Tektronix®
COMMITTED TO EXCELLENCE

This manual supports TNIX
versions 1.0, 1.2, 1.3, and 1.4.

PLEASE CHECK FOR CHANGE INFORMATION AT THE REAR OF THIS MANUAL

8560 Series
MULTI-USER SOFTWARE DEVELOPMENT UNIT
SYSTEM REFERENCE MANUAL
TNIX VERSION 1

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Serial Number ____________

070-3941-00
Product Group 61

First Printing DEC 1981
Revised MAR 1983

MANUAL REVISION STATUS

8560 MUSDU SYSTEM REFERENCE MANUAL (070-3941-00)

This System Users Manual supports Versions 1.0, 1.2, 1.3, and 1.4 of the 8560 TNIX operating system.

REV DATE	DESCRIPTION
DEC 1981	Original Issue
NOV 1982	Replaced: pages 1-5-1-10, 1-61-1-64, 1-85, 1-86, 1-119, 1-120, 2-30, 4-3, 4-4, 5-10, 6-19-6-22, 8-6-8-9, 8-12, 8-15, 8-18, 8-19, 8-22-8-27.
NOV 1982	Added: pages 1-28a, 1-28b, 1-72a, 1-72b, 4-4a, 4-4b, 5-8a, 5-8b, 5-10a-5-10f, 5-22a, 5-22b, 6-56a, 6-56b, 8-4a, 8-4b, 8-12a, 8-12b, 8-14a, 8-14b, 8-18a, 8-18b, 8-20a, 8-20b.
MAR 1983	Replaced: pages 8-4a, 8-4b, 8-6, 8-7, 8-8, 8-14b, 8-15, 8-18b, 8-19.
MAR 1983	Added: pages 8-4c, 8-7a, 8-7b, 8-7c, 8-7d, 8-19a, 8-19b.

Section 0
Introduction

OVERVIEW

This manual describes the publicly available features of the **TNIX** operating system. It provides a terse, easily accessed summary of commands, command options, system calls, and other information that you may require while working within the **TNIX** system. Learning guides, in–depth discussions of some of the command *processors,* and other "how-to" types of information can be found in the *8560 MUSDU System Users Manual.* Some of the commands and programming packages described in this manual are not included in the base **TNIX** package, but are available as optional software packages.

This manual is divided into eight sections:

Section 1	Commands
Section 2	System Calls
Section 3	Subroutines
Section 4	Special Files
Section 5	File Formats and Conventions
Section 6	Category C Software
Section 7	Macro packages and language conventions
Section 8	System Maintenance

References to specific entries or pages in this manual are of the form **command–name***(section-number)*. For example, "consult **ed***(1)* " means that you should look up the **ed** entry in Section 1 for further information on the topic you are reading about. **Commands** are programs invoked directly by the user, as opposed to subroutines, which are called by the user's programs. Most commands reside in directory */bin* (for *bin*ary programs); some, mostly the Category C commands, reside in */usr/bin* to save space in */bin.* Both directories are searched automatically by the command interpreter.

System calls are entries into the **TNIX** supervisor. Each system call has one or more C language interfaces, as described in Section 2. The underlying assembly language interface is also given.

Section 3 describes the standard system **subroutines.** The primary libraries in which the subroutines are kept are described in *intro*(3). The subroutines are described in terms of the C language, but most will work with Fortran.

Section 4, Special Files, discusses characteristics of system files or I/O devices. Names in this section refer to hardware device names rather than the names of the special files themselves.

Section 5, File Formats and Conventions, documents the structure of particular types of files; for example, the output format of both the loader and assembler is given. Files used by only one command, such as the assembler's intermediate files, are not described.

Section 6 describes the Category C software available with the **TNIX** operating system.

Section 7 contains charts, tables, and programming tools for writing in various specialized languages —an ASCII table, macro packages for typesetting and document preparation, etc.

Section 8, Maintenance, outlines some of the procedures intended for use by the system manager. Consult the Systems Maintenance section of the *8560 MUSDU System Users Manual* for complete information on system maintenance procedures.

Section 1
Commands

INTRODUCTION

A command consists of the command name followed by zero or more arguments, each separated by a space. Arguments are either options or filenames; an argument beginning with a minus sign '–' is an option, even if it is positionally where a file name should be. It is unwise to begin filenames with'–'.

Options are of two major types — flags and parameters. Flags have an on-off value; a parameter indicates that the following argument is a value to be used as indicated by the parameter syntax.

Filename arguments are processed in order of occurance.

Any command that accepts multiple input files accepts a '—' in place of a filename; this indicates that the standard input is to be used.

Several single character flags may be grouped after a single minus sign as one argument (e.g. −xyz instead of −x −y −z).

The **name** subsection lists the exact name of the command and subroutine covered and gives a short description of its purpose.

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

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

Square brackets [ ] around an argument indicate that the argument is optional. When an argument is given as 'name', it always refers to a file name.

Ellipses '...' are used to show that the previous argument-prototype may be repeated.

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

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

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

The **diagnostics** subsection discusses the diagnostic indications which may be produced. Self-explanatory messages are not listed.

The **bugs** subsection lists known bugs and occasional deficiencies. A suugested fix is sometimes offered.

In section 2 an **assembler** subsection carries the assembly language system interface.

INTRO(1)

NAME

intro – introduction to commands

DESCRIPTION

This section describes publicly accessible commands in alphabetic order.

DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and, in the case of 'normal' termination, one supplied by the program — see *wait* and *exit*(2). The former byte is 0 for normal termination, the latter 0 for successful execution. A nonzero return value indicates problems such as erroneous parameters, bad or inaccessible data, or some other inability to cope with the task at hand. The returned status is called 'exit code', 'exit status' or 'return code', and is described only when special conventions are involved.

8540(1)

NAME

8540, 8550 – program to run commands on an 8540 or 8550

SYNTAX

<8540/8550 command> [flag options] [argument] ...

8540 *string*

8550 *string*

DESCRIPTION

8540 controls 8540s and/or 8550s attached to the 8560. Each 8540/8550 command is defined on the 8560 by a link, whose name is the 8540/8550 command name, to **8540**. When an 8540/8550 command is invoked, **8540** will format the flag-options and arguments into a blank separated carriage return terminated command line which it then sends to the proper 8540. **8540** will then wait for the 8540/8550 to finish the command. **8540** will also service requests for file system operations from the 8540/8550.

Three "channels" are always available to the 8540/8550 command (the **TNIX** stdin, stdout, and stderr). Other "channels" may be opened and used by the 8540 issuing an "open" request and then subsequent "read" or "write" requests.

When **8540** is terminated, whether by an interrupt or quit signal, or the 8540 program finishing, **8540** will save information about any files that have been opened by an HSI request and have not been closed.

When the next 8540/8550 command requests to do some I/O other than with stdin, stdout, or stderr; **8540** will look in the file that contains the saved information and open and position the file so that it will seem as if it was never closed.

When invoked as **8540** or **8550**, **8540** will send *string* to the 8540/8550 without change and without being prefixed by the name of the command which invoked **8540**. All other operations will be performed as described above. This command allows 8540/8550 string assignments and 8550 command file invocations to be controlled from the 8560.

[This page intentionally left blank.]

ACE(1)

NAME

ace—advanced crt-oriented editor

SYNTAX

ace inputfile [*outputfile*] [**–c** *configfile*]

DESCRIPTION

Ace is a screen-oriented text editor. Editing commands are read from TNIX standard input; terminal output is sent to TNIX standard output.

OPTIONS

inputfile	The file to be created or edited with **ace**.
outputfile	The file that contains the editing changes; *inputfile* is not modified.
–c *configfile*	The terminal configuration file that describes the operating characteristics of the terminal used with **ace**. You can create, modify, or display *configfile* with **aceconfig***(1)*.

THE DEFAULT TERMINAL CONFIGURATION FILE

Ace requires a terminal configuration file in order to operate properly. If you do not use a TEKTRONIX CT8500 terminal with **ace**, or you use a modified terminal configuration file, see **aceconfig***(1)* or the ACE Configurator section of the 8500 Modular MDL Series ACE Version 2 Reference Manual.

If you do not specify a *configfile* in the **ace** command line, **ace** determines which terminal configuration file to use according to the following rules, in the order in which they are listed:

1. If the shell variable ACECONFIG has been assigned a value, **ace** uses that value for *configfile.*
2. If the shell variable ACECONFIG has not been assigned a value, but the shell variable TERM has been assigned a value, **ace** uses "${TERM}.cfg" for *configfile.* ("${TERM}.cfg" is the value of TERM with the suffix ".cfg" appended to it). If "${TERM}.cfg" is not in the current directory, **ace** uses "/usr/lib/ace/${TERM}.cfg" for *configfile.*
3. If the ACECONFIG and TERM variables are not set, or are set to the null string, then **ace** uses ct8500.cfg (in the current directory) for *configfile.* If ct8500.cfg is not in the current directory, **ace** uses "/usr/lib/ace/ct8500.cfg" for *configfile.*

INVOCATION EXAMPLE

Here is an example of an **ace** invocation line:

```
$ ace w1.in w1.out -c new.cfg <CR>
```

This example:

- executes **ace** from the TNIX shell;
- configures **ace** to the terminal you are using, with the "new.cfg" terminal configuration file; and
- writes the editing changes made to "w1.in" to the "w1.out" file.

DEFAULT TERMINAL CONFIGURATION EXAMPLE

The following example shows how to set up your ".profile" file (the shell command file that is executed each time you log in to TNIX) so that you can set the TERM variable to the name of the terminal that you will use with **ace**.

Add the following lines to your ".profile" file in your home (/usr/yourname) directory:

```
TERM=ct8500
echo -n "terminal type (default = ${TERM})?"
read term
test -n "$term" && TERM="$term"
export TERM
```

Then, when your ".profile" file is executed, the following prompt will be displayed:

```
terminal type (default = ct8500)?
```

If you type a <CR> in response to the prompt, the TERM variable is set to "ct8500"; otherwise, TERM is set to the value that you enter in response to the prompt. If you want to use the terminal configuration file called "/usr/lib/ace/aaa.cfg" with **ace**, type

```
terminal type (default = ct8500)?  aaa <CR>
```

in response to the above prompt. Then, execute your ".profile" file by typing:

```
$ . .profile <CR>
```

(This can be useful if you entered the wrong terminal type). When you invoke **ace** with the command line

```
$ ace yourfile <CR>
```

ace will use the "/usr/lib/ace/aaa.cfg" terminal configuration file.

NOTE

(If a file called "aaa.cfg" exists in your current directory, **ace** will use that file instead of "/usr/lib/ace/aaa.cfg".)

FILES

/usr/tmp/acetm*N.pid*	The temporary file used by **ace**. *N* is an integer from 1 to 3, and *pid* is the current process id number.
/usr/lib/ace	The directory that is searched if a terminal configuration file is not found in the current directory.
/usr/lib/ace/ct8500.cfg	The default terminal configuration file supplied with **ace** for use with the TEKTRONIX CT8500 terminal.

SEE ALSO

8560 MUSDU ACE Version 2 Users Booklet
8500 Modular MDL Series ACE Version 2 Reference Manual
aceconfig*(1)*

ACECONFIG(1)

NAME

aceconfig—create, modify, or view a terminal configuration file

SYNTAX

aceconfig [**–c –m –v**] [*configfile*]

DESCRIPTION

Aceconfig is an interactive program that creates, modifies, and displays the **ace** terminal configuration file. The terminal configuration file tells **ace** how to recognize the character sequences that correspond to **ace** commands. The terminal configuration file also tells **ace** which character sequences to send to the terminal to display the results of these commands.

OPTIONS

–c	Create a terminal configuration file.
–m	Modify a terminal configuration file.
–v	View a terminal configuration file. If you do not specify one of these flags when you invoke **aceconfig**, **aceconfig** views the terminal configuration file if it already exists, otherwise **aceconfig** creates the terminal configuration file.
configfile	The name of the file to be created, viewed, or modified. If you do not specify a *configfile* in the **aceconfig** command line, **aceconfig** determines which terminal configuration file to use according to the following rules, in the order in which they are listed:

1. If the shell variable ACECONFIG has been assigned a value, **aceconfig** uses that value for *configfile.*
2. If the shell variable ACECONFIG has not been assigned a value, but the shell variable TERM has been assigned a value, **aceconfig** uses "${TERM}.cfg" for *configfile.* ("${TERM}.cfg" is the value of TERM with the suffix ".cfg" appended to it). If "${TERM}.cfg" is not in the current directory, **aceconfig** uses "/usr/lib/ace/${TERM}.cfg" for *configfile.*
3. If the ACECONFIG and TERM variables are not set, or are set to the null string, then **aceconfig** uses ct8500.cfg (in the current directory) for *configfile.* If ct8500.cfg is not in the current directory, **aceconfig** uses "/usr/lib/ace/ct8500.cfg" for *configfile.*

FILES

/usr/lib/ace	The directory that is searched if a terminal configuration file is not found in the current directory.
/usr/lib/ace/ct8500.cfg	The configuration file supplied with **ace** for use with the TEKTRONIX CT8500 terminal.
/etc/termcap	The terminal definition file. Configuration parameters that are defined in "/etc/termcap" are used by **aceconfig** when modifying or creating a terminal configuration file.

SEE ALSO

8560 MUSDU ACE Version 2 Users Booklet
8500 Modular MDL Series ACE Version 2 Reference Manual
ace*(1)*, **termcap***(5)*

[This page intentionally left blank.]

ASM(1)

NAME

asm—invoke processor-dependent assembler

SYNTAX

asm objectfile listfile sourcefile1 [*sourcefile2 sourcefile2* . . .]

PARAMETERS

objectfile	The file that you specify to receive the object code generated by the assembler. If you want **asm** to send *objectfile* to the standard output, you must specify a null string in place of *objectfile* by typing ''.
listfile	The file that you specify to receive the listing generated by the assembler. If you want **asm** to send *listfile* to the standard output, you must specify a null string in place of *listfile* by typing ''.
sourcefile	The file that contains the source code to be assembled by **asm. You must specify at least one** *sourcefile*. You can specify additional sourcefiles in the **asm** command line.

DESCRIPTION

Asm is the Tektronix Assembler. The source code, residing in one or more files, is translated into object code (machine language), which is written to the specified file or device. An assembler listing is also generated and is written to the specified file or device.

To use a specific microprocessor with the **asm** command, you must set the "uP" shell variable equal to the name of the microprocessor. For example, to use the 8085 microprocessor with the **asm** command, set the "uP" variable to "8085" by typing:

```
$ uP=8085; export uP <CR>
```

The proper assembler must be installed on your system in order for the **asm** command to work.

EXAMPLES

The following examples assume that 8085.o is *objectfile*, 8085.l is *listfile*, and 8085.s is *sourcefile*. If you do not want to specify *objectfile*, type:

```
$ asm '' 8085.l 8085.s <CR>
```

If you do not want to specify *listfile*, type:

```
$ asm 8085.o '' 8085.s <CR>
```

If you do not want to specify both *objectfile* and *listfile*, type:

```
$ asm '' '' 8085.s <CR>
```

INVOKING PROCESSOR-DEPENDENT COMMANDS

Asm is also an interface to various processor-dependent commands, such as **pas**, **ics**, **icsp**, **pdb**, etc., that run on the 8560. **Asm** prefixes the processor-dependent command that you enter with /bin/$uP, then executes the newly-created command-name with the **exec** command (see **exec***(2)*). The arguments that you entered with the original processor-dependent command are passed unchanged to the newly-created command-name. You can link files to the processor-dependent command—the file with the corresponding name, /bin/${uP}asm, will be executed.

SEE ALSO

exec*(2)*

[This page intentionally left blank.]

BASENAME(1)

NAME

basename – strip filename affixes

SYNTAX

basename string [suffix]

DESCRIPTION

Basename deletes any prefix ending in '/' and the *suffix,* if present in *string,* from *string,* and prints the result on the standard output. It is normally used inside substitution marks \` \` in shell procedures.

This shell procedure invoked with the argument */usr/src/cmd/cat.c* compiles the named file and moves the output to *cat* in the current directory:

```
15n    cc $1
       mv a.out `basename $1 .c`
```

SEE ALSO

sh(1)

[This page intentionally left blank.]

CAT(1)

NAME

cat − catenate and print

SYNTAX

cat [−u] [*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 *file* is given, or if the argument '−' is encountered, **cat** reads from the standard input.

OPTIONS

−u Output is buffered in 512-byte blocks unless the standard output is a terminal or the −u option is present.

SEE ALSO

pr(1), cp(1)

NOTES

Beware of 'cat a b >a' and 'cat a b >b', which destroy input files before reading them.

[This page intentionally left blank.]

CD(1)

NAME

cd – change working directory

SYNTAX

cd [*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, **cd** would be ineffective if it were written as a normal command. It is therefore recognized and executed by the Shell.

SEE ALSO

sh(1), pwd(1), chdir(2)

[This page intentionally left blank.]

CHMOD(1)

NAME

chmod – change mode

SYNTAX

chmod *mode file...*

DESCRIPTION

The mode of each named file is changed according to *mode,* which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

04000	set user ID on execution
02000	set group ID on execution
01000	save image after execution—used only for heavily used commands
00400	read by owner
00200	write by owner
00100	execute (search in directory) by owner
00070	read, write, execute (search) by group
00007	read, write, execute (search) by others

A symbolic *mode* has the form:

[who] op permission [, [who] op permission] ...

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for all **(ugo)**. If *who* is omitted, the default is *a* but the setting of the file creation mask (see umask(2)) is taken into account. The file creation mask shows which bits should be off.

Op can be + to add *permission* to the file's mode, – to take away *permission* and **=** to assign *permission* absolutely (all other bits in that group will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group id) and **t** (save image). Letters **u, g** or **o** indicate that *permission* is to be copied from the current mode. Omitting *permission* is only useful with **=** to take away all permissions.

The first example denies write permission to others, the second makes a file executable by everyone:

chmod o–w file

chmod +x file

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The permission **s** is only useful with **u** or **g**.

Only the owner of a file (or the super-user) may change its mode. Only the superuser may set the save image (1000) bit.

SEE ALSO

ls(1), chmod(2), chown (1), stat(2), umask(2)

[This page intentionally left blank.]

CHOWN(1)

NAME

chown, chgrp – change owner or group

SYNTAX

chown *owner file...*

chgrp *group file...*

DESCRIPTION

Chown changes the owner of *file(s)* to *owner.* The owner may be either a decimal user-id or a login name found in the password file.

