediate object and Fortran files are removed.

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

−c Suppresses the loading phase of the compilation, as does any error in anything.

−f Save Fortran intermediate files. This is primarily for debugging.

−r Ratfor only; don't try to compile the Fortran. This implies **−f** and **−c.**

−v Don't list intermediate file names while compiling.

Arguments whose names end with '.f' are taken to be Fortran source programs; they are compiled in the normal manner. (Only one Fortran routine is allowed in a '.f' file.) Other arguments are taken to be either loader flag arguments, or Fortran-compatible object programs, typically produced by an earlier *rc* run, or perhaps libraries of Fortran-compatible routines. These programs, together with the results of any compilations specified, are loaded to produce an executable program with name **a.out.**

FILES

ratjunk	temporary
/usr/bin/ratfor	preprocessor
/usr/fort/fc1	Fortran compiler

SEE ALSO

"RATFOR − A Rational Fortran".
fc(I) for Fortran error messages.

DIAGNOSTICS

Yes, both from *rc* itself and from Fortran.

BUGS

Limit of about 50 arguments, 10 block data files.

#define and #include lines in ".f" files are not processed.

-

NAME

rev − reverse lines of a file

SYNOPSIS

rev

DESCRIPTION

Rev copies the standard input to the standard output, reversing the order of characters in every line.

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 a file has no write permission, *rm* prints the file name and its mode, then reads a line from the standard input. If the line begins with **y**, the file is removed, otherwise it is not. The question is not asked if option **−f** was given or if the standard input is not a typewriter.

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 to be removed appears.

BUGS

Needs a **−r** flag.
Actually, write permission in the directory's parent is *not* required.
Mildly unpleasant consequences can follow removal of your own or someone else's current directory.

-

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.

Input consists of intermixed *text lines,* which contain information to be formatted, and *request lines,* which contain instructions about how to format it. Request lines begin with a distinguished *control character,* normally a period.

Output lines may be *filled* as nearly as possible with words without regard to input lineation. Line *breaks* may be caused at specified places by certain commands, or by the appearance of an empty input line or an input line beginning with a space.

The capabilities of *roff* are specified in the attached Request Summary. Numerical values are denoted there by n or +n, titles by t, and single characters by c. Numbers denoted +n may be signed + or −, in which case they signify relative changes to a quantity, otherwise they signify an absolute resetting. Missing n fields are ordinarily taken to be 1, missing t fields to be empty, and c fields to shut off the appropriate special interpretation.

Running titles usually appear at top and bottom of every page. They are set by requests like

.he ′part1′part2′part3′

Part1 is left justified, part2 is centered, and part3 is right justified on the page. Any % sign in a title is replaced by the current page number. Any nonblank may serve as a quote.

ASCII tab characters are replaced in the input by a *replacement character,* normally a space, according to the column settings given by a .ta command. (See .tr for how to convert this character on output.)

Automatic hyphenation of filled output is done under control of .hy. When a word contains a designated *hyphenation character,* that character disappears from the output and hyphens can be introduced into the word at the marked places only.

FILES

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

SEE ALSO

nroff (I), troff (I)

BUGS

Roff is the simplest of the runoff programs, but is utterly frozen.

-

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 parameterless macro to be invoked by request ‘.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 becomes ‘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	Add 5 to page offset; number lines in margin from 1 on each page.
.n2 n	no	no	Add 5 to page offset; number lines from n; stop 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, except at top of page.
.ss	yes	yes	Single space output lines, equivalent to ‘.ls 1’.
.ta n n..		-	Pseudotab settings. Initial tab settings are columns 9 17 25 ...
.tc c	no	space	Tab replacement character becomes ‘c’.
.ti +n	yes	-	Temporarily indent next output line n spaces.
.tr cdef..	no	-	Translate c into d, e into f, etc.
.ul n	no	-	Underline the letters and numbers in the next n input lines.

NAME

sh – shell (command interpreter)

SYNOPSIS

sh [**–t**] [**–c**] [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 a non-directory file has executable mode, but not the form of an executable program (does not begin with the proper magic number) then it is assumed to be an ASCII file of commands and a new Shell is created to execute it. See ''Argument passing'' below.

If the file cannot be found, a diagnostic is printed.