Chgrp changes the group-ID of *file(s)* to *group.* The group may be either a decimal group-id or a group name found in the group-ID file.

Only the super-user can change owner or group.

FILES

/etc/passwd
/etc/group

SEE ALSO

chown(2), passwd(5), group(5)

[This page intentionally left blank.]

CMP(1)

NAME

cmp – compare two files

SYNTAX

cmp [**–ls**] *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.

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 DIAGNOSTICS below. **–l** and **–s** are mutually exclusive.

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

[This page intentionally left blank.]

COMM(1)

NAME

comm – select or reject lines common to two sorted files

SYNTAX

comm [– [**123**]] *file1 file2*

DESCRIPTION

Comm reads *file1* and *file2,* which should be ordered in ASCII collating sequence, 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(1), diff(1), uniq(1)

[This page intentionally left blank.]

CP(1)

NAME

cp – copy

SYNTAX

cp *file1 file2*

cp *file... directory*

cp *directory1 directory2*

DESCRIPTION

File1 is copied onto *file2*. The mode and owner of *file2* are preserved if it already existed; the mode of the source file is used otherwise.

In the second form, one or more *files* are copied into the *directory* with their original file-names.

The third form will copy the subtree with the root at *directory1* to *directory2*. *Directory2* must not be in the *directory1* subtree.

SEE ALSO

cat(1), pr(1), mv(1)

NOTES

Cp refuses to copy a file onto itself.

[This page intentionally left blank.]

DATE(1)

NAME

date – print and set the date

SYNTAX

date *[dd-mmm-yy] [hh:mm[:ss]] [-t SSS] [-d DDD] [[-w [hh:mm]] [-e [hh:mm]]]*

date *[[yy]mmddhhmm [.ss]] [-t SSS] [-d DDD] [[-w [hh:mm]] [-e [hh:mm]]]*

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 *dd* is the day number in the month; *mmm* is the lowercase first three letters of the month; *mm* is the month number; *yy* is the last two digits of the year; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *ss* is optional and is the seconds. In the first form of date specification the time only may be set and the day-month-year string will not be modified. If the day-month-year string only is specified the time will not be modified. Also if seconds are omitted system seconds will not be modified. For example:

```
date 14-nov-80 12:45
```

or

```
date 8011141245
```

set the date to Nov 14, 1980 12:45 PM.

```
date 3:45:20
```

sets the datetime to Nov 14, 1980 3:45:20 AM if it were done on the same day as the above date command.

```
date -t PST -d PDT -w 8:30
```

leaves the date the same but sets the timezone strings and the hours offset from Greenwich Mean Time to be 8 1/2 hours. **Date** takes care of the conversion to and from local standard and daylight time in the United States.

OPTIONS

–t SSS sets the system standard timezone string. If this string is not set or is null the system will print out GMT +/- HH for the timezone. HH representing hours west of GMT.

–d DDD
sets the system daylight timezone string. If this string is null or if it was never set the system assumes that there is no daylight savings time.

−w hh:mm
: sets the hours and minutes west of Greenwich Mean Time that the timezone strings represent if you are west of the Prime Meridian.

−e hh:mm
: sets the hours and minutes east of Greenwich Mean Time that the timezone strings represent if you are east of the Prime Meridian.

The above options may be invoked independently of setting the date, time, or the other options.

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.

DF(1)

NAME

df—disk free

SYNTAX

df [*filesystem*] ...

DESCRIPTION

Df prints out the number of free blocks available for file allocation on the *filesystem(s)*. If no file system is specified, the free space on each filesystem listed in the file **/etc/checklist** is printed. (If **/etc/checklist** cannot be read by **df**, the "/dev/rhd0" filesystem will be used.)

FILES

/etc/checklist

SEE ALSO

checklist*(5)* **hd***(5)*

[This page intentionally left blank.]

DIFF(1)

NAME

diff – differential file comparator

SYNTAX

diff [**–efbh**] *file1 file2*

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1 (file2)* is '–', the standard input is used. If *file1 (file2)* is a directory, then a file in that directory whose file-name is the same as the file-name of *file2 (file1)* is used. 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 *file1* into *file2* The **-e** option produces actual editor commands for use in converting files. The numbers after the letters pertain to *file2* In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert *file2* into *file1* 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 '>'.

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

OPTIONS

–b	Causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.
–e	Produces a script of **a** , **c** and **d** commands for the editor **ed** , which will recreate *file2* from *file1*
–f	Produces a similar script to that produced by **–e** , not useful with **ed** , in the opposite order.
–h	Does a fast, simplistic job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **–e**, **–f**, and **–h** are mutually exclusive.

FILES

/tmp/d?????
/usr/lib/diffh for –h

SEE ALSO

cmp(1), comm(1), ed(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

NOTES

Editing scripts produced under the **–e** or **–f** option will not correctly create lines consisting of a single '.'

[This page intentionally left blank.]

DSC50(1)

NAME

dsc50, mk50, wr50, del50, rd50, dir50, ln50 – manipulate 8550 format diskettes

SYNTAX

dsc50 [−**mwdxtls**] [−**epbrvq**] [−**c** *path*] [−**o** *owner*]
[−**n** *name*] [−**a** *n*] [−**f** *disk*] [*source... dest*]

DESCRIPTION

Dsc50 performs many DOS/50 functions on an 8550 diskette. The operation to perform can be specified as a command line option, in an interactive mode, or as the name of the command. Specifying an operation in the command name is equivalent to invoking "dsc50" with the corresponding operation option. For example:

```
wr50 /usr/daveh/tset /SOURCES/SETUP
```

is the same as typing:

```
dsc50 -w /usr/daveh/tset /SOURCES/SETUP
```

Note: because **dsc50** deals with only one 8550 diskette at a time, 8550 pathnames are not prefixed by /VOL/volume_name.

Source and *dest* are **TNIX** and 8550 files or directories on which the operations are to be performed.

In 8550 pathnames, '.' refers to the current directory,'..' to its parent directory (see the −**c** option).

Any directory appearing as a *source* file specifies all files within that directory (see the recursive flag, *−r*). *Source* subdirectories (if no *−r* and not performing a table of contents operation) and inaccessible files are ignored. If more than one *source* file is specified or if *source* is a directory, *dest* must be a directory. If *dest* is a directory, the *source* file(s) will be copied (or linked, depending on the operation) into the corresponding file(s) within that directory. For example, given that /mystuff is a directory on the 8550 diskette,

```
dsc50 -w /usr/daveh/adb.hex /usr/src/sccs.src /mystuff
```

will copy into the 8550 diskette files /mystuff/adb.hex and /mystuff/sccs.src. For copy operations, if the *dest* file already exists, it will be overwritten. For link operations, if the *dest* file already exists, it will be unlinked.

Interactive mode is entered by invoking **dsc50** with no operation option. Once in this mode, **dsc50** will read commands from standard in until the −**e** option is specified or the end-of-file is encountered. Commands in interactive mode have the same syntax as **dsc50** with the following exceptions:

The command name is omitted

An initial '-' is optional

Any line beginning with an exclamation mark (!) is given to sh(1) to be interpreted as a shell command.

OPTIONS

An operation is specified by one of the following letters. The default operation is to go into interactive mode.

−w	write. The **TNIX** *source* file(s) are copied to the 8550 *dest* file/directory. A *source* file name of − refers to standard in.
−m	makes a new DOS/50 volume. Similar to the −*w* operation, except the 8550 diskette is initialized (without physically formatting it) prior to copying the file(s).
−d	deletes the 8550 *source* files.
−x	extract. The 8550 *source* file(s) are copied to the **TNIX** *dest* file/directory. A *dest* file name of − refers to standard out.
−t	table of contents. A directory listing of the *source* file(s) is printed. If no *source* is specified, '.' is used.
−l	links the existing 8550 *source* file(s) to the 8550 *dest* file/directory.
−s	prints the setup (the current state of all options) to standard out in the form of a dsc50 command. Useful only in interactive mode.

The following options may be specified on the command line or in interactive mode.

−c *path*

Change the current 8550 directory to *path*. This will be the directory relative to which all 8550 pathnames not beginning with slash (/) are interpreted. An error will be reported if this directory does not exist. The default current directory is "/", the root directory of the 8550 diskette.

−o *owner*

specifies the owner name to be given to anything created on the 8550 diskette (files, directories, or the entire volume). The default owner is "NO.NAME".

−n *name*

specifies the volume name to be given to the 8550 diskette when it is created by the **−m** operation. The default name is "NO.NAME".

−a *n*	specifies the number (amount) of file slots to be created on the 8550 diskette by the **−m** operation. The default is 256 slots.
−f *disk*	specifies a file to use as the 8550 diskette instead of the default device, /dev/rfd0.

The following flags may be used on the command line or in interactive mode. In interactive mode, specification of one of these flags toggles that flag.

−e	exits. Terminates interactive mode after executing the given operation. Ignored if not in interactive mode.
−p	prompt. Ignored if not in interactive mode. If in interactive mode, **dsc50** will prompt for each command with a dash (-). The default is to not prompt in interactive mode.

−**b** Binary transfer. Because the "end of line" character differs between **TNIX** and DOS/50 text files, some character translation must occur when a text file is copied to or from the disk. This translation is performed by **dsc50** on all files unless the −**b** option is specified.

−**r** recursive action specified. If any directory appears in a *source* directory, the current command will be applied recursively to that subdirectory. Normally, subdirectories are skipped.

−**v** Normally **dsc50** does its work silently. The −**v** (verbose) option causes it to type the name of each file it treats preceded by the operation name. With the −**t** operation, −**v** prints the following information about each file:

the file type and capabilities of the file,

its modification date-time,

its owner's name,

its size,

and its link count.

−**q** query. Causes **dsc50** to pause before treating each file, type the operation name and the file name (as with −**v**) and await the user's response. Response **y** means 'yes', so the file is treated. Response **n** or a null response means 'no', and the file does not take part in whatever is being done. Response **N** is the same as answering **n** to this and all further questions involved in the current operation. Response **Y** is the same as answering **y** to this and all further questions involved in the current operation. End-of-file on standard in will exit the operation. Query does not take place for the table of contents (-t) or setup (-s) operations.

FILES

/dev/rfd0

NOTES

Invoking **dsc50** by any name other than those listed above is equivalent to invoking it by the name **dsc50** with no operation option.

Interrupting **dsc50** while it is modifying the DOS/50 disk (I.E. during a make, write, delete, or link operation) may corrupt that disk.

[This page intentionally left blank.]

DU(1)

NAME

du – summarize disk usage

SYNTAX

du [**–as**] [*file*] ...

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each specified directory or file. If *file* is missing, '.' is used. Absence of arguments causes an entry to be generated for each directory only.

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

OPTIONS

–s	causes only the grand total of each file or directory specified to be given.
–a	causes an entry to be generated for each file. The **–s** and **–a** options are mutually exclusive.

NOTES

Non-directories given as arguments (not under **–a** option) are not listed.
If there are too many distinct linked files, *du* counts the excess files multiply.

[This page intentionally left blank.]

ECHO(1)

NAME

echo – echo arguments

SYNTAX

echo [–n] [*arg*] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a newline on the standard output.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, do

```
        echo ... 1>&2
```

where ... is a diagnostic message.

OPTIONS

–n No newline is added to the output. This flag must preceed all other arguments in order to be recognised.

NOTES

Because **echo** is designed to generate a wide variety of messages, extra flags and illegal flags are treated as arguments to be echoed and do not generate an error.

[This page intentionally left blank.]

ED(1)

NAME

ed – text editor

SYNTAX

ed [**–cpx**] [*file*]

DESCRIPTION

Ed is the standard text editor.

If a *file* 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.

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.*

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. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to 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. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus \ [. and sometimes ^ * $.
2. A . matches any character.
3. A \ followed by any character except a digit or ( ) matches that character.
4. A nonempty string *s* bracketed *[s]* (or *[^s]*) matches any character in (or not in) *s.* In *s,* \ has no special meaning, and] may only appear as the first letter. A substring *a–b,* with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of form 1-4 followed by * matches a sequence of 0 or more matches of the regular expression.
6. A regular expression, *x,* of form 1-8, bracketed *\(x \)* matches what *x* matches.
7. A \ followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *nth* \(matched.
8. A regular expression of form 1-8, *x,* followed by a regular expression of form 1-7, *y* matches a match for *x* followed by a match for *y,* with the *x* match being as long as possible while still permitting a *y* match.

9. A regular expression of form 1-8 preceded by ^ (or followed by $), is constrained to matches that begin at the left (or end at the right) end of a line.

10. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.

11. An empty regular expression stands for a copy of 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 name *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.

5. A regular expression enclosed in slashes '/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.

6. A regular expression enclosed in question marks '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches 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 (or 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 (or 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 versions of the editor, the character '^' in addresses is 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 an insufficient number 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 first 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, most commands 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 line preceding the deleted lines.

(. , .)**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 *'filename'* 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. If *'filename'* is missing, the remembered name is used.

E *filename*

This command is the same as *e*, except that no diagnostic results when no *w* has been given since the last buffer alteration.

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 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, or, if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

(. , . +1) j
This command joins the addressed lines into a single line; intermediate newlines simply disappear. '.' is left at the resulting line.

(.) kx 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 two-digit octal, and long lines are folded. The *l* command may be placed on the same line after any non-i/o command.

(. , .) ma
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 non-i/o command.

P *c* If *c* is given it becomes the new prompt character. Otherwise, toggle prompting. The default prompt character is '*'. Initially prompting is turned off.

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

Q This command is the same as *q,* except that no diagnostic results when no *w* has been given since the last buffer alteration.

($) 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 file name is remembered if there was no remembered file name already. 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 newline 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 '\'. 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). The original lines are not deleted. '.' is left on the last line of the copy.

(. , .) **u**

The undo command restores the preceding contents of the current line, which must be the last line in which a substitution was made.

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

This command is the same as the global command *g* 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 0666 (readable and writable by everyone). The file name is remembered if there was no remembered file name already. 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 printed.

(1,$)W *filename*

This command is the same as *w,* except that the addressed lines are appended to the file.

x A key string is demanded from the standard input. Later *r, e* and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt (1).* An explicitly empty key turns off encryption.

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

!<shell command>
The remainder of the line after the '!' is sent to *sh (1)* to be interpreted as a command. ' . ' is unchanged.

(. +1) **<newline>**
An address alone on a line causes the addressed line to be printed. This line becomes the current line. 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 memory: each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.

OPTIONS

−x If −x is present, an *x* command is simulated first to handle an encrypted file.

−c The optional −c suppresses the printing of character counts by *e, r,* and *w* commands.

−p Turn on prompt (*).

FILES

/tmp/e*
ed.hup: work is saved here if terminal hangs up

SEE ALSO

sed(1), crypt(1)

DIAGNOSTICS

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

To protect against throwing away valuable work, a *q* or *e* command is considered to be in error, unless a *w* has occurred since the last buffer change. A second *q* or *e* will be obeyed regardless.

NOTES

The *l* command mishandles DEL.
A *!* command cannot be subject to a *g* command.
Because 0 is an illegal address for a *w* command, it is not possible to create an empty file with **ed**.

EXPR(1)

NAME

expr – evaluate arguments as an expression

SYNTAX

expr *expression*

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expr | *expr*
: yields the first *expr* if it is neither null nor '0', otherwise yields the second *expr*

expr & *expr*
: yields the first *expr* if neither *expr* is null or '0', otherwise yields '0'.

expr relop expr
: where *relop* is one of < <= = != >= >, yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expr* are integers, otherwise lexicographic.

expr + *expr*

expr - *expr*
: addition or subtraction of the arguments.

expr * *expr*

expr / *expr*

expr % *expr*
: multiplication, division, or remainder of the arguments.

expr : *expr*
: The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of *ed (1)* . The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

(*expr*)
: parentheses for grouping.

Examples:

To add 1 to the Shell variable *a* :

```
a=`expr $a + 1`
```

To find the filename part (least significant part) of the pathname stored in variable *a* , which may or may not contain '/':

```
expr $a : '.*/\(.*\)' \| $a
```

Note the quoted Shell metacharacters.

SEE ALSO

ed(1), sh(1), test(1)

DIAGNOSTICS

Expr returns the following exit codes:

0	if the expression is neither null nor '0',
1	if the expression is null or '0',
2	for invalid expressions.

FBR(1)

NAME

fbr – file backup and restore

SYNTAX

fbr -{**cruxtd**} [**-viw**] [**–l** *comment*] [**–m** *directory*] [**–f** *archive*] [–] [*file*] ...

DESCRIPTION

Fbr saves and restores directories and files on a floppy disk archive, preserving aliases (multiple links to the same file). Its actions are controlled by the *function* and optional arguments. Other arguments to the command are file or directory names specifying which files are to be transferred, deleted, or listed. A parameter consisting only of a dash indicates that file and directory names should be read from standard in at this point, one name per line. If reading names from standard in is not specified, the appearance of a directory name in the parameters refers to all files and (recursively) subdirectories of that directory. If *function* is **–x**, **–t**, or **–d**, filenames may contain regular expressions of the form acceptable to sh(1). Note that such expressions must be quoted so that the shell does not attempt to interpret them.

EXAMPLES

To archive the subtree rooted at the current directory and use the current directory name as the archive comment:

```
fbr -c -l ‘pwd‘
```

To archive all user files and directories that have changed in the last two days:

```
find /usr -mtime -2 -print | fbr -c -l ’incremental’ -
```

OPTIONS

Function is one of the following letters:

–c creates a new archive. The archive is initialized prior to writing the named files to it. If create is specified with no file argument, ‘.’ is the default.

It is suggested that the file arguments supplied be relative to the current directory (I.E. ‘abc’ rather than ‘/usr/dmr/abc’). Files stored in this way can be more easily transported between users and installations.

–r "Replace." The named files are written on the archive. If files with the same names already exist on the archive, 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 archive. –u is like –r , but a file is replaced only if its modification date is later than the date stored on the archive; that is to say, if it has changed since it was last archived.

–x extracts the named files from the archive to the file system. The access and modification date-times and the mode are restored. If **fbr** is being run by the superuser, the owner and owner’s group are also restored. If no file argument is given, the entire contents of the archive are extracted.

−t "Table of Contents." Lists the names of the specified files. If no file argument is given, the entire contents of the archive are listed.

−d deletes the named files from the archive. If no file argument is given, no files are deleted.

The following flags may be used in addition to the flag which selects the function desired.

−v Normally **fbr** does its work silently. The −v (verbose) option, when used with functions other than −t, causes **fbr** to type the name of each file it processes preceded by the function name. With the −t function, −v prints information from the archive label as well as the following information concerning each file:

The mode of the file,

the username and groupname of its owner,

its size,

its modification date-time,

the starting block number of its data on the archive,

and the name of the file.

For all functions, −v causes an archive space usage message to be printed at the completion of the command.

−i Errors reading the archive are noted, but no action is taken. Normally, errors cause a return to the command level. This flag is ignored if the *function* requires changes to be made to the archive (I.E. −i is only useful for performing tables of contents and file extraction).

−w causes **fbr** to wait before treating each file, type the function name and the file name (as with v) and await the user's response. Response y means 'yes', so the file is processed. Response **n** or a null response means 'no', and the file does not take part in whatever is being done. Response **N** is the same as answering **n** to this and all further questions. Response Y is the same as answering y to this and all further questions. An end-of-file on standard in or an interrupt at this point will terminate the program without modifying the archive or file system.

This flag is ignored if a table of contents function (-*t*) is specified.

−l *comment*
When used in conjuction with any function which alters the archive (I.E. -**d,-r,-u, or -c**) and *comment* is not a null string, this option causes **fbr** to write the *comment* string into the archive label. Otherwise, −l is ignored.

−f *archive*
uses *archive,* rather than the floppy disk, as the archive. If this option is used with the −c function and *archive* exists and has a non-zero size, that size will be the size of the created archive. Otherwise, the archive created will exactly contain the specified files (I.E. will have no free space).

−**m** *directory*
moves the specified files relative to *directory* rather than the root or current directory by temporarily prepending *directory* to each pathname to be operated on. Useful only with the −**x** function.

FILES

/tmp/fbr??????
/dev/fd0
/etc/passwd
/etc/group
/bin/mkdir or /usr/bin/mkdir

SEE ALSO

fbr(5), find(1), sh(1).

DIAGNOSTICS

Several, the non-obvious ones are:

Phase error - the file changed between the time it was selected for archiving and the time that it was archived.

Out of memory - not enough memory is available to store all the filenames to be operated on. The number of names and/or their lengths must be reduced.

NOTES

Pathnames are limited to 106 characters.

Multiple links to a file can only be recognized as such if the links in question are all backed up or restored in the same **fbr** invocation.

It is probably unwise to specify query (-*w*) and 'read names from standard in' ('-'), since names are read until EOF and the query exits if EOF.

[This page intentionally left blank.]

FILE(1)

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

It often makes mistakes. In particular it often suggests that command files are C programs.

[This page intentionally left blank.]

FIND(1)

NAME

find – find files

SYNTAX

find *pathname expression*

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the *pathname* list 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 (bits 0777) exactly match the octal number *onum* (see *chmod (1)*). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat (2)*) become significant and the permission is tested for existence of the bits in *onum* rather than exact match with *onum* .

–**type** *x*
: True if the type of the file is *x* , where *x* 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* (login name or numeric user ID).

–**group** *gname*
: True if the file belongs to group *gname* (group name or numeric group ID).

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

–**inum** *n*
: True if the file has inode number *n* .

–**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 a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument '{}' is replaced by the current pathname.

−ok *command*
Like **−exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response y .

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

−newer *file*
True if the current file has been modified more recently than the argument *file* .

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

2) The negation of a primary ('!' is the unary *not* operator).

3) Concatenation of primaries (the "and" operation is implied by the juxtaposition of two primaries).

4) Alternation of primaries ('**-o**' is the "or" operator).

EXAMPLE
To remove all files named 'a.out' or '*.o' that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

FILES
/etc/passwd
/etc/group

SEE ALSO
sh(1), test(1), filsys(5)

FORMAT(1)

NAME

format – write disk format

SYNTAX

format [**−s**]

DESCRIPTION

Format writes the appropriate formatting information to prepare a flexible diskette for subsequent data storage. It is recommended that all new diskettes be formatted before being used.

OPTIONS

−s	Format the disk single density. **Format's** default is to format in double density.

FILES

/dev/rfd0

SEE ALSO

format(8)

[This page intentionally left blank.]

GREP(1)

NAME

grep, egrep, fgrep – search a file for a pattern

SYNTAX

grep [**-vclnbshy**] ... [[**-e**] *expression*] [*file*] ...

egrep [**-vclnbsh**] ... [**-f** *file*] [[**-e**] *expression*] [*file*] ...

fgrep [**-vclnbshx**] ... [**-f** *file*] [*strings*] [*file*]

DESCRIPTION

Commands of the **grep** family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output; unless the –**h** flag is used, the file name is shown if there is more than one input file.

Grep patterns are limited regular expressions in the style of **ed (1)** ; it uses a compact nondeterministic algorithm. **Egrep** patterns are *extended* regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. **Fgrep** patterns are fixed strings; it is fast and compact.

Care should be taken when using the characters $ * [^ | ? ′ " ( ) and \ in the *expression* as they are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes ′ ′.

Fgrep searches for lines that contain one of the (newline-separated) *strings* .

Egrep accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character matches that character.

The character ^ ($) matches the beginning (end) of a line.

A . matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the string. Ranges of ASCII character codes may be abbreviated as in 'a–z0–9'. A] may occur only as the first character of the string. A literal – must be placed where it can't be mistaken as a range indicator.

A regular expression followed by * (+, ?) matches a sequence of 0 or more (1 or more, 0 or 1) matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is [] then *+? then concatenation then | and newline.

OPTIONS

−v All lines but those matching are printed.

−c Only a count of matching lines is printed.

−l The names of files with matching lines are listed (once) separated by newlines.

−s No output is produced, only status. Options −c, −l, and −s are mutually exclusive.

−n Each line is preceded by its line number in the file.

−b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

−h Do not print filename headers with output lines.

−y Lower case letters in the pattern will also match upper case letters in the input (**grep** only).

−e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a −.

−f *file* The regular expression (**egrep**) or string list (**fgrep**) is taken from the *file* .

−x (Exact) only lines matched in their entirety are printed (**fgrep** only).

SEE ALSO
ed(1), sed(1), sh(1)

DIAGNOSTICS
Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

NOTES
Ideally there should be only one **grep** , but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

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

KILL(1)

NAME

kill – terminate a process with extreme prejudice

SYNTAX

kill [– *signal*] *processid* ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. If a signal number preceded by '–' is given as first argument, that signal is sent instead of terminate (see **signal (2)**). This will kill processes that do not catch the signal; in particular 'kill –9 ...' is a sure kill.

By convention, if process number 0 is specified, all members in the process group (i.e. processes resulting from the current login) are signaled.

The killed processes must belong to the current user unless he is the super-user. To shut the system down and bring it up single user the super-user may use 'kill –1 1'; see **init (8)**.

The process number of an asynchronous process started with '&' is reported by the shell. Process numbers can also be found by using **ps (1)**.

SEE ALSO

ps(1), kill(2), signal(2)

[This page intentionally left blank.]

LIBGEN(1)

NAME

libgen—library generator

SYNTAX

libgen [**–c** *commandfile*] [**–d** *modulename*...] [**–h** *string*] [**–i** *objfile*...] [**–n** *newlib*] [**–o** *oldlib*] [**–r** *objfile*] [**–v**] [**–x** *modulename* [*pathname*]]

DESCRIPTION

The Library Generator *libgen* is a general-purpose utility program used to create and maintain object module libraries for use with the linker.

Libgen collects assembler-generated or compiler-generated object modules into library files. From these library files, the object modules can be individually accessed by the linker, based on the information provided by each object module.

OPTIONS

–c *commandfile*	Invokes a *libgen* command file.
–d *modulename...*	Deletes library module(s).
–h *string*	Specifies the header for the new library.
–i *objfile...*	Inserts new module(s) into the library.
–n *newlib*	Designates a new library file.
–o *oldlib*	Specifies an old library file.
–r *objfile...*	Replaces an old module(s) with a new module(s).
–v	Generate informative messages. A banner containing the version number will be printed and switches will be listed as they are processed.
–x *modulename* [*pathname*]	Extracts (copies) a module to an object file.
pathname	The name of the file.
commandfile	The pathname of the command file containing a series of *libgen* command options.
modulename	The name of the library module.
string	An ASCII string that identifies the library. The *string* may contain any printable ASCII characters but should not start with a dash (–). The maximum length of *string* is 80 characters.

FILES

###.lib.tmp.nn	temporary file (where nn is an integer).

SEE ALSO

8500 Modular MDL Series B Assembler Core Users Manual
lstr*(1)*

[This page intentionally left blank.]

LINK(1)

NAME

link—link object modules

SYNTAX

link [**-CDLOcdmorstvx**] [*parameters*]

DESCRIPTION

Link merges one or more independently assembled object files into a load file, suitable for loading into memory. Assembler-generated files, library files, or linked files may be used as linker input.

OPTIONS

-C	Assigns a classname to one or more section(s).
-D	Defines a global symbol at link time.
-L	Locates a class or section to a specified memory area.
-O	Specifies the object module, library, and linked load files to be linked by **link**.
-c	Invokes a linker command file.
-d	Debug information is generated.
-m	Defines a memory map configuration.
-o	Designates the output file for the linked code.
-r	Relink information is generated.
-s	Specifies global symbol files to be linked.
-t	Specifies relocation type of named class(es) or section(s).
-v	Specifies a verbose listing. Prints a banner with the version number; lists switches on the standard output as they are processed, lists the number of errors, lists the transfer address, tells whether the file is relinkable, and whether debugging information is in the file.
-x	Specifies load module transfer address.

SEE ALSO

8500 Modular MDL Series B Assembler Core Users Manual
lstr*(1)*

[This page intentionally left blank.]

LN(1)

NAME

ln – make a link

SYNTAX

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* .

SEE ALSO

rm(1)

NOTES

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

[This page intentionally left blank.]

LPR(1)

NAME

lpr – line printer spooler

SYNTAX

lp1r [**–cmr**] [*file*] ...
lp2r [**–cmr**] [*file*] ...

DESCRIPTION

Lp1r or **lp2r** cause the *files* to be queued for printing on line printer 1 or 2 respectively. If no files are named, or if the file '–' is encountered, the standard input is read.

OPTIONS

–r	Remove the file when it has been queued.
–c	Copy the file to insulate against changes that may happen before printing.
–m	Report by **mail(1)** when printing is complete. This is automatic if lpr can't link to the file to print.

FILES

/usr/spool/lp?/lock
/usr/spool/lp?/cf* data file
/usr/spool/lp?/df* daemon control file
/usr/spool/lp?/tf* temporary version of control file
/dev/lp?

SEE ALSO

lpd(8)

[This page intentionally left blank.]

LOGIN(1)

NAME

login – sign on

SYNTAX

login [*username*]

DESCRIPTION

The **login** command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one described here.

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, the user is informed of the existence of *.mail* and message-of-the-day files. **Login** initializes the user and group IDs and the working directory, then executes a command interpreter (usually **sh(1)**) according to specifications found in a password file. Argument 0 of the command interpreter is '–sh'.

Login is recognized by **sh(1)** and executed directly (without forking).

FILES

/etc/motd	message-of-the-day
/etc/passwd	password file

SEE ALSO

init(8), newgrp(1), getty(8), mail(1), passwd(1), passwd(5)

DIAGNOSTICS

'Login incorrect,' if the name or the password is bad.
'No Shell', 'cnnnot open password file'.

[This page intentionally left blank.]

LS(1)

NAME

ls – list contents of directory

SYNTAX

ls [**−ltasrucig**] [*file*] ...
ls [**−ltasdrucig**] [*directory*] ...

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.

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; normally ' . ' and ' .. ' are suppressed.
−s	Give size in blocks, including indirect 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).
−c	Use time of last modification to inode (mode, etc.) instead of last modification to file for sorting (−t) or printing (−l). Options −u and −c are mutually exclusive.
−i	Print i-number in first column of the report for each file listed.
−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 (normally 'x' or '–') is t if the 1000 bit of the mode is on. See **chmod(1)** for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

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

LSTR(1)

NAME

lstr—list symbols found in B series object format file

SYNTAX

lstr [**-ghnosuv**] [*file* ...]

DESCRIPTION

Lstr prints the symbol table of each object *module* in the argument list. If an argument is a library, a listing for each object file in the library will be produced.

Each symbol is preceded by its value (zeros if undefined) and a single letter. The single letter must be one of the following:

- **U**—undefined;
- **A**—absolute section;
- **S**—SECTION section;
- **D**—global data, entry, or constant;
- **C**—COMMON section;
- **R**—RESERVE section;
- **u**—undefined local data, entry, or constant; or
- **l**—local data, entry, or constant.

The output is sorted alphabetically.

OPTIONS

-g	Print only global (external) symbols.
-h	Print the header in a library inserted with the **libgen***(1)* **-h** switch.
-n	Sort numerically rather than alphabetically.
-o	Prepend file or library element name to each output line rather than only once.
-s	Append the length of sections to their output lines.
-u	Print only undefined symbols.
-v	Print the version number of **lstr***(1)*.

SEE ALSO

libgen*(1)*, **link***(1)*, **nm***(6)*

[This page intentionally left blank.]

MAIL(1)

NAME

mail – send or receive mail among users

SYNTAX

mail *person* ...
mail [**−rqp**] [**−f** *file*]

DESCRIPTION

Mail with no argument prints a user's mail, message-by-message, in last-in, first-out order. For each message, **mail** reads a line from the standard input to direct disposition of the message.

newline
Go on to next message.

d Delete message and go on to the next.

p Print message again.

– Go back to previous message.

s *[file] ...*
Save the message in the named *files* ('mbox' default).

w *[file] ...*
Save the message, without a header, in the named *files* ('mbox' default).

m *[person] ...*
Mail the message to the named *persons* (yourself is default).

EOT (control-D)
Put unexamined mail back in the mailbox and stop.

q Same as EOT.

x Exit, without changing the mailbox file.

! *command*
Escape to the Shell to do *command* .

? Print a command summary.

An interrupt stops the printing of the current letter.

When *persons* are named, **mail** takes the standard input up to an end-of-file (or a line with just '.') and adds it to each *person's* 'mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *person* is usually a user name recognized by **login(1)** .

Each user owns his own mailbox, which is by default generally readable but not writable. The command does not delete an empty mailbox nor change its mode, so a user may make it unreadable if desired.

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

OPTIONS

−r	causes the mailbox to be printed in first-in, first-out order rather than the default.
−p	prints the mailbox without questions.
−q	causes **mail** to exit after interrupts without changing the mailbox, rather than just stopping printing of the current letter.
−f *file*	causes the given file to be printed as if it was the mail file.

FILES

/usr/spool/mail/*	mailboxes
/etc/passwd	to identify sender and locate persons
mbox	saved mail
/tmp/ma*	temp file
dead.letter	unmailable text

SEE ALSO

xsend(1), write(1)

NOTES

There is a locking mechanism intended to prevent two senders from accessing the same mailbox, but it is not perfect and races are possible.

MAKE(1)

NAME

make – maintain program groups

SYNTAX

make [**–iknträs**][–f *makefile*][*file*] ...

DESCRIPTION

Make executes commands in *makefile* to update one or more target *files* . The default is to use the script in '**makefile**' (or '**Makefile**' if '**makefile**' does not exist.)

Make updates or creates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target.

Pound sign (#) and newline surround comments.

The following makefile says that 'pgm' depends on two files 'a.o' and 'b.o', and that they in turn depend on '.s' files and a common file 'incl'. It gives the rule for making "pgm" from "a.o" and "b.o", and the rules for making "a.o" and "b.o" from "a.s" and "b.s".

```
pgm: a.o b.o
        ld a.o b.o -lm -o pgm
a.o: incl a.s
        as -o a.o a.s
b.o: incl b.s
        as -o b.o b.s
```

Makefile entries of the form

```
string1 = string2
```

are macro definitions. Subsequent appearances of *$(string1)* are replaced by *string2* . If *string1* is a single character, the parentheses are optional.

Make infers prerequisites for files for which *makefile* gives no construction commands. For example, a '.s' file may be inferred as prerequisite for a '.o' file and be assembled to produce the '.o' file. Thus the preceding example can be done more briefly:

```
pgm: a.o b.o
        ld a.o b.o -lm -o pgm
a.o b.o: incl
```

The **make** program itself does not know what file name suffixes are interesting or how to transform a file with one suffix into a file with another suffix. This information is stored in an internal table that has the form of a description file. If the -r flag is used, this table is not used.

A user may add new suffixes and change or add default rules to his description files. The format of the suffix list and the transformation rules follow.

The list of suffixes is actually the dependency list for the name "SUFFIXES"; **make** looks for a file with any of the suffixes on the list. If such a file exists, and if there is a transformation rule for that combination, **make** acts as described earlier.

The transformation rule names are the concatenation of the two suffixes. The name of the rule to transform a ".r" file to a ".o" file is thus ".r.o". If the rule is present and no explicit command sequence has been given in the user's description files, the command sequence for the rule ".r.o" is used. If a command is generated by using one of these suffixing rules, the macro $* is given the value of the stem (everything except the suffix) of the name of the file to be made, $@ is the full name of the file to be made, $< is the list of the prerequisites, and $? is the list of prerequisites that are out of date.

The order of the suffix list is significant, since it is scanned from left to right, and the first name that is formed that has both a file and a rule associated with it is used. If new names are to be appended, the user can just add an entry for "SUFFIXES" in his own description file; the dependents will be added to the usual list. A "SUFFIXES" line without any dependents deletes the current list. (It is necessary to clear the current list if the order of names is to be changed).

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target '.SILENT' is in *makefile* , or the first character of the command is '@'.

Commands returning nonzero status cause **make** to terminate unless the special target '.IGNORE' is in *makefile* or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target depends on the special name '.PRECIOUS'.

OPTIONS

−f *makefile*
: the given file is used as the script rather than '**makefile**' or '**Makefile**' . If *makefile* is '-' the standard input is used as the script. More than one **−f** option may appear.

−i Equivalent to the special entry '.IGNORE:'.

−k When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

−n Trace and print, but do not execute the commands needed to update the targets.

−t Touch, i.e. update the modified date of targets, without executing any commands.

−r Equivalent to an initial special entry '.SUFFIXES:' with no list.

−s Equivalent to the special entry '.SILENT:'.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1)

NOTES

Some commands return nonzero status inappropriately. Use −i to overcome the difficulty , or start that particular command with <tab><hyphen>.
Commands that are directly executed by the shell, notably **cd(1)** , are ineffectual across newlines in **make** .