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

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

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

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

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

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

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

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

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

For example, either of the command lines

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

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

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

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

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

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

The character ‘?’ matches any single character in a file name.

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

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

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

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

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

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 is read and interpreted. If not given, this subinstance of the Shell continues 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.*

The argument '−t,' used alone, causes *sh* to read the standard input for a single line, execute it as a command, and then exit. This facility replaces the older 'mini-shell.' It is useful for interactive programs which allow users to execute system commands.

The argument '−c' (used with one following argument) causes the next argument to be taken as a command line and executed. No new-line need be present, but new-line characters are treated appropriately. This facility is useful as an alternative to '−t' where the caller has already read some of the characters of the command to be executed.

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.

Processes that are created with '&' 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', CACM, July, 1974, which gives the theory of operation of the Shell.
chdir (I), login (I), wait (I), shift (I)

BUGS

There is no way to redirect the diagnostic output.

NAME

shift − adjust Shell arguments

SYNOPSIS

shift

DESCRIPTION

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

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

prints each of its arguments in 3-column format.

Shift is executed within the Shell.

SEE ALSO

sh (I)

BUGS

-

NAME

size − size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

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

BUGS

-

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

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

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

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

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

SEE ALSO

sleep (II)

BUGS

Time must be less than 65536 seconds.

-

NAME

sort, usort − sort or merge files

SYNOPSIS

sort [−**abdnrt***x*] [**+***pos* [*−pos*]] . . . [−**mo**] [name] . . .
usort [**−umo**] [name] . . .

DESCRIPTION

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

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

a Do not map lower case letters.

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

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

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

r Reverse the sense of comparisons.

t*x* Tab character between fields is *x*.

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

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

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

These flag arguments are also understood:

−m Merge only, the input files are already sorted.

−o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs, except under the merge flag **−m**.

Usort is a somewhat specialized version of *sort* which accepts no collating sequence options: order is always plain ASCII. It also strips out the second and following copies of duplicated lines. A *u* flag prevents this stripping. *Usort* also understands the *m* and *o* options in the same way as *sort.*

FILES

/usr/tmp/stm???

BUGS

NAME

spell − find spelling errors

SYNOPSIS

spell [**−v**] file ...

DESCRIPTION

Spell collects the words from the named documents, and looks them up in a dictionary. The words not found are printed on the standard output. Words which are reasonable transformations of dictionary entries (e.g. a dictionary entry plus *s*) are not printed. If no files are given, the input is from the standard input.

If the **−v** flag is given, all words which are not literally in the dictionary are printed; those which can be transformed to lie in the dictionary are so marked, and the others are marked with asterisks.

The process takes several minutes.

FILES

/usr/lib/w2006, /usr/dict/words, /usr/lib/spell[123]

SEE ALSO

typo (I)

BUGS

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

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

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

The dictionary cannot be distributed because of copyright limitations.

NAME

split − split a file into pieces

SYNOPSIS

split −*n* [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if − is given in its stead, then the standard input file is used.

BUGS

NAME

strip – remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

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

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

FILES

/tmp/stm? temporary file

SEE ALSO

ld (I), as (I)

BUGS

-

NAME

stty − set typewriter options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output typewriter. With no argument, it reports the current settings of the options. 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
cooked	same as '−raw'
−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 when printing
tabs	preserve tabs
ek	reset erase and kill characters back to normal # and @.
erase *c*	set erase character to *c*.
kill *c*	set kill character to *c*.
cr0 cr1 cr2 cr3	select style of delay for carriage return (see below)
nl0 nl1 nl2 nl3	select style of delay for linefeed (see below)
tab0 tab1 tab2 tab3	select style of delay for tab (see below)
ff0 ff1	select style of delay for form feed (see below)
tty33	set all modes suitable for Teletype model 33
tty37	set all modes suitable for Teletype model 37
vt05	set all modes suitable for DEC VT05 terminal
tn300	set all modes suitable for GE Terminet 300
ti700	set all modes suitable for Texas Instruments 700 terminal
tek	set all modes suitable for Tektronix 4014 terminal
hup	hang up dataphone on last close.
−hup	do not hang up dataphone on last close.
0	hang up phone line immediately
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb	Set typewriter baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

The various delay algorithms are tuned to various kinds of terminals. In general the specifications ending in '0' mean no delay for the corresponding character.