[This page intentionally left blank.]

MAN(1)

NAME

man - print sections of this manual

SYNTAX

man [*chapter*] [*title*] ...

DESCRIPTION

Man locates and prints the section of this manual named title in the specified chapter. (In this context, the word 'page' is often used as a synonym for 'section'.) The title is entered in lower case. The chapter number does not need a letter suffix. If no chapter is specified, the whole manual is searched for title and all occurences of it are printed.

FILES

/usr/man/cat?/*

SEE ALSO

man(7).

[This page intentionally left blank.]

MESG(1)

NAME

mesg – permit or deny messages

SYNTAX

mesg [**–ny**]

DESCRIPTION

Mesg reports the current state of message permission.

OPTIONS

–n	denies other users messages to this user via **write (1)** by revoking non-user write permission on the user's terminal.
–y	similarly reenstates permission. Options –y and –n are mutually exclusive.

FILES

/dev/tty*
/dev

SEE ALSO

write(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

[This page intentionally left blank.]

MKDIR(1)

NAME

mkdir – make a directory

SYNTAX

mkdir *dirname ...*

DESCRIPTION

Mkdir creates specified directories in mode 0777 (readable, writeable, searchable by everyone). These default permissions can be removed by the umask (see **umask(2)**). Standard entries, ' . ' , for the directory itself, and ' .. ' for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

rm(1),chmod(1),umask(2)

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

[This page intentionally left blank.]

MLOAD(1)

NAME

mload—TEKHEX upload/download

SYNTAX

mload [**-uds**] [**-p** *prompstring*] *name*

DESCRIPTION

Mload provides the host-side protocol for memory image transfers to or from TNIX, using the standard or extended TEKHEX transfer protocol. *Name* is a file containing (or to contain) the extended TEKHEX formatted memory image.

OPTIONS

-u	Specifies upload, e.g., 8002 to 8560.
-d	Specifies download, e.g., 8560 to 8002 (the default). Options **-u** and **-d** are mutually exclusive.
-p prompstring	An optional input prompt.
-s	Slow down the transfer. A slowed transfer is required by TEKDOS on the 8002.

SEE ALSO

stty*(1)*, **uload***(1)*

NOTES

The echo and lcase modes (set with the **stty***(1)* command) are disabled during the transfer.

[This page intentionally left blank.]

MV(1)

NAME

mv – move or rename files and directories

SYNTAX

mv *file1 file2*

mv *file ... directory*

mv *directory1 directory2*

DESCRIPTION

Mv moves (changes the name of) *file1* to *file2* .

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, **mv** prints the mode (see **chmod (2)**) and reads the standard input to obtain a line; if the line begins with y , the move takes place; if not, **mv** exits.

In the second form, one or more *files* are moved to the *directory* with their original file-names.

In the third form, the subtree rooted at *directory1* is moved to *directory2* . *Directory2* must not be in the *directory1* subtree.

SEE ALSO

cp(1), chmod(2)

NOTES

Mv refuses to move a file onto itself.

If *file1* and *file2* lie on different file systems, **mv** must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

[This page intentionally left blank.]

NEWGRP(1)

NAME

newgrp – log in to a new group

SYNTAX

newgrp *group*

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to **login(1)** . 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 a user logs in, s/he is a member of the group specified for that user in the password file, /etc/passwd. **Newgrp** is known to the shell, which executes it directly without a fork.

FILES

/etc/group, /etc/passwd

SEE ALSO

login(1), group(5)

[This page intentionally left blank.]

NICE(1)

NAME

nice, nohup – run a command at low priority

SYNTAX

nice [*–number*] *command*

nohup *command*

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

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

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. **Nohup** should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal.

FILES

nohup.out standard output and standard error file under *nohup*

SEE ALSO

nice(2)

DIAGNOSTICS

Nice returns the exit status of the subject command.

[This page intentionally left blank.]

OD(1)

NAME

od – octal dump

SYNTAX

od [**–bcdox**] [**-s** *offset*[.][**b**]] [**-e** *offset*[.][**b**]] [*file*]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the flags. If no flags are present, **–o** is default.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

A dump line consisting of a lone '*' indicates repetition of the previously dumped data.

OPTIONS

–b	Interpret bytes in octal.
–c	Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
–d	Interpret words in decimal.
–o	Interpret words in octal.
–x	Interpret words in hex.

–s *offset*

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 of 512 bytes.

-e *offset*

Same as **-s** but specifies where dumping is to end. Default is dumping until end-of-file.

SEE ALSO

adb(1)

[This page intentionally left blank.]

PASSWD(1)

NAME

passwd – change login password

SYNTAX

passwd [*name*]

DESCRIPTION

This command changes (or installs) a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(5), crypt(3)

[This page intentionally left blank.]

PR(1)

NAME

pr – print file

SYNTAX

pr [**–ntm**] [**-h** *header*] [**-c** *n*] [**-p** *n*] [**-w** *n*] [**-l** *n*] [**-s** *c*] [*file*] ...

DESCRIPTION

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

Inter-terminal messages via **write (1)** are forbidden during a **pr** .

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

OPTIONS

–c *n* Produce *n* -column output.

–p *n* Begin printing with page *n* .

–h *header*
Use *header* as the page header.

–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.

–m Print all *files* simultaneously, each in one column,

-n Add line numbers.

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(1)

DIAGNOSTICS

There are no diagnostics when **pr** is printing on a terminal.

[This page intentionally left blank.]

PS(1)

NAME

ps − process status

SYNTAX

ps [**-alx**]

DESCRIPTION

Ps prints certain data about active processes. The short (default) listing contains the process ID, tty number, the cumulative execution time of the process and an approximation to the command line.

The long listing is columnar and contains

F	Flags associated with the process. 01: in memory; 02: system process; 04: locked in memory (e.g. for physical I/O); 10: being swapped; 20: being traced by another process.
S	The state of the process. 0: nonexistent; S: sleeping; W: waiting; R: running; I: intermediate; Z: terminated; T: stopped.
UID	The user ID of the process owner.
PID	The process ID of the process; it is possible to kill a process if you know its process ID.
PPID	The process ID of the parent process.
CPU	Processor utilization for scheduling.
PRI	The priority of the process; high numbers mean low priority.
NICE	Used in priority computation.
ADDR	The memory address of the process if resident, otherwise the disk address.
SZ	The size in blocks of the memory image of the process.
WCHAN	The event for which the process is waiting or sleeping; if blank, the process is running.
TTY	The controlling tty for the process.
TIME	The cumulative execution time for the process.

The command and its arguments.

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>. **Ps** makes an educated guess as to the file name and arguments given when the process was created by examining 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.

OPTIONS

−a	display information about all processes with terminals rather than only one's own processes.
−x	even display information about processes with no terminal.
−l	produce a long listing.

FILES

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

SEE ALSO

kill(1)

NOTES

Things can change while **ps** is running; the picture it gives is only a close approximation to reality.
Some data printed for defunct processes is irrelevant.

PWD(1)

NAME
pwd – working directory name

SYNTAX
pwd

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

SEE ALSO
cd(1)

[This page intentionally left blank.]

RM(1)

NAME

rm, rmdir – remove (unlink) files

SYNTAX

rm [−**fri**] *file* ...

rmdir *dir* ...

DESCRIPTION

Rm removes the entries for one or more *file(s)* 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 and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains.

If a designated file is a directory, an error comment is printed.

OPTIONS

-f	forces a file with no write permission to be deleted without question.
-r	recursively delete the entire contents of the specified directory, including the directory itself.
-i	(interactive) **rm** asks whether to delete each file, and, under -r , whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2)

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file '..', merely to avoid the antisocial consequences of inadvertently doing something like 'rm −r .*'.

[This page intentionally left blank.]

SH(1)

NAME

sh, for, case, if, while, : , . , break, continue, cd, eval, exec, exit, export, login, logout, newgrp, read, readonly, set, shift, times, trap, umask, wait – command language

SYNTAX

sh [**−ceiknrstuvx**] [*arg*] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file.

Commands

A *simple-command* is a sequence of non–blank *words* separated by blanks (a blank is a **tab** or a **space**). The first word specifies the name of the command to be executed. Except as specified below the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see **exec (2)**). The *value* of a simple-command is its exit status if it terminates normally or 200+*status* if it terminates abnormally (see **signal (2)** for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by | . The standard output of each command but the last is connected by a **pipe (2)** to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more *pipelines* separated by ; , **&** , **&&** or || and optionally terminated by ; or **&** . ; and **&** have equal precedence which is lower than that of **&&** and || . **&&** and || also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol **&&** (||) causes the *list* following to be executed only if the preceding *pipeline* returns a zero (non zero) value. Newlines may appear in a *list* , instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. The value returned by a command is that of the last simple-command executed in the command.

for name [in word ...] do list done
: Each time a **for** command is executed *name* is set to the next word in the **for** word list. If **in word** ... is omitted then **in "$@"** is assumed. Execution ends when there are no more words in the list.

case word in [pattern [| pattern] ...) list ;;] ... esac
: A **case** command executes the *list* associated with the first pattern that matches *word* . The form of the patterns is the same as that used for file name generation.

if list then list [elif list then list] ... [else list] fi
: The *list* following **if** is executed and if it returns zero the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero the *list* following **then** is executed. Failing that the **else** *list* is executed.

while list [do list] done

A **while** command repeatedly executes the **while** *list* and if its value is zero executes the **do** *list* ; otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list* . **until** may be used in place of **while** to negate the loop termination test.

(list) Execute *list* in a subshell.

{ list } *list* is simply executed.

The following words are only recognized as the first word of a command and when not quoted.

if then else elif fi case in esac for while until do done { }

Command substitution

The standard output from a command enclosed in a pair of grave accents (` `) may be used as part or all of a word; trailing newlines are removed.

Parameter substitution

The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set** . Variables may be set by writing

name = *value* [*name* = *value*] ...

${parameter}

A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters *** @ # ? − $!** . The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is a digit then it is a positional parameter. If *parameter* is ***** or **@** then all the positional parameters, starting with **$1** , are substituted separated by spaces. **$0** is set from argument zero when the shell is invoked.

${parameter−word}

If *parameter* is set then substitute its value; otherwise substitute *word* .

${parameter= word}

If *parameter* is not set then set it to *word* ; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${parameter ? word}

If *parameter* is set then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${parameter+word}

If *parameter* is set then substitute *word* ; otherwise substitute nothing.

In the above *word* is not evaluated unless it is to be used as the substituted string. (So that, for example, echo ${d–`pwd`} will only execute *pwd* if *d* is unset.)

The following *parameters* are automatically set by the shell.

#	The number of positional parameters in decimal.
–	Options supplied to the shell on invocation or by **set** .
?	The value returned by the last executed command in decimal.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following *parameters* are used but not set by the shell.

HOME	The default argument (home directory) for the **cd** command.
PATH	The search path for commands (see **execution**).
MAIL	If this variable is set to the name of a mail file then the shell informs the user of the arrival of mail in the specified file.
PS1	Primary prompt string, by default '$ '.
PS2	Secondary prompt string, by default '> '.
IFS	Internal field separators, normally **space** , **tab** , and **newline** .

Blank interpretation
After parameter and command substitution, any results of substitution are scanned for internal field separator characters (those found in **$IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ′′) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File name generation
Following substitution, each command word is scanned for the characters ***** , ? and **[** . If one of these characters appears then the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern then the word is left unchanged. The character . at the start of a file name or immediately following a / , and the character / , must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the characters enclosed. A pair of characters separated by – matches any character lexically between the pair.

Quoting
The following characters have a special meaning to the shell and cause termination of a word unless quoted.

; **&** ( ) | < > **newline** **space** **tab**

A character may be *quoted* by preceding it with a \ . \ **newline** is ignored. All characters enclosed between a pair of quote marks (′′), except a single quote, are quoted. Inside double quotes (" ") parameter and command substitution occurs and \ quotes the characters \ ` and **$**.

"**$***" is equivalent to "**$1 $2** ..." whereas
"**$@**" is equivalent to "**$1**" "**$2**" ... .

Prompting
When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a newline is typed and further input is needed to complete a command then the secondary prompt (**$PS2**) is issued.

Input output
Before a command is executed its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are not passed on to the invoked command. Substitution occurs before *word* or *digit* is used.

< **word** Use file *word* as standard input (file descriptor 0).

> **word** Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise it is truncated to zero length.

>> **word**
Use file *word* as standard output. If the file exists then output is appended (by seeking to the end); otherwise the file is created.

<< **word**
The shell input is read up to a line the same as *word* , or end of file. The resulting document becomes the standard input. If any character of *word* is quoted then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, \newline is ignored, and \\ is used to quote the characters \\ **$** ` and the first character of *word* .

< **& digit**
The standard input is duplicated from file descriptor *digit* ; see **dup (2)** . Similarly for the standard output using > .

< **&** – The standard input is closed. Similarly for the standard output using > .

If one of the above is preceded by a digit then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example,

```
... 2>&1
```

creates file descriptor 2 to be a duplicate of file descriptor 1.

If a command is followed by **&** then the default standard input for the command is the empty file (/dev/null). Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input output specifications.

Environment
The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see **exec (2)** and **environ (5)** . The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a *parameter* for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these *parameters* or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's *parameter* to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to *parameters* . Thus these two lines are equivalent

```
TERM=450 cmd args
(export TERM; TERM=450; cmd args)
```

Signals
The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&** ; otherwise signals have the values inherited by the shell from its parent. (But see also **trap** .)

Execution
Each time a command is executed the above substitutions are carried out. Except for the 'special commands' listed below a new process is created and an attempt is made to execute the command via an **exec (2)** .

The shell parameter **$PATH** defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (:) . The default path is **:/bin:/usr/bin** . If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an *a.out* file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands
The following commands are executed in the shell process and except where specified no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. file Read and execute commands from *file* and return. The search path **$PATH** is used to find the directory containing *file* .

break [n]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.

continue [n]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n -th* enclosing loop.

cd [arg]
Change the current directory to *arg* . The shell parameter **$HOME** is the default *arg* .

eval [arg ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [arg ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input output arguments may appear and, if no other arguments are given, cause the shell input output to be modified.

exit [n]
Causes a non interactive shell to exit with the exit status specified by *n* . If *n* is omitted then the exit status is that of the last command executed. (An end of file will also exit from the shell. See **invocation** below.)

export [name ...]
The given names are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given then a list of the names to be exported is printed.

login [arg ...]
Equivalent to 'exec login arg ...'.

logout Terminates the current shell, logging out if the current shell is a login shell.

newgrp [arg ...]
Equivalent to 'exec newgrp arg ...'.

read name ...
One line is read from the standard input; successive words of the input are assigned to the variables *name* in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

readonly [name ...]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given then a list of all readonly names is printed.

set [−eknptuvx [arg ...]]

−e	If non interactive then exit immediately if a command fails.
−k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
−n	Read commands but do not execute them.
−t	Exit after reading and executing one command.
−u	Treat unset variables as an error when substituting.
−v	Print shell input lines as they are read.
−x	Print commands and their arguments as they are executed.
−	Turn off the −**x** and −**v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−** .

Remaining arguments are positional parameters and are assigned, in order, to **$1** , **$2** , etc. If no arguments are given then the values of all names are printed.

shift The positional parameters from **$2** ... are renamed **$1** ...

times Print the accumulated user and system times for processes run from the current shell.

trap [arg] [n] ...

Arg is a command to be read and executed when the shell receives signal(s) *n* . (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by invoked commands. If *n* is 0 then the command *arg* is executed on exit from the shell, otherwise upon receipt of signal *n* as numbered in **signal (2)** . *Trap* with no arguments prints a list of commands associated with each signal number.

umask [nnn]

The user file creation mask is set to the octal value *nnn* (see **umask (2)**). If *nnn* is omitted, the current value of the mask is printed.

wait [n]

Wait for the specified process and report its termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

Invocation

If the first character of argument zero is – , this is a login shell. Commands are read from **/etc/profile** , then from **$HOME/.profile** , if such files exist, before commands are read from any other source. A login shell does not terminate when end-of-file is reached on its standard in. Instead it prints a short message suggesting that the **logout** command be used.

OPTIONS

The following flags are interpreted by the shell when it is invoked.

–c *string*

If the **–c** flag is present then commands are read from *string* .

–s If the **–s** flag is present or if no arguments remain then commands are read from the standard input. Shell output is written to file descriptor 2.

–i If the –i flag is present or if the shell input and output are attached to a terminal (as told by **gtty()** - see **ioctl(2)**) then this shell is *interactive* . In this case the terminate signal SIGTERM (see **signal (2)**) is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptable). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are the same as for the **set** command.

–e If non interactive then exit immediately if a command fails.

–k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

–n Read commands but do not execute them.

–t Exit after reading and executing one command.

−u	Treat unset variables as an error when substituting.
−v	Print shell input lines as they are read.
−x	Print commands and their arguments as they are executed.
−	Turn off the −**x** and −**v** options.

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

test(1), exec(2),

DIAGNOSTICS

Errors detected by the shell, such as syntax errors cause the shell to return a non zero exit status. If the shell is being used non interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

SLEEP(1)

NAME

sleep – suspend execution for an interval

SYNTAX

sleep *time*

DESCRIPTION

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

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

or to execute a command every so often, as in:

```
while true
do
     command
     sleep 37
done
```

SEE ALSO

alarm(2), sleep(3)

NOTES

Time must be less than 65536 seconds.

[This page intentionally left blank.]

SLP(1)

NAME

slp – set line printer characteristics

SYNTAX

slp *printer options...*

DESCRIPTION

Slp modifies the output characteristics of *printer* according to *options,* exiting with a status of two (2) if the command was invoked improperly, one (1) if some other problem occurred, or zero (0) if all went well.

Printer is the special file for the desired printer (e.g. /dev/lp1).

OPTIONS

nl The attached printer processes newlines - perform no newline processing.

–nl The attached printer does not process newlines - replace newline with carriage return line feed on output.

nl=string
Replace newline on output with *string.* The *string* is copied character for character except that a backslash followed by one to three octal digits is copied as the corresponding ascii character e.g. nl="\000\014" replaces newline with a null followed by a form feed. Only the first seven characters resulting from *string* are copied. A zero length *string* is illegal.

tabs The attached printer processes tabs - perform no tab processing.

–tabs The attached printer does not process tabs - replace tabs with the appropriate number of spaces on output.

SEE ALSO

stty(1)

NOTES

All *options* must follow the *printer* parameter.

The printer characteristics revert to the default each time the system is booted.

Because each *option* is processed independently, all *options* preceeding a bad *option* have been processed even though the command "failed".

The **slp** command only sets characteristics for printers connected to the aux devices. Use the *stty(1)* command for printers on IOP ports.

[This page intentionally left blank.]

SORT(1)

NAME

sort – sort or merge files

SYNTAX

sort [**–mubdfinrc**] [**-t** *c*] [+*pos1* [-*pos2*]] ... [**-o** *file*] [**-T** *directory*] [*file*] ...

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The file '–' means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes.

OPTIONS

The ordering of the sort is affected globally by the following options.

–b Ignore leading blanks (spaces and tabs) in field comparisons.

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

–f Fold upper case letters onto lower case.

–i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

–n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **–n** implies option **–b** .

–r Reverse the sense of comparisons.

-t *c* 'Tab character' separating fields is *c* .

The notation +*pos1* -*pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2* . *Pos1* and *pos2* each have the form *m.n* , optionally followed by one or more of the flags **bdfinr** , where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2* . A missing *.n* means .0; a missing – **pos2** means the end of the line. Under the **–t c** option, fields are strings separated by *c* ; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

–c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

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

–o *file* 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.

−T *directory*
The next argument is the name of a directory in which temporary files should be made.

−u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

Examples Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort −u +0f +0 list
```

Print the password file (**passwd (5)**) sorted by user id number (the 3rd colon-separated field).

```
sort −t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort −um +0 −1 dates
```

FILES
/usr/tmp/stm*, /tmp/*: first and second tries for temporary files

SEE ALSO
uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS
Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **−c** .

NOTES
Very long lines are silently truncated.

STTY(1)

NAME

stty—set terminal options

SYNTAX

stty [option] ...

DESCRIPTION

Stty sets certain I/O options on the current output terminal. With no argument, it reports the current settings of the options.

OPTIONS

An appended "i" refers to input, an "o" refers to output. An appended "{io}" represents "eveni" and "eveno".

-f	Force the change specified to occur even if the terminal characteristic table has been changed. The characteristic table is a larger data structure associated with the terminal. This table can be set via **ioctl***(2)* to change more terminal characteristics than are possible through **stty**. Without this flag, you will receive a warning message if the characteristic table has been changed and you attempt to set an option. The option will not be set when this warning is given.
even{io}	Even parity on input, output.
odd{io}	Odd parity on input, output.
nopar{io}	No parity (bit 7 is always 0) on input, output.
mark{io}	Mark parity (bit 7 is always 1) on input, output.
data{io}	Data on input, output (8th bit is not stripped).
nocare{io}	Don't care parity on input, output.
raw	Raw mode input (no erase, kill, interrupt, quit, logout; parity bit passed back).
-raw	Negate raw mode.
cooked	Negate raw mode.
cbreak	Make each character available to **read***(2)* as received; no erase and kill.
-cbreak	Make characters available to **read** only when newline is received.
-nl	Allow carriage return for newline, and output CR-LF for carriage return or newline.
nl	Accept only newline (linefeed) to end lines.
echo	Echo back every character typed.
-echo	Do not echo characters.
aresm	Turns on auto resume, which causes any input to resume the output, if it has been suspended.
-aresm	Turns off auto resume.
dtr	The 8560 monitors the DTR line (pin 20). The 8560 stops output when your terminal sets the DTR line low, and resumes output when your terminal sets the DTR line high. The command: `$ stty dtr <CR>` allows you to turn off your terminal without getting logged out by the 8560.
-dtr	The 8560 logs you out if the DTR line goes low.
xonxof	Turns on XON/XOF (CNTL-Q/CNTL-S) flagging on input.
-xonxof	Turns off XON/XOF (CNTL-Q/CNTL-S) flagging on input.

cts	The 8560 controls the CTS line (pin 5). The 8560 sets the CTS line low when it cannot accept any more input, and sets the CTS line high when it can accept input.
–cts	The 8560 keeps the CTS line high.
–tabs	Replace tabs by a proportionate number of spaces when printing.
tabs	Preserve tabs.
ek	Reset erase and kill characters back to normal ∧H and ∧U.
erase c	Set erase character to **c**. **C** can be of the form "∧X" which is interpreted as a CONTROL X.
kill c	Set kill character to **c**.
300 600 1200 2400 4800 9600	Set terminal baud rate to the number given, if possible.
IU	Specifies that an 8540 Integration Unit (IU) or 8550 Microcomputer Development Lab (MDL) is connected to the port. Support for an IU must be enabled before any commands can be issued to the IU or any control characters used by the HSI protocol can be sent to a terminal connected to the IU. It is recommended that IU support be turned on during system startup before any traffic occurs on the port.
–IU	Turns off IU/MDL support.

DIAGNOSTICS

Invalid option—Warning that configuration table has been changed (see **ioctl***(2)*).

SEE ALSO

ioctl*(2)*

SU(1)

NAME

su – substitute user id temporarily

SYNTAX

su [*userid*]

DESCRIPTION

Su demands the password of the specified *userid* , and if it is given, changes to that *userid* and invokes the Shell **sh** (1) without changing the current directory or the user environment (see **environ (5)**). The new user ID stays in force until the Shell exits.

If no *userid* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

SEE ALSO

sh(1)

[This page intentionally left blank.]

SYNC(1)

NAME

sync — update the super block

SYNTAX

sync

DESCRIPTION

Sync executes the **sync** system primitive. If the system is to be stopped, **sync** must be called to insure file system integrity. See **sync (2)** for details.

SEE ALSO

sync(2), update(8)

[This page intentionally left blank.]

TAIL(1)

NAME

tail – deliver the last part of a file

SYNTAX

tail [**-lbrc**][**–s** *number*][**–e** *number*][*file*]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no *file* is named, the standard input is used.

OPTIONS

–s *number*
: Print beginning at *number* units from the start of the file. Units defaults to lines.

–e *number*
: Print beginning at *number* units from the end of the file. Units defaults to lines.

–r Print lines in reversed order.

–l *Number* is in number of lines.

–b *Number* is in number of 512-byte blocks.

–c *Number* is in number of characters.

NOTES

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

The *–l, –b,* and *–c* options are mutually exclusive.

[This page intentionally left blank.]

TEE(1)

NAME

tee – pipe fitting

SYNTAX

tee [**–ia**] [*file*] ...

DESCRIPTION

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

OPTIONS

-i	causes interrupts to be ignored.
-a	causes the output to be appended to the *files* rather than overwriting them.

[This page intentionally left blank.]

TEST(1)

NAME

test – condition command

SYNTAX

test *expr*

DESCRIPTION

test evaluates the expression *expr* , and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. **test** returns a non zero exit if there are no arguments.

The following primitives are used to construct *expr* .

−r *file*	true if the file exists and is readable.
−w *file*	true if the file exists and is writable.
−f *file*	true if the file exists and is not a directory.
−d *file*	true if the file exists and is a directory.
−**s** *file*	true if the file exists and has a size greater than zero.
−t *[fildes]*	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
−**z** *s1*	true if the length of string *s1* is zero.
−n *s1*	true if the length of the string *s1* is nonzero.
s1 = s2	true if the strings *s1* and *s2* are equal.
s1 != s2	true if the strings *s1* and *s2* are not equal.
s1	true if *s1* is not the null string.
n1 −**eq** n2	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons −**ne** , −**gt** , −**ge** , −lt , or −**le** may be used in place of −**eq** .

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 *test* . Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO

sh(1), find(1)

[This page intentionally left blank.]

TIME(1)

NAME

time – time a command

SYNTAX

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.* Times are reported in seconds.

The times are printed on the diagnostic output stream.

NOTES

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

[This page intentionally left blank.]

TOUCH(1)

NAME

touch – update date last modified of a file

SYNTAX

touch [**−c**] **file**

DESCRIPTION

Touch attempts to set the modified date of each *file.* This is done by reading a character from the file and writing it back.

If a *file* does not exist, an attempt will be made to create it unless the **−c** option is specified.

[This page intentionally left blank.]

TR(1)

NAME

tr – translate characters

SYNTAX

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* . When *string2* is short it is padded to the length of *string1* by duplicating its last character. 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 01 through 0377 octal; **−d** deletes all input characters in *string1;* **−s** squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a-b* means a range of characters from *a* to *b* in increasing ASCII order. The character '\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A '\' followed by any other character stands for that character.

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 second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr −cs A−Za−z '\012' <file1 >file2
```

SEE ALSO

ed(1)

NOTES

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

[This page intentionally left blank.]

TRUE(1)

NAME

true, false – provide truth values

SYNTAX

true
false

DESCRIPTION

True does nothing, successfully. **False** does nothing, unsuccessfully. They are typically used in input to **sh (1)** such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1)

DIAGNOSTICS

True has exit status zero, **false** nonzero.

[This page intentionally left blank.]

TTY(1)

NAME
tty – get terminal name

SYNTAX
tty

DESCRIPTION
Tty prints the pathname of the user's terminal.

DIAGNOSTICS
'not a tty' if the standard input file is not a terminal.

[This page intentionally left blank.]

ULOAD(1)

NAME

uload – unformatted up/down load

SYNTAX

uload [**–ud**] [**-p** *cc...c*] [**-s** *nn*] *file*

DESCRIPTION

Uload provides the host side protocol for unformatted text transfers to or from **TNIX**. This is the protocol used for unformatted upload and download by the 8550 system. *File* contains (or will contain) the unformatted text. On uploading *file* will default to standard input. On downloading *file* will default to standard output.

OPTIONS

-d specifies download - download is the default

-u specifies upload. Options **–d** and **–u** are mutually exclusive.

-p *cc...c*
is an optional prompt, corresponding to the *P=* parameter on the TEKDOS COMM command. On uploads, it is used to prompt for each line and defaults to an empty string. On downloads, it is sent on a line by itself to indicate end of file and defaults to *'$$$$'* .

-s *nn* flag will slow down the transfer by padding each line with nulls, where *nn* specifies the number of nulls to send. A slowed transfer may be required by TEKDOS on the 8002.

For example, to send a 8002 file to be printed by **TNIX**, utter:

```
COMM            [on the 8002 to connect to TNIX]
{TNIX login}
uload -u | lpr {null}<{8002-file}    [{null} is ascii NUL]
                [transfer will now take place]
                ["*RIOT* EOJ" printed by 8002]
{ctrl-d}        [TNIX end-of-file indicator]
```

The file will now start printing on the **TNIX** lineprinter.

SEE ALSO

mload(l)

NOTES

Prompt sequences (**–p** option) that include dollar signs ($) must be in single-quotes because dollar signs have a special meaning to the shell.

Stty echo mode is disabled during the transfer.

[This page intentionally left blank.]

UNIQ(1)

NAME

uniq – report repeated lines in a file

SYNTAX

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 (1)**. 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(1), comm(1)

[This page intentionally left blank.]

WAIT(1)

NAME

wait – await completion of process

SYNTAX

wait

DESCRIPTION

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

Because the **wait (2)** system call must be executed in the parent process, the Shell itself executes **wait** , without creating a new process.

SEE ALSO

sh(1)

NOTES

Not all the processes of a 3 or more stage pipeline are children of the Shell, and thus can't be waited for.

[This page intentionally left blank.]

WC(1)

NAME

wc – word count

SYNTAX

wc [**−lwc**] [*name*]...

DESCRIPTION

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

If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters **l** , **w** , or **c** .

[This page intentionally left blank.]

WHO(1)

NAME

who – who is on the system

SYNTAX

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

DESCRIPTION

Who , without an argument, lists the login name, terminal name, and login time for each current **TNIX** 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, terminal 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, as in '**who am I**' (and also '**who are you**'), **who** tells who you are logged in as.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5)

[This page intentionally left blank.]

WRITE(1)

NAME

write – write to another user

SYNTAX

write *user* [*ttyname*]

DESCRIPTION

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

Message from yourname yourttyname...

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

If you want to write to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the **mesg(1)** command. At the outset writing is allowed. Certain commands, in particular **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(1), who(1), mail(1)

Section 2
System Calls

INTRODUCTION

System calls allow your program to obtain operating system services from the TNIX kernel via a C or assembly language interface. An executing process initiates a system call by issuing a trap instruction to the operating system. The trap instruction initiates a hardware trap, thereby switching the user process to kernel execution mode.

Most system calls return an error status to the calling process, allowing the process to monitor the status of the system call. Additionally, an error number is available to the calling process as the external variable *errno*. *Errno* can be passed to the subroutine *perror(3)*, which returns a terse error message for problem analysis to the standard error file. Consult the "Shell" section of your **8560 System Users Manual** for a more complete description of error message handling.

INTRO(2)

NAME

intro, errno – introduction to system calls and error numbers

SYNTAX

#include <errno.h>

DESCRIPTION

Most of the system calls listed below have an error return. An error condition is indicated by an otherwise impossible returned value. Almost always this is -1; the individual sections specify the details. An error number is also made available in the external variable **errno** . **Errno** is not cleared on successful calls, so it should be tested only after an error has occurred.

There is a table of messages associated with **errno,** and a routine for printing an explanatory message; See **perror (3)**. The possible error numbers are not recited with each writeup in section 2, since many errors are possible for most of the calls. Here is a list of the error numbers, their names as defined in <errno.h>, and the messages available using **perror** .

0 *Error 0*

Unused.

1 *EPERM Not owner*

Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 *ENOENT No such file or directory*

This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 *ESRCH No such process*

The process whose number was given to **signal** and **ptrace** does not exist, or is already dead.

4 *EINTR Interrupted system call*

An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 *EIO I/O error*

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

6 *ENXIO No such device or address*

I/O on a special file refers to a subdevice that does not exist, or beyond the limits of the device.

7 *E2BIG Arg list too long*

An argument list longer than 5120 bytes is presented to **exec** .

8 *ENOEXEC Exec format error*

A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number, see **a.out (5).**

9 *EBADF Bad file number*

Either a file descriptor refers to no open file, or a read (resp. write) request is made to a file that is open only for writing (resp. reading).

10 *ECHILD No children*

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

11 *EAGAIN No more processes*

In a **fork,** the system's process table is full or the user is not allowed to create any more processes.

12 *ENOMEM Not enough core*

During an **exec** or **break,** a program asks for more core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers.

13 *EACCES Permission denied*

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

14 *EFAULT Bad address*

The system encountered a hardware fault in attempting to access the arguments of a system call.

15 *ENOTBLK Block device required*

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

16 *EBUSY Mount device busy*

An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment).

17 *EEXIST File exists*

An existing file was mentioned in an inappropriate context, e.g. **link** .

18 *EXDEV Cross-device link*

A link to a file on another device was attempted.

19 *ENODEV No such device*

An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20 *ENOTDIR Not a directory*

A non-directory was specified where a directory is required, for example in a path name or as an argument to **chdir** .

21 *EISDIR Is a directory*

An attempt to write on a directory.

22 *EINVAL Invalid argument*

Some invalid argument: dismounting a non-mounted device, mentioning an unknown signal in **signal**, reading or writing a file for which **seek** has generated a negative pointer. Also set by math functions.

23 *ENFILE File table overflow*

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

24 *EMFILE Too many open files*

The current process has opened too many files. Some must be closed before another can be opened.

25 *ENOTTY Not a typewriter*

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

26 *ETXTBSY Text file busy*

An attempt to execute a pure-procedure program that is currently open for writing (or reading!). Also an attempt to open for writing a pure-procedure program that is being executed.

27 *EFBIG File too large*

The size of a file exceeded the maximum (about 1.0E9 bytes).

28 *ENOSPC No space left on device*

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

29 *ESPIPE Illegal seek*

An **lseek** was issued to a pipe. This error should also be issued for other non-seekable devices.

30 *EROFS Read-only file system*

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

31 *EMLINK Too many links*

An attempt to make more than 32767 links to a file.

32 *EPIPE Broken pipe*

A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 *EDOM Math argument*

The argument of a function in the math package (3M) is out of the domain of the function.

34 *ERANGE Result too large*

The value of a function in the math package (3M) is unrepresentable within machine precision.

SEE ALSO

perror(3)

ASSEMBLER

as /usr/include/sys.s *file...*

The PDP11 assembly language interface is given for each system call. The assembler symbols are defined in '/usr/include/sys.s'.

Return values appear in registers r0 and r1; it is unwise to count on these registers being preserved when no value is expected. An erroneous call is always indicated by turning on the c-bit of the condition codes. The error number is returned in r0. The presence of an error is most easily tested by the instructions *bes* and *bec* ('branch on error set (or clear)'). These are synonyms for the *bcs* and *bcc* instructions.

ACCESS(2)

NAME
access – determine accessibility of file

SYNTAX
access(name, mode)
char *name;

DESCRIPTION
Access checks the given file *name* for accessibility according to *mode* , which is 4 (read), 2 (write) or 1 (execute) or a combination thereof. Specifying mode 0 tests whether the directories leading to the file can be searched and the file exists.

An appropriate error indication is returned if *name* cannot be found or if any of the desired access modes would not be granted. On disallowed accesses −1 is returned and the error code is in *errno* , 0 is returned from successful tests.

The user and group IDs with respect to which permission is checked are the real UID and GID of the process, so this call is useful to set-UID programs.

Notice that it is only access bits that are checked. A directory may be announced as writable by *access* , but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but *exec* will fail unless it is in proper format.

SEE ALSO
stat(2)

ASSEMBLER
(access = 33.)
sys access; name; mode

ALARM(2)

NAME

alarm – schedule signal after specified time

SYNTAX

alarm(seconds)
unsigned seconds;

DESCRIPTION

Alarm causes signal SIGALRM, see *signal (2)* , to be sent to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is cancelled. Because the clock has a 1-second resolution, the signal may occur up to one second early; because of scheduling delays, resumption of execution when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 65535 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

pause(2), signal(2)

ASSEMBLER

(alarm = 27.)
(seconds in r0)
sys alarm
(previous amount in r0)

BRK(2)

NAME

brk, sbrk, break – change memory allocation

SYNTAX

char *brk(addr)

char *sbrk(incr)

DESCRIPTION

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

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

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

SEE ALSO

exec(2)

DIAGNOSTICS

Zero is returned if the break could be set; −1 if the program requests more memory than the system limit or if too many segmentation registers would be required to implement the break.

Sbrk returns a pointer to the start of the new area if successful, and 0 if not.

NOTES

Setting the break in the range 0177701 to 0177777 is the same as setting it to zero.

ASSEMBLER

(break = 17.)
sys break; addr

Break performs the function of *brk* . The name of the routine differs from that in C for historical reasons.