SEE ALSO

stty (II)

BUGS

-

NAME

tee − pipe fitting

SYNOPSIS

tee [name ...]

DESCRIPTION

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

BUGS

NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

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

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

The times are printed on the diagnostic output stream.

BUGS

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

-

NAME

tp − manipulate DECtape and magtape

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or magtape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

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

r The named files are written on 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 like **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 from the tape. At least one name 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 and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t lists the names of the specified files. 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 is 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

A single file with several links to it is treated like several files.

-

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

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

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

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

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

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

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

SEE ALSO

sh (I), ed (I), ascii (V)

BUGS

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

-

NAME

troff − format text

SYNOPSIS

troff [**+***n*] [−*n*] [**−s***n*] [**−n***n*] [**−r***an*] [**−m***name*] [−**t**] [−**f**] [−**w**] [−**a**] [**−p***n*] files

DESCRIPTION

Troff formats text for a Graphic Systems phototypesetter according to control lines embedded in the text files. It reads the standard input if no file arguments are given. An argument of just "**−**" refers to the standard input. 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.*

−s*n* Print output in groups of *n* pages, stopping the typesetter after each group.

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

−r*an* Set number register *a* to the value *n.*

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

−**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 phototypesetter is available, if currently busy.

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

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

FILES

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

SEE ALSO

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

BUGS

-

NAME

tty – get typewriter name

SYNOPSIS

tty

DESCRIPTION

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

DIAGNOSTICS

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

BUGS

-

NAME

typo − find possible typos

SYNOPSIS

typo [−**1**] [−**n**] file ...

DESCRIPTION

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

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

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

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

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

FILES

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

BUGS

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

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

-

NAME

uniq − report repeated lines in a file

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

SEE ALSO

sort (I), comm (I)

BUGS

-

NAME

wait − await completion of process

SYNOPSIS

wait

DESCRIPTION

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

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

SEE ALSO

sh (I)

BUGS

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

-

NAME

wc – word count

SYNOPSIS

wc [name ...]

DESCRIPTION

Wc counts lines and words in the named files, or in the standard input if no name appears. A word is a maximal string of printing characters delimited by spaces, tabs or newlines. All other characters are simply ignored.

BUGS

-

NAME

who – who is on the system

SYNOPSIS

who [who-file] [**am I**]

DESCRIPTION

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

Without an argument, *who* examines the /etc/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file will be /usr/adm/wtmp, which contains a record of all the logins since it was created. Then *who* lists 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.

With two arguments, *who* behaves as if it had no arguments except for restricting the printout to the line for the current typewriter. Thus 'who am I' (and also 'who are you') tells you who you are logged in as.

FILES

/etc/utmp

SEE ALSO

login (I), init (VIII)

BUGS

-

NAME

write − write to another user

SYNOPSIS

write user [ttyno]

DESCRIPTION

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

message from yourname...

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

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

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

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

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

FILES

/etc/utmp to find user
/bin/sh to execute '!'

SEE ALSO

mesg (I), who (I), mail (I)

BUGS

-

NAME

yacc − yet another compiler-compiler

SYNOPSIS

yacc [−**vor**] [grammar]

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output is *y.tab.c,* which must be compiled by the C compiler and loaded with any other routines required (perhaps a lexical analyzer) and the Yacc library:

```
cc y.tab.c other.o −ly
```

If the −**v** flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

The −**o** flag calls an optimizer for the tables; the optimized tables, with parser included, appear on file *y.tab.c*

The −**r** flag causes Yacc to accept grammars with Ratfor actions, and produce Ratfor output on *y.tab.r;* −**r** implies the −**o** flag. Typical usage is then

```
rc y.tab.r other.o
```

SEE ALSO

"LR Parsing", by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974. "The YACC Compiler-compiler", internal memorandum.

AUTHOR

S. C. Johnson

FILES

y.output	
y.tab.c	
y.tab.r	when ratfor output is obtained
yacc.tmp	when optimizer is called
/lib/liby.a	runtime library for compiler
/usr/yacc/fpar.r	ratfor parser
/usr/yacc/opar.c	parser for optimized tables
/usr/yacc/yopti	optimizer postpass

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file.

BUGS

Because file names are fixed, at most one Yacc process can be active in a given directory at a time.