CHDIR(2)

NAME

chdir, chroot – change default directory

SYNTAX

chdir(dirname)
char *dirname;

chroot(dirname)
char *dirname;

DESCRIPTION

Dirname is the address of the pathname of a directory, terminated by a null byte. *Chdir* causes this directory to become the current working directory, the starting point for path names not beginning with '/'.

Chroot sets the root directory, the starting point for path names beginning with '/'. The call is restricted to the super-user.

Chdir and Chroot only change the working and root directories for the current process and any of its children.

SEE ALSO

cd(1)

DIAGNOSTICS

Zero is returned if the directory is changed; −1 is returned if the given name is not that of a directory or is not searchable.

ASSEMBLER

(chdir = 12.)
sys chdir; dirname

(chroot = 61.)
sys chroot; dirname

CHMOD(2)

NAME

chmod – change mode of file

SYNTAX

chmod(name, mode)
char *name;

DESCRIPTION

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

04000	set user ID on execution
02000	set group ID on execution
01000	save text image after execution
00400	read by owner
00200	write by owner
00100	execute (search on directory) by owner
00070	read, write, execute (search) by group
00007	read, write, execute (search) by others

If an executable file is set up for sharing then mode 01000 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.

Only the owner of a file (or the super-user) may change the mode. Only the super-user can set the 1000 mode.

SEE ALSO

chmod(1)

DIAGNOSTICS

Zero is returned if the mode is changed; −1 is returned if *name* cannot be found or if current user is neither the owner of the file nor the super-user.

ASSEMBLER

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

CHOWN(2)

NAME

chown – change owner and group of a file

SYNTAX

chown(name, owner, group)
char *name;

DESCRIPTION

The file whose name is given by the null-terminated string pointed to by *name* has its *owner* and *group* changed as specified. Only the super-user may execute this call, because if users were able to give files away, they could defeat the (nonexistent) file-space accounting procedures.

SEE ALSO

chown(1), passwd(5)

DIAGNOSTICS

Zero is returned if the owner is changed; −1 is returned on illegal owner changes.

ASSEMBLER

(chown = 16.)
sys chown; name; owner; group

CLOSE(2)

NAME

close – close a file

SYNTAX

close(fildes)

DESCRIPTION

Given a file descriptor such as returned from an *open* , *creat* , *dup* or *pipe (2)* call, *close* closes the associated file. A close of all files is automatic on *exit* , but since there is a limit on the number of open files per process, *close* is necessary for programs which deal with many files.

Files are closed upon termination of a process, and device file descriptors may be closed automatically by a sucessful *exec(2)* , following an *ioctl(2)* call.

SEE ALSO

creat(2), dup(2), open(2), pipe(2), exec(2), ioctl(2)

DIAGNOSTICS

Zero is returned if a file is closed; −1 is returned for an unknown file descriptor.

ASSEMBLER

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

CREAT(2)

NAME

creat – create a new file

SYNTAX

creat(name, mode)
char *name;

DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file called *name* , given as the address of a null-terminated string. If the file did not exist, it is given mode *mode* , as modified by the process's mode mask (see *umask (2)*). Also see *chmod (2)* for the construction of the *mode* argument.

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

The file is also opened for writing, and its file descriptor is returned.

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

SEE ALSO

write(2), close(2), chmod(2), umask (2)

DIAGNOSTICS

The value −1 is returned if: a needed directory is not searchable; the file does not exist and the directory in which it is to be created is not writable; the file does exist and is unwritable; the file is a directory; there are already too many files open.

ASSEMBLER

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

DUP(2)

NAME

dup, dup2 – duplicate an open file descriptor

SYNTAX

```
dup(fildes)
int fildes;

dup2(fildes, fildes2)
int fildes, fildes2;
```

DESCRIPTION

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

In the second form of the call, *fildes* is a file descriptor referring to an open file, and *fildes2* is a non-negative integer less than the maximum value allowed for file descriptors. *Dup2* causes *fildes2* to refer to the same file as *fildes* . If *fildes2* already referred to an open file, it is closed first.

SEE ALSO

creat(2), open(2), close(2), pipe(2)

DIAGNOSTICS

The value −1 is returned if: the given file descriptor is invalid; there are already too many open files.

ASSEMBLER

```
(dup = 41.)
(file descriptor in r0)
(new file descriptor in r1)
sys dup
(file descriptor in r0)
```

The *dup2* entry is implemented by adding 0100 to *fildes* .

EXEC(2)

NAME

execl, execv, execle, execve, execlp, execvp, exec, exece, environ – execute a file

SYNTAX

execl(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

execv(name, argv)
char *name, *argv[ ];

execlp(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

execvp(name, argv)
char *name, *argv[ ];

execle(name, arg0, arg1, ..., argn, 0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[ ];

execve(name, argv, envp);
char *name, *argv[ ], *envp[ ];

extern char **environ;

DESCRIPTION

Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

Files remain open across *exec* unless explicit arrangement has been made; see *ioctl (2)* . Ignored signals remain ignored across these calls, but signals that are caught (see *signal (2)*) are reset to their default values.

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

The *name* argument is a pointer to the null-terminated name of the file to be executed. The pointers *arg [0]* , *arg [1]* ... address null-terminated strings. Conventionally *arg [0]* is the name of the file.

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

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

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

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

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

Argv is directly usable in another *execv* because *argv [argc]* is 0.

Envp is a pointer to an array of strings that constitute the *environment* of the process. Each string consists of a name, an "=", and a null-terminated value. The array of pointers is terminated by a null pointer. The shell *sh (1)* passes an environment entry for each global shell variable defined when the program is called. See *environ (5)* for some conventionally used names. The C run-time start-off routine places a copy of *envp* in the global cell *environ* , which is used by *execv and execl* to pass the environment to any subprograms executed by the current program. The *exec* routines use lower-level routines as follows to pass an environment explicitly:
execle(file, arg0, arg1, . . . , argn, 0, environ);
execve(file, argv, environ);

Execlp and *execvp* are called with the same arguments as *execl* and *execv* , but duplicate the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

FILES

/bin/sh shell, invoked if command file found by *execlp* or *execvp*

SEE ALSO

fork(2), environ(5)

DIAGNOSTICS

If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see *a.out (5)*), if maximum memory is exceeded, or if the arguments require too much space, a return constitutes the diagnostic; the return value is −1. An exception to this is with execvp and execlp which will execute a shell command file (which does not start with a valid magic number). Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

NOTES

If *execvp* is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of *argv[0]* and *argv[−1]* will be modified before return.

ASSEMBLER

(exec = 11.)
sys exec; name; argv

(exece = 59.)
sys exece; name; argv; envp

When the called file starts execution on the 8560, the stack pointer points to a word containing the number of arguments. Just above this number is a list of pointers to the argument strings, followed by a null pointer, followed by the pointers to the environment strings and then another null pointer. The strings themselves follow; a 0 word is left at the very top of memory.

Addresses increase downward in the following diagram.

```
sp->            nargs
       arg0
       ...
       argn
       0
       env0
       ...
       envm
       0
arg0:  <arg0\0>
       ...
env0:  <env0\0>
       0
```

This arrangement happens to conform well to C calling conventions.

EXIT(2)

NAME

exit – terminate process

SYNTAX

exit(status)
int status;

_exit(status)
int status;

DESCRIPTION

Exit is the normal means of terminating a process. *Exit* closes all the process's files and notifies the parent process if it is executing a *wait*. The low-order 8 bits of *status* are available to the parent process.

This call can never return.

The C function *exit* may cause cleanup actions before the final 'sys exit'. The C function *_exit* circumvents all cleanup.

SEE ALSO

wait(2)

ASSEMBLER

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

FORK(2)

NAME

fork – spawn new process

SYNTAX

fork( )

DESCRIPTION

Fork is the only way new processes are created. The new process's memory image is a copy of that of the caller of *fork* . The only distinction is the fact that the value returned in the old (parent) process contains the process ID of the new (child) process, while the value returned in the child is 0. Process ID's range from 1 to 30,000. This process ID is used by *wait (2)* .

Files open before the fork are shared, and have a common read-write pointer. In particular, this is the way that standard input and output files are passed and also how pipes are set up.

SEE ALSO

wait(2), exec(2)

DIAGNOSTICS

Returns −1 and fails to create a process if: there is inadequate swap space, the user is not super-user and has too many processes, or the system's process table is full. Only the super-user can take the last process-table slot.

ASSEMBLER

(fork = 2.)
sys fork
(new process return)
(old process return, new process ID in r0)

The return locations in the old and new process differ by one word. The C-bit is set in the old process if a new process could not be created.

GETPID(2)

NAME

getpid – get process identification

SYNTAX

getpid( )

DESCRIPTION

Getpid returns the process ID of the current process. Most often it is used to generate uniquely-named temporary files.

ASSEMBLER

(getpid = 20.)
sys getpid
(pid in r0)

GETUID(2)

NAME

getuid, getgid, geteuid, getegid – get user and group identity

SYNTAX

getuid( )

geteuid( )

getgid( )

getegid( )

DESCRIPTION

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

Getgid returns the real group ID, *getegid* the effective group ID.

SEE ALSO

setuid(2)

ASSEMBLER

(getuid = 24.)
sys getuid
(real user ID in r0, effective user ID in r1)

(getgid = 47.)
sys getgid
(real group ID in r0, effective group ID in r1)

INDIR(2)

NAME

indir – indirect system call

ASSEMBLER

(indir = 0.)
sys indir; call

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

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

If *indir* is executed indirectly, it is a no-op. If the instruction at the indirect location is not a system call, *indir* returns error code EINVAL; see *intro (2).*

IOCTL(2)

NAME

ioctl, stty, gtty – control device

SYNTAX

#include <sgtty.h>

ioctl(fildes, request, argp)
struct sgttyb *argp;

stty(fildes, argp)
struct sgttyb *argp;

gtty(fildes, argp)
struct sgttyb *argp;

DESCRIPTION

Ioctl performs a variety of functions on character special files (devices). The write-ups of various devices in section 2.6.8 discuss how *ioctl* applies to them.

For certain status setting and status inquiries about terminal devices, the functions *stty* and *gtty* are equivalent to
ioctl(fildes, TIOCSETP, argp)
ioctl(fildes, TIOCGETP, argp)

respectively; see **tty(4)** .

The following two calls apply to any open file:

ioctl(fildes, FIOCLEX, NULL);
ioctl(fildes, FIONCLEX, NULL);

The first causes the file to be closed automatically during a successful *exec* operation; the second reverses the effect of the first.

SEE ALSO

stty(1), exec(2), fd(4), hd(4), tty(4)

DIAGNOSTICS

Zero is returned if the call was successful; -1 if the file descriptor does not refer to the kind of file for which it was intended, or if the data structure description is invalid.

NOTES

Strictly speaking, since *ioctl* may be extended in different ways to devices with different properties, *argp* should have an open-ended declaration like

union { struct sgttyb ... ; ... } *argp;

The important thing is that the size is fixed by 'struct sgttyb'.

ASSEMBLER

(ioctl = 54.)
sys ioctl; fildes; request; argp

(stty = 31.)
(file descriptor in r0)
stty; argp

(gtty = 32.)
(file descriptor in r0)
sys gtty; argp

KILL(2)

NAME

kill – send signal to a process

SYNTAX

kill(pid, sig);

DESCRIPTION

Kill sends the signal *sig* to the process specified by the process number *pid.* See *signal (2)* for a list of signals.

The sending and receiving processes must have the same effective user ID, otherwise this call is restricted to the super-user.

If the process number is 0, the signal is sent to all other processes in the group of processes originating from the same terminal as the sender.

If the process number is −1, and the user is the super-user, the signal is broadcast universally except to processes 0 and 1, the scheduler and initialization processes, see *init (8)* .

Processes may send signals to themselves.

SEE ALSO

signal(2), kill(1)

DIAGNOSTICS

Zero is returned if the process is killed; −1 is returned if the process does not have the same effective user ID and the user is not super-user, or if the process does not exist.

ASSEMBLER

(kill = 37.)
(process number in r0)
sys kill; sig

LINK(2)

NAME

link – link to a file

SYNTAX

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

DESCRIPTION

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

SEE ALSO

ln(1), unlink(2)

DIAGNOSTICS

Zero is returned when a link is made; −1 is returned when *name1* cannot be found; when *name2* already exists; when the directory of *name2* cannot be written; when an attempt is made to link to a directory by a user other than the super-user; when an attempt is made to link to a file on another file system; when a file has too many links.

ASSEMBLER

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

LOCK(2)

NAME

lock – lock a process in primary memory

SYNTAX

lock(flag)

DESCRIPTION

If the *flag* argument is non-zero, the process executing this call will not be swapped except if it is required to grow. If the argument is zero, the process is *unlocked.* This call may only be executed by the super-user.

NOTES

Locked processes interfere with the compaction of primary memory and can cause deadlock. This system call is not considered a permanent part of the system.

ASSEMBLER

(lock = 53.)
sys lock; flag

LSEEK(2)

NAME

lseek, tell – move read/write pointer

SYNTAX

long lseek(fildes, offset, whence)
long offset;

long tell(fildes)

DESCRIPTION

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

If *whence* is 0, the pointer is set to *offset* bytes.

If *whence* is 1, the pointer is set to its current location plus *offset* .

If *whence* is 2, the pointer is set to the size of the file plus *offset* .

The returned value is the resulting pointer location.

The function *tell (fildes)* is identical to *lseek (fildes, 0L, 1)* .

(The 'L' in '0L' specifies a long value in C.)

Seeking far beyond the end of a file, then writing, creates a gap or 'hole', which occupies no physical space and reads as zeros.

SEE ALSO

open(2), creat(2)

DIAGNOSTICS

−1 is returned for an undefined file descriptor, seek on a pipe, or seek to a position before the beginning of file.

NOTES

Lseek is a no-op on character special files.

ASSEMBLER

(lseek = 19.)
(file descriptor in r0)
sys lseek; offset1; offset2; whence

Offset1 and *offset2* are the high and low words of *offset* ; r0 and r1 contain the pointer upon return.

MKNOD(2)

NAME

mknod – make a directory or a special file

SYNTAX

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

DESCRIPTION

Mknod creates a new file whose name is the null-terminated string pointed to by *name* . The mode of the new file (including directory and special file bits) is initialized from *mode* . (The protection part of the mode is modified by the process's mode mask; see *umask(2)*). The first block pointer of the i-node is initialized from *addr* . For ordinary files and directories *addr* is normally zero. In the case of a special file, *addr* specifies which special file.

Mknod may be invoked only by the super-user.

SEE ALSO

mkdir(1), mknod(1), filsys(5)

DIAGNOSTICS

Zero is returned if the file has been made; −1 if the file already exists or if the user is not the super-user.

ASSEMBLER

(mknod = 14.)
sys mknod; name; mode; addr

MOUNT(2)

NAME

mount, umount—mount or remove file system

SYNTAX

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

DESCRIPTION

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

Name must exist already. *Name* must be a directory (unless the root of the mounted file system is not a directory). Its old contents are inaccessible while the file system is mounted.

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

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

SEE ALSO

mount*(8)*

DIAGNOSTICS

Mount returns 0 if the action occurred; –1 if *special* is inaccessible or not an appropriate file, if *name* does not exist, if *special* is already mounted, if *name* is in use, or if there already are too many file systems mounted.

Umount returns 0 if the action occurred; –1 if if the special file is inaccessible or does not have a mounted file system, or if there are active files in the mounted file system.

ASSEMBLER

(mount = 21.)
sys mount; special; name; rwflag
(umount = 22.)
sys umount; special

NICE(2)

NAME
nice – set program priority

SYNTAX
nice(incr)

DESCRIPTION
The scheduling priority of the process is augmented by *incr* . Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without inconvienencing other users.

Negative increments are ignored except on behalf of the super-user. The priority is limited to the range −20 (most urgent) to 20 (least).

The priority of a process is passed to a child process by *fork (2)* . For a privileged process to return to normal priority from an unknown state, *nice* should be called successively with arguments −40 (goes to priority −20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

SEE ALSO
nice(1)

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

OPEN(2)

NAME

open – open for reading or writing

SYNTAX

open(name, mode)
char *name;

DESCRIPTION

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

The file is positioned at the beginning (byte 0). The returned file descriptor must be used for subsequent calls for other input-output functions on the file.

SEE ALSO

creat(2), read(2), write(2), dup(2), close(2)

DIAGNOSTICS

The value −1 is returned if the file does not exist, if one of the necessary directories does not exist or is unreadable, if the file is not readable (resp. writable), or if too many files are open, or if the mode is invalid.

ASSEMBLER

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

PAUSE(2)

NAME

pause – stop until signal

SYNTAX

pause( )

DESCRIPTION

Pause never returns normally. It is used to give up control while waiting for a signal from *kill (2)* or *alarm (2)* .

SEE ALSO

kill(1), kill(2), alarm(2), signal(2)

ASSEMBLER

(pause = 29.)
sys pause

PHYS(2)

NAME

phys – allow a process to access physical addresses

SYNTAX

phys(segreg, size, physadr)

DESCRIPTION

The argument *segreg* specifies a process virtual (data-space) address range of 8K bytes starting at virtual address *segreg* ×*8K* bytes. This address range is mapped into physical address *physadr* ×*64* bytes. Only the first *size* ×*64* bytes of this mapping is addressable. If *size* is zero, any previous mapping of this virtual address range is nullified. For example, the call

```
phys(6, 1, 0177775);
```

will map virtual addresses 0160000-0160077 into physical addresses 017777500-017777577. In particular, virtual address 0160060 is the PDP-11 console located at physical address 017777560.

This call may only be executed by the super-user.

SEE ALSO

PDP-11 segmentation hardware

DIAGNOSTICS

The function value zero is returned if the physical mapping is in effect. The value −1 is returned if not super-user, if *segreg* is not in the range 0-7, if *size* is not in the range 0-127, or if the specified *segreg* is already used for other than a previous call to *phys* .

NOTES

This system call is obviously very machine dependent and very dangerous. This system call is not considered a permanent part of the system.

ASSEMBLER

(phys = 52.)
sys phys; segreg; size; physadr

PIPE(2)

NAME

pipe – create an interprocess channel

SYNTAX

pipe(fildes)
int fildes[2];

DESCRIPTION

The *pipe* system call creates an I/O mechanism called a pipe. The file descriptors returned can be used in read and write operations. When the pipe is written using the descriptor *fildes [1]* up to 4096 bytes of data are buffered before the writing process is suspended. A read using the descriptor *fildes [0]* will pick up the data. Writes with a count of 4096 bytes or less are atomic (i.e. indivisible); no other process can intersperse data.

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

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

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) returns an end-of-file.

SEE ALSO

sh(1), read(2), write(2), fork(2)

DIAGNOSTICS

The function value zero is returned if the pipe was created; −1 if too many files are already open. A signal (see *signal(2)*) is generated if a write on a pipe with only one end is attempted.

NOTES

Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.

ASSEMBLER

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

PROFIL(2)

NAME

profil – execution time profile

SYNTAX

profil(buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;

DESCRIPTION

Buff points to an area of memory whose length (in bytes) is given by *bufsiz*. After this call, the user's program counter (pc) is examined each clock tick (60th or 50th second depending on local line frequency); *offset* is subtracted from it, and the result multiplied by *scale* . If the resulting number corresponds to a word inside *buff* , that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777(8) gives a 1-1 mapping of pc's to words in *buff* ; 077777(8) maps each pair of instruction words together. 02(8) maps all instructions onto the beginning of *buff* (producing a non-interrupting memory clock).

Profiling is turned off by giving a *scale* of 0 or 1. It is rendered ineffective by giving a *bufsiz* of 0. Profiling is turned off when an *exec* is executed, but remains on in child and parent both after a *fork* . Profiling will be turned off if an update in *buff* would cause a memory fault.

SEE ALSO

prof(1)

ASSEMBLER

(profil = 44.)
sys profil; buff; bufsiz; offset; scale

PTRACE(2)

NAME

ptrace – process trace

SYNTAX

#include <signal.h>

ptrace(request, pid, addr, data)
int *addr;

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process, and examine and change its memory image. Its primary use is for the implementation of breakpoint debugging. There are four arguments whose interpretation depends on a *request* argument. Generally, *pid* is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like 'illegal instruction' or externally generated like 'interrupt.' See *signal (2)* for the list. Then the traced process enters a stopped state and its parent is notified via *wait (2)* . When the child is in the stopped state, its memory image can be examined and modified using *ptrace* . If desired, another *ptrace* request can then cause the child either to terminate or to continue, possibly ignoring the signal.

The value of the *request* argument determines the precise action of the call:

0 This request and (under some circumstances) request 10 are the only ones used by the child process. It declares that the process is to be traced by its parent. All the other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

1 The word in the child process's address space at *addr* is returned. *Addr* must be even. The child must be stopped. The input *data* is ignored.

3 The word of the system's per-process data area corresponding to *addr* is returned. *Addr* must be even and less than 512. This space contains the registers and other information about the process; its layout corresponds to the *user* structure in the system.

4 The given *data* is written at the word in the process's address space corresponding to *addr* , which must be even. No useful value is returned. Attempts to write in pure text fail if another process is executing the same file.

6 The process's system data is written, as it is read with request 3. Only a few locations can be written in this way: the general registers, the floating point status and registers, and certain bits of the processor status word.

7 The *data* argument is taken as a signal number and the child's execution continues at location *addr* as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal that caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop. If *addr* is (int *)1 then execution continues from where it stopped.

8 The traced process terminates.

9 Execution continues as in request 7; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is SIGTRAP. On the PDP-11 the T-bit is used and just one instruction is executed. This is part of the mechanism for implementing breakpoints.

10 If *data* is non-zero, then subsequent system calls made below *addr* are declared illegal and generate a SIGSYS. An *addr* of 0177777 (on the PDP-11) effectively disables all system calls. If either *data* or *addr* is zero, then system calls are again handled normally. This allows system calls to be traced, and is used to simulate *trap* instructions when emulating DEC environments. Unlike the other requests, *pid* can refer to the calling process or a traced child. In some emulation schemes, this is the only request used and there is in fact no child.

As indicated, requests 1 through 9 can be used only when the subject process has stopped. The *wait* call is used to determine when a process stops; in such a case the 'termination' status returned by *wait* has the value 0177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, *ptrace* inhibits the set-user-id facility on subsequent *exec (2)* calls. If a traced process calls *exec* , it will stop before executing the first instruction of the new image showing signal SIGTRAP.

SEE ALSO

wait(2), signal(2), adb(1)

DIAGNOSTICS

The value −1 is returned if *request* is invalid, *pid* is not a traceable process, *addr* is out of bounds, or *data* specifies an illegal signal number.

NOTES

The request 0 call should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use 'illegal instruction' signals at a very high rate) could be efficiently debugged.

ASSEMBLER

(ptrace = 26.)
(data in r0)
sys ptrace; pid; addr; request
(value in r0)

READ(2)

NAME

read – read from file

SYNTAX

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

DESCRIPTION

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

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

SEE ALSO

open(2), creat(2), dup(2), pipe(2)

DIAGNOSTICS

As mentioned, 0 is returned when the end of the file has been reached. If the read was otherwise unsuccessful the return value is −1. Many conditions can generate an error: physical I/O errors, bad buffer address, preposterous *nbytes* , file descriptor not that of an input file.

ASSEMBLER

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

SETUID(2)

NAME

setuid, setgid – set user and group ID

SYNTAX

setuid(uid)

setgid(gid)

DESCRIPTION

The user ID (group ID) of the current process is set to the argument. Both the effective and the real ID are set. These calls are only permitted to the super-user or if the argument is the real ID.

SEE ALSO

getuid(2)

DIAGNOSTICS

Zero is returned if the user (group) ID is set; −1 is returned otherwise.

ASSEMBLER

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

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

SIGNAL(2)

NAME

signal – catch or ignore signals

SYNTAX

#include <signal.h>

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

DESCRIPTION

A signal is generated by some abnormal event, initiated either by user at a typewriter (quit, interrupt), by a program error (bus error, etc.), or by request of another program (kill). Normally all signals cause termination of the receiving process, but a *signal* call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals with names as in the include file.

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction (not reset when caught)
SIGTRAP	5*	trace trap (not reset when caught)
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe or link with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
	16	unassigned

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

If *func* is SIG_DFL, the default action for signal *sig* is reinstated; this default is termination, sometimes with a memory image. If *func* is SIG_IGN the signal is ignored. Otherwise when the signal occurs *func* will be called with the signal number as argument. A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal is reset to SIG_DFL after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another *signal* call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a *read* or *write (2)* on a slow device (like a typewriter; but not a file); and during *pause* or *wait (2)* . When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call.

The value of *signal* is the previous (or initial) value of *func* for the particular signal.

After a *fork (2)* the child inherits all signals. *Exec (2)* resets all caught signals to default action.

SEE ALSO
kill(1), kill(2), ptrace(2)

DIAGNOSTICS
The value (int)−1 is returned if the given signal is out of range.

NOTES
If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

The type specification of the routine and its *func* argument are problematical.

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

If *label* is 0, default action is reinstated. If *label* is odd, the signal is ignored. Any other even *label* specifies an address in the process where an interrupt is simulated. An RTI or RTT instruction will return from the interrupt.

STAT(2)

NAME

stat, fstat – get file status

SYNTAX

#include <sys/types.h>
#include <sys/stat.h>

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

fstat(fildes, buf)
struct stat *buf;

DESCRIPTION

Stat obtains detailed information about a named file. *Fstat* obtains the same information about an open file known by the file descriptor from a successful *open* , *creat* , *dup* or *pipe (2)* call.

Name points to a null-terminated string naming a file; *buf* is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be searchable. The layout of the structure pointed to by buf as defined in *<stat.h>* is given below. *St_mode* is encoded according to the '#define' statements.

```
struct stat
{
        dev_t   st_dev;         /* device where the inode lives  */
        ino_t   st_ino;         /* inode number*/
        unsigned short st_mode;/* mode*/
        short   st_nlink;       /* link count     */
        short   st_uid;         /* set user id    */
        short   st_gid;         /* set group id   */
        dev_t   st_rdev; /* device where data lives        */
        off_t   st_size; /* file size        */
        time_t  st_atime;       /* access time  */
        time_t  st_mtime;       /* modification time      */
        time_t  st_ctime;       /* creation time */
};
```

```
#define S_IFMT      0170000         /* type of file */
#define S_IFDIR     0040000         /* direct */
#define S_IFCHR          0020000         /* character special */
#define S_IFBLK          0060000         /* block special */
#define S_IFREG          0100000         /* regular */
#define S_ISUID     0004000         /* set user id on execution */
#define S_ISGID          0002000         /* set group id on execution */
#define S_ISVTX          0001000         /* save swapped text even */
                              /* after use      */
#define S_IREAD          0000400         /* read permission, owner */
#define S_IWRITE    0000200         /* write permission, owner */
#define S_IEXEC          0000100         /* execute/search permission, */
                              /* owner          */
```

The mode bits 0000070 and 0000007 encode group and others permissions (see *chmod (2)*). The defined types, *ino_t* , *off_t* , *time_t* , name various width integer values; *dev_t* encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h> (see *types (5)* .

When *fildes* is associated with a pipe, *fstat* reports an ordinary file with restricted permissions. The size is the number of bytes queued in the pipe.

st_atime is the time that the file was last read. For reasons of efficiency, it is not set when a directory is searched, although this would be more logical. *st_mtime* is the time the file was last written or created. It is not set by changes of owner, group, link count, or mode. *st_ctime* is set both by writing and changing the i-node.

SEE ALSO

ls(1), filsys(5)

DIAGNOSTICS

Zero is returned if a status is available; −1 if the file cannot be found.

ASSEMBLER

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

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

STIME(2)

NAME

stime – set time

SYNTAX

```
#include <sys/types.h>
#include <sys/timeb.h>
stime(tp)
struct timeb *tp;
```

DESCRIPTION

Stime sets the system's idea of the time and date. Time, pointed to by *tp* , is a structure defined by *<sys/timeb.h>* :

```
struct timeb {
        time_t  time;           /* seconds past the epoch    */
        unsigned short millitm;       /* up to 1000 milliseconds of */
                                /* more precise interval     */
        short   timezone;       /* minutes of time westward  */
                                /* from Greenwich            */
        short   dstflg;         /* Daylight savings time flag */
}
```

The epoch is 0000 GMT Jan 1, 1970. Dstflg is a flag which is non-zero if daylight savings time is locally observed at some times during the year. Only the super-user may use this call.

SEE ALSO

date(1), time(2)

DIAGNOSTICS

Zero is returned if the time was set; −1 if user is not the super-user.

ASSEMBLER

(stime = 25.)
sys ftime; tp

SYNC(2)

NAME

sync — update super-block

SYNTAX

sync( )

DESCRIPTION

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

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

The writing, although scheduled, is not necessarily complete upon return from *sync*.

SEE ALSO

sync(1), update(8)

ASSEMBLER

(sync = 36.)
sys sync

TIME(2)

NAME

time, ftime – get date and time

SYNTAX

long time(0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;

DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If *tloc* is nonnull, the return value is also stored in the place to which *tloc* points.

The *ftime* entry fills in a structure pointed to by its argument, as defined by *<sys/timeb.h>* :

```
struct timeb {
        time_t    ttime;
        unsigned short millitm;
        short     timezone;
        short     dstflag;
};
struct timebtz{
        struct timeb t;/* The old timeb structure */
        char      tzname[4];/* time zone name */
        char      tzdayname[4];/* daylight timezone name */
};
```

The timeb structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local timezone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time is locally observed at times during the year. The epoch is 0000 GMT Jan. 1, 1970.

If the pointer argument to *ftime* is an odd address the timebtz structure is returned. It is returned to where the odd address points minus one. The timebtz structure contains the timeb structure plus the standard timezone string, and daylight timezone string.

SEE ALSO

date(1), stime(2)

ASSEMBLER

(ftime = 35.)
sys ftime; bufptr

(time = 13.; obsolete call)
sys time
(time since 1970 in r0-r1)

TIMES(2)

NAME

times – get process times

SYNTAX

times(buffer)
struct tbuffer *buffer;

DESCRIPTION

Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ= 60 or 50 depending on local line frequency.

After the call, the buffer will appear as follows:

```
struct tbuffer {
        long    proc_user_time;
        long    proc_system_time;
        long    child_user_time;
        long    child_system_time;
};
```

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

SEE ALSO

time(1), time(2)

ASSEMBLER

(times = 43.)
sys times; buffer

UMASK(2)

NAME

umask – set file creation mode mask

SYNTAX

umask(complmode)

DESCRIPTION

Umask sets a mask used whenever a file is created by *creat (2)* or *mknod (2)* : the actual mode (see *chmod (2)*) of the newly-created file is the logical **and** of the mode given to chmod, creat, or mknod, and the complement of the argument given to umask. Only the low-order 9 bits of the mask (the protection bits) participate. In other words, the mask shows the bits to be turned off when files are created.

The previous value of the mask is returned by the call. The value is initially 0 (no restrictions). The mask is inherited by child processes.

SEE ALSO

creat(2), mknod(2), chmod(2)

ASSEMBLER

(umask = 60.)
sys umask; complmode

UNLINK(2)

NAME

unlink – remove directory entry

SYNTAX

unlink(name)
char *name;

DESCRIPTION

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

SEE ALSO

rm(1), link(2)

DIAGNOSTICS

Zero is normally returned; −1 indicates that the file does not exist, that its directory cannot be written, or that the file contains pure procedure text that is currently in use. Write permission is not required on the file itself. It is also illegal to unlink a directory (except for the super-user).

ASSEMBLER

(unlink = 10.)
sys unlink; name

UTIME(2)

NAME

utime – set file times

SYNTAX

#include <sys/types.h>
utime(file, timep)
char *file;
time_t timep[2];

DESCRIPTION

The *utime* call uses the 'accessed' and 'updated' times in that order from the *timep* vector to set the corresponding recorded times for *file* .

The *timep* vector is a 2 element array of time_t types.

The caller must be the owner of the file or the super-user. The 'inode-changed' time of the file is set to the current time.

SEE ALSO

stat (2)

ASSEMBLER

(utime = 30.)
sys utime; file; timep

WAIT(2)

NAME

wait – wait for process to terminate

SYNTAX

wait(status)
int *status;

wait(0)

DESCRIPTION

Wait causes its caller to delay until a signal is received or one of its child processes terminates. If any child has died since the last *wait* , return is immediate; if there are no children, return is immediate with an error indication. The normal return yields the process ID of the terminated child. In the case of several children several *wait* calls are needed to learn of all the deaths.

If *(int) status* is nonzero, the high byte of the word pointed to receives the low byte of the argument of *exit* when the child terminated. The low byte receives the termination status of the process. See *signal (2)* for a list of termination statuses (signals); 0 status indicates normal termination. A special status (0177) is returned for a stopped process which has not terminated and can be restarted. See *ptrace (2)* . If the 0200 bit of the termination status is set, a memory image of the process was produced by the system.

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

SEE ALSO

exit(2), fork(2), signal(2)

DIAGNOSTICS

Returns −1 if there are no children not previously waited for.

ASSEMBLER

(wait = 7.)
sys wait
(process ID in r0)
(status in r1)

The high byte of the status is the low byte of r0 in the child at termination.

WRITE(2)

NAME

write – write on a file

SYNTAX

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

DESCRIPTION

A file descriptor is a word returned from a successful *open* , *creat* , *dup* , or *pipe (2)* call.

Buffer is the address of *nbytes* contiguous bytes which are written on the output file. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Writes which are multiples of 512 characters long and begin on a 512-byte boundary in the file are more efficient than any others.

SEE ALSO

creat(2), open(2), pipe(2)

DIAGNOSTICS

Returns −1 on error: bad descriptor, buffer address, or count; physical I/O errors.

ASSEMBLER

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

Section 3
Standard Subroutines

INTRODUCTION

The *standard subroutines* described in this section are a library of programming tools available to the user. These software development tools are intended to simplify the task of programming by providing you with a set of standardized, debugged routines which can be readily incorporated into your programs.

INTRO(3)

NAME

intro – introduction to library functions

SYNTAX

#include <stdio.h>

#include <math.h>

DESCRIPTION

This section describes functions that may be found in various libraries, other than those functions that directly invoke TNIX system primitives, which are described in section 2. Functions are divided into various libraries distinguished by the section number at the top of the page:

(3) These functions, together with those of section 2 and those marked (3S), constitute library *libc,* which is automatically loaded by the C compiler *cc (1)* and the Fortran compiler *f77 (1).* The link editor *ld (1)* searches this library under the '–lc' option. Declarations for some of these functions may be obtained from include files indicated on the appropriate pages.

(3M) These functions constitute the math library, *libm.* They are automatically loaded as needed by the Fortran compiler *f77 (1).* The link editor searches this library under the '–lm' option. Declarations for these functions may be obtained from the include file <math.h>.

(3S) These functions constitute the 'standard I/O package', see *stdio (3).* These functions are in the library *libc* already mentioned. Declarations for these functions may be obtained from the include file <stdio.h>.

(3X) Various specialized libraries have not been given distinctive captions. The files in which these libraries are found are named on the appropriate pages.

FILES

/lib/libc.a
/lib/libm.a, /usr/lib/libm.a (one or the other)

SEE ALSO

stdio(3), nm(1), ld(1), cc(1), f77(1), intro(2)

DIAGNOSTICS

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases the external variable *errno* (see *intro (2))* is set to the value EDOM or ERANGE. The values of EDOM and ERANGE are defined in the include file *<math.h>.*

ASSEMBLER

In assembly language these functions may be accessed by simulating the C calling sequence. For example, *ecvt (3)* might be called this way:

```
setd
mov     $sign,–(sp)
mov     $decpt,–(sp)
mov     ndigit,–(sp)
movf    value,–(sp)
jsr     pc,_ecvt
add     $14.,sp
```

ABORT(3)

NAME

abort – generate IOT fault

DESCRIPTION

Abort executes the PDP11 IOT instruction. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

SEE ALSO

adb(1), signal(2), exit(2)

DIAGNOSTICS

Usually 'IOT trap – core dumped' from the shell.

ABS(3)

NAME

abs – integer absolute value

SYNTAX

abs(i)

DESCRIPTION

Abs returns the absolute value of its integer operand.

SEE ALSO

floor(3) for *fabs*

NOTES

You get what the hardware gives on the largest negative integer.

ASSERT(3X)

NAME

assert – program verification

SYNTAX

#include <assert.h>

assert (expression)

DESCRIPTION

Assert is a macro that indicates *expression* is expected to be true at this point in the program. It causes an *exit (2)* with a diagnostic comment on the standard output when *expression* is false (0). Compiling with the *cc (1)* option **–DNDEBUG** effectively deletes *assert* from the program.

DIAGNOSTICS

'Assertion failed: file *f* line *n*.' *F* is the source file and *n* the source line number of the *assert* statement.

ATOF/ATOI/ATOL(3)

NAME

atof, atoi, atol – convert ASCII to numbers

SYNTAX

double atof(nptr)
char *nptr;

atoi(nptr)
char *nptr;

long atol(nptr)
char *nptr;

DESCRIPTION

These functions convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional 'e' or 'E' followed by an optionally signed integer.

Atoi and *atol* recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3)

NOTES

There are no provisions for overflow.

CRYPT/SETKEY/ENCRYPT(3)

NAME

crypt, setkey, encrypt – DES encryption

SYNTAX

```
char *crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;
```

DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to *crypt* is a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the *encrypt* entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

NOTES

The return value points to static data whose content is overwritten by each call.

CTIME(3)

NAME

ctime, localtime, gmtime, asctime, timezone – convert date and time to ASCII

SYNTAX

```
char *ctime(clock)
long *clock;

#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

DESCRIPTION

Ctime converts a time pointed to by *clock* such as returned by *time (2)* into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

Localtime and *gmtime* return pointers to structures containing the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time TNIX uses. *Asctime* converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
struct tm { /* see ctime(3) */
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday = 0), year – 1900, day of year (0-365), and a flag that is nonzero if daylight saving time is in effect.

When local time is called for, the program consults the system to determine the time zone and whether the standard U.S.A. daylight saving time adjustment is appropriate. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

Timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; e.g. in Afghanistan *timezone(−(60*4+30), 0)* is appropriate because it is 4:30 ahead of GMT and the string **GMT+4:30** is produced.

SEE ALSO

time(2)

NOTES

The return values point to static data whose content is overwritten by each call.

CTYPE(3)

NAME

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii – character classification

SYNTAX

#include <ctype.h>

isalpha(c)

. . .

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value EOF (see *stdio (3)).*

isalpha
c is a letter

isupper
c is an upper case letter

islower
c is a lower case letter

isdigit *c* is a digit

isalnum
c is an alphanumeric character

isspace
c is a space, tab, carriage return, newline, or formfeed

ispunct
c is a punctuation character (neither control nor alphanumeric)

isprint *c* is a printing character, code 040(8) (space) through 0176 (tilde)

iscntrl *c* is a delete character (0177) or ordinary control character (less than 040).

isascii *c* is an ASCII character, code less than 0200

SEE ALSO

ascii(7)

DBM(3X)

NAME

dbminit, fetch, store, delete, firstkey, nextkey – data base subroutines

SYNTAX

typedef struct { char *dptr; int dsize; } datum;

dbminit(file) char *file;

datum fetch(key) datum key;

store(key, content) datum key, content;

delete(key) datum key;

datum firstkey();

datum nextkey(key); datum key;

DESCRIPTION

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two filesystem accesses. The functions are obtained with the loader option **–ldbm** .

Keys and *contents* are described by the *datum* typedef. A *datum* specifies a string of *dsize* bytes pointed to by *dptr.* Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has '.dir' as its suffix. The second file contains all data and has '.pag' as its suffix.

Before a database can be accessed, it must be opened by *dbminit.* At the time of this call, the files *file .dir* and *file .pag* must exist. (An empty database is created by creating zero-length '.dir' and '.pag' files.)

Once open, the data stored under a key is accessed by *fetch* and data is placed under a key by *store* . A key (and its associated contents) is deleted by *delete* . A linear pass through all keys in a database may be made, in an (apparently) random order, by use of *firstkey* and *nextkey* . *Firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the data base:

```
for(key=firstkey(); key.dptr!=NULL; key=nextkey(key))
```

DIAGNOSTICS

All functions that return an *int* indicate errors with negative values. A zero return indicates ok. Routines that return a *datum* indicate errors with a null (0) *dptr.*

NOTES

The '.pag' file will contain holes so that its apparent size is about four times its actual content. Older TNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

Dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 512 bytes). Moreover all key/content pairs that hash together must fit on a single block. *Store* will return an error in the event that a disk block fills with inseparable data.

Delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by *firstkey* and *nextkey* depends on a hashing function, not on anything interesting.

ECVT/FCVT/GCVT(3)

NAME

ecvt, fcvt, gcvt – output conversion

SYNTAX

char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
char *buf;

DESCRIPTION

Ecvt converts the *value* to a null-terminated string of *ndigit* ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to ecvt, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by *ndigits* .

Gcvt converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf.* It attempts to produce *ndigit* significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3)

NOTES

The return values point to static data whose content is overwritten by each call.

END/ETEXT/EDATA(3)

NAME

end, etext, edata – last locations in program

SYNTAX

extern end;
extern etext;
extern edata;

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region.

When execution begins, the program break coincides with *end,* but many functions reset the program break, among them the routines of *brk (2), malloc (3),* standard input/output *(stdio (3)),* the profile (**–p**) option of *cc (1),* etc. The current value of the program break is reliably returned by 'sbrk(0)', see *brk (2).*

SEE ALSO

brk(2), malloc(3)

EXP/LOG/LOG10/POW/SQRT(3M)

NAME

exp, log, log10, pow, sqrt – exponential, logarithm, power, square root

SYNTAX

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

DESCRIPTION

Exp returns the exponential function of *x*.

Log returns the natural logarithm of *x* ; *log10* returns the base 10 logarithm.

Pow returns x^y.

Sqrt returns the square root of *x*.

SEE ALSO

hypot(3), sinh(3), intro(2)

DIAGNOSTICS

Exp and *pow* return a huge value when the correct value would overflow; *errno* is set to ERANGE. *Pow* returns 0 and sets *errno* to EDOM when the second argument is negative and non-integral and when both arguments are 0.

Log returns 0 when *x* is zero or negative; *errno* is set to EDOM.

Sqrt returns 0 when *x* is negative; *errno* is set to EDOM.

FCLOSE(3S)

NAME

fclose, fflush – close or flush a stream

SYNTAX

#include <stdio.h>

fclose(stream)
FILE ***stream;**

fflush(stream)
FILE ***stream;**

DESCRIPTION

Fclose causes any buffers for the named *stream* to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling *exit (2).*

Fflush causes any buffered data for the named output *stream* to be written to that file. The stream remains open.

SEE ALSO

close(2), fopen(3), setbuf(3)

DIAGNOSTICS

These routines return EOF if *stream* is not associated with an output file, or if buffered data cannot be transferred to that file.

FERROR/FEOF/CLEARERR/FILENO(3S)

NAME

feof, ferror, clearerr, fileno – stream status inquiries

SYNTAX

#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream;

DESCRIPTION

Feof returns non-zero when end of file is read on the named input *stream,* otherwise zero.

Ferror returns non-zero when an error has occurred reading or writing the named *stream,* otherwise zero. Unless cleared by *clearerr,* the error indication lasts until the stream is closed.

Clrerr resets the error indication on the named *stream.*

Fileno returns the integer file descriptor associated with the *stream,* see *open (2).*

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3), open(2)

FLOOR(3M)

NAME

fabs, floor, ceil – absolute value, floor, ceiling functions

SYNTAX

#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double(x);

DESCRIPTION

Fabs returns the absolute value| *x* |.

Floor returns the largest integer not greater than *x* .

Ceil returns the smallest integer not less than *x* .

SEE ALSO

abs(3)

FOPEN/FREOPEN/FDOPEN(3S)

NAME

fopen, freopen, fdopen – open a stream

SYNTAX

#include <stdio.h>

FILE ***fopen(filename, type)**
char *filename, *type;

FILE ***freopen(filename, type, stream)**
char *filename, *type;
FILE ***stream;**

FILE ***fdopen(fildes, type)**
char *type;

DESCRIPTION

Fopen opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

r	open for reading
w	create for writing
a	append: open for writing at end of file, or create for writing

If a + sign is included in the type string after the type identifier, then the stream will be opened for reading and writing (i.e. mode 2; see open(2)).

Freopen substitutes the named file in place of the open *stream* . It returns the original value of *stream* . The original stream is closed.

Freopen is typically used to attach the preopened constant names, **stdin, stdout, stderr,** to specified files.

Fdopen associates a stream with a file descriptor obtained from *open, dup, creat,* or *pipe (2).* The *type* of the stream must agree with the mode of the open file.

SEE ALSO

open(2), fclose(3)

DIAGNOSTICS

Fopen and *freopen* return the pointer NULL if *filename* cannot be accessed.

NOTES

Fdopen is not portable to systems other than TNIX. *Types* **a+** r+ and **w+** may not be portable. Use with care.

FREAD/FWRITE(3S)

NAME

fread, fwrite – buffered binary input/output

SYNTAX

#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

DESCRIPTION

Fread reads, into a block beginning at *ptr, nitems* of data of the type of **ptr* from the named input *stream* . It returns the number of items actually read.

Fwrite appends at most *nitems* of data of the type of **ptr* beginning at *ptr* to the named output *stream* . It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3), getc(3), putc(3), gets(3), puts(3), printf(3), scanf(3)

DIAGNOSTICS

Fread and *fwrite* return 0 upon end of file or error.

FREXP/LDEXP/MODF(3)

NAME

frexp, ldexp, modf – split into mantissa and exponent

SYNTAX

double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;

double modf(value, iptr)
double value, *iptr;

DESCRIPTION

Frexp returns the mantissa of a double *value* as a double quantity, *x*, of magnitude less than 1 and stores an integer *n* such that *value* = *x* *2** *n* indirectly through *eptr.*

Ldexp returns the quantity *value *2** exp.*

Modf returns the positive fractional part of *value* and stores the integer part indirectly through *iptr.*

FSEEK/FTELL/REWIND(3S)

NAME

fseek, ftell, rewind – reposition a stream

SYNTAX

#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream* . The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

Fseek undoes any effects of *ungetc (3).*

Ftell returns the current value of the offset relative to the beginning of the file associated with the named *stream* . It is measured in bytes on TNIX; on some other systems it is a magic cookie, and the only foolproof way to obtain an *offset* for *fseek* .

Rewind(stream) is equivalent to *fseek(stream, 0L, 0).*

SEE ALSO

lseek(2), fopen(3)

DIAGNOSTICS

Fseek returns −1 for improper seeks.

GETC/GETCHAR/FGETC/GETW(3S)

NAME

getc, getchar, fgetc, getw – get character or word from stream

SYNTAX

#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;

DESCRIPTION

Getc returns the next character from the named input *stream* .

Getchar() is identical to *getc(stdin)* .

Fgetc behaves like *getc,* but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next word from the named input *stream* . It returns the constant EOF upon end of file or error, but since that is a good integer value, *feof* and *ferror (3)* should be used to check the success of *getw* . *Getw* assumes no special alignment in the file.

SEE ALSO

fopen(3), putc(3), gets(3), scanf(3), fread(3), ungetc(3)

DIAGNOSTICS

These functions return the integer constant EOF at end of file or upon read error.

A stop with message, 'Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by *fopen* .

NOTES

Because it is implemented as a macro, *getc* treats a *stream* argument with side effects incorrectly. In particular, 'getc(*f++);' doesn't work sensibly.

GETENV(3)

NAME

getenv – value for environment name

SYNTAX

char *getenv(name)
char *name;

DESCRIPTION

Getenv searches the environment list (see *environ (5))* for a string of the form *name = value* and returns *value* if such a string is present, otherwise 0 (NULL).

SEE ALSO

environ(5), exec(2)

GETGRENT(3)

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent – get group file entry

SYNTAX

```
#include <grp.h>

struct group *getgrent();

struct group *getgrgid(gid) int gid;

struct group *getgrnam(name) char *name;

int setgrent();

int endgrent();
```

DESCRIPTION

Getgrent, getgrgid and *getgrnam* each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
struct   group { /* see getgrent(3) */
         char     *gr_name;
         char     *gr_passwd;
         int      gr_gid;
         char     **gr_mem;
};
```

The members of this structure are:

gr_name
The name of the group.

gr_passwd
The encrypted password of the group.

gr_gid The numerical group-ID.

gr_mem
Null-terminated vector of pointers to the individual member names.

Getgrent simply reads the next line while *getgrgid* and *getgrnam* search until a matching *gid* or *name* is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete.

FILES

/etc/group

SEE ALSO

getlogin(3), getpwent(3), group(5)

DIAGNOSTICS

A null pointer (0) is returned on EOF or error.

NOTES

All information is contained in a static area so it must be copied if it is to be saved.

GETLOGIN(3)

NAME

getlogin – get login name

SYNTAX

char *getlogin();

DESCRIPTION

Getlogin returns a pointer to the login name as found in */etc/utmp* . It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same userid is shared by several login names.

If *getlogin* is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to first call *getlogin* and if it fails, to call *getpwuid* .

FILES

/etc/utmp

SEE ALSO

getpwent(3), getgrent(3), utmp(5)

DIAGNOSTICS

Returns NULL (0) if name not found.

NOTES

The return values point to static data whose content is overwritten by each call.

GETPASS(3)

NAME

getpass – read a password

SYNTAX

char *getpass(prompt)
char *prompt;

DESCRIPTION

Getpass reads a password from the file */dev/tty*, or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES

/dev/tty

SEE ALSO

crypt(3)

NOTES

The return value points to static data whose content is overwritten by each call.

GETPW(3)

NAME

getpw – get name from UID

SYNTAX

getpw(uid, buf)
char *buf;

DESCRIPTION

Getpw searches the password file for the (numerical) *uid , and fills in buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

FILES

/etc/passwd

SEE ALSO

getpwent(3), passwd(5)

DIAGNOSTICS

Non-zero return on error.

GETPWENT(3)

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent – get password file entry

SYNTAX

```
#include <pwd.h>

struct passwd *getpwent();

struct passwd *getpwuid(uid) int uid;

struct passwd *getpwnam(name) char *name;

int setpwent();

int endpwent();
```

DESCRIPTION

Getpwent, getpwuid and *getpwnam* each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct  passwd { /* see getpwent(3) */
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        int     pw_quota;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};
```

The fields *pw_quota* and *pw_comment* are unused; the others have meanings described in *passwd (5).*

Getpwent reads the next line (opening the file if necessary); *setpwent* rewinds the file; *endpwent* closes it.

Getpwuid and *getpwnam* search from the beginning until a matching *uid* or *name* is found (or until EOF is encountered).

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), passwd(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

NOTES

All information is contained in a static area so it must be copied if it is to be saved.

GETS/FGETS(3S)

NAME

gets, fgets – get a string from a stream

SYNTAX

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;
```

DESCRIPTION

Gets reads a string into *s* from the standard input stream **stdin**. The string is terminated by a newline character, which is replaced in *s* by a null character. *Gets* returns its argument.

Fgets reads *n –1* characters, or up to a newline character, whichever comes first, from the *stream* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its first argument.

SEE ALSO

puts(3), getc(3), scanf(3), fread(3), ferror(3)

DIAGNOSTICS

Gets and *fgets* return the constant pointer **NULL** upon end of file or error.

NOTES

Gets deletes a newline, *fgets* keeps it, all in the name of backward compatibility.

HYPOT/CABS(3M)

NAME

hypot, cabs – euclidean distance

SYNTAX

#include <math.h>

double hypot(x, y)
double x, y;

double cabs(z)
struct { double x, y;} z;

DESCRIPTION

Hypot and *cabs* return

15n sqrt(x*x + y*y),

taking precautions against unwarranted overflows.

SEE ALSO

exp(3) for *sqrt*

J0/J1/JN/Y0/Y1/YN(3M)

NAME

j0, j1, jn, y0, y1, yn – bessel functions

SYNTAX

```
#include <math.h>

double j0(x)
double x;

double j1(x)
double x;

double jn(n, x);
double x;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause *y0, y1,* and *yn* to return a huge negative value and set *errno* to EDOM.

L3TOL/LTOL3(3)

NAME

l3tol, ltol3 – convert between 3-byte integers and long integers

SYNTAX

```
l3tol(lp, cp, n)
long *lp;
char *cp;

ltol3(cp, lp, n)
char *cp;
long *lp;
```

DESCRIPTION

L3tol converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp* .

Ltol3 performs the reverse conversion from long integers *(lp)* to three-byte integers *(cp).*

These functions are useful for file-system maintenance; disk addresses are three bytes long.

SEE ALSO

filsys(5)

MALLOC/FREE/REALLOC/CALLOC(3)

NAME

malloc, free, realloc, calloc – main memory allocator

SYNTAX

char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

DESCRIPTION

Malloc and *free* provide a simple general-purpose memory allocation package. *Malloc* returns a pointer to a block of at least *size* bytes beginning on a word boundary.

The argument to *free* is a pointer to a block previously allocated by *malloc ;* this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by *malloc* is overrun or if some random number is handed to *free* .

Malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls *sbrk* (see *break (2))* to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

Realloc also works if *ptr* points to a block freed since the last call of *malloc, realloc* or *calloc ;* thus sequences of *free, malloc* and *realloc* can exploit the search strategy of *malloc* to do storage compaction.

Calloc allocates space for an array of *nelem* elements of size *elsize.* The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS

Malloc, realloc and *calloc* return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. *Malloc* may be recompiled to check the arena very stringently on every transaction; see the source code.

NOTES

When *realloc* returns 0, the block pointed to by *ptr* may be destroyed.

MKTEMP(3)

NAME

mktemp – make a unique file name

SYNTAX

char *mktemp(template)
char *template;

DESCRIPTION

Mktemp replaces *template* by a unique file name, and returns the address of the template. The template should look like a file name with six trailing X's, which will be replaced with the current process id and a unique letter.

SEE ALSO

getpid(2)

MONITOR(3)

NAME

monitor – prepare execution profile

SYNTAX

monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
short buffer[ ];

DESCRIPTION

An executable program created by 'cc –p' automatically includes calls for *monitor* with default parameters; *monitor* needn't be called explicitly except to gain fine control over profiling.

Monitor is an interface to *profil (2). Lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* short integers. *Monitor* arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc* . At most *nfunc* call counts can be kept; only calls of functions compiled with the profiling option **–p** of *cc (1)* are recorded. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

```
extern etext();
...
monitor((int)2, etext, buf, bufsize, nfunc);
```

Etext lies just above all the program text, see *end (3).*

To stop execution monitoring and write the results on the file *mon.out,* use

```
monitor(0);
```

then *prof (1)* can be used to examine the results.

FILES

mon.out

SEE ALSO

prof(1), profil(2), cc(1)

MP(3X)

NAME

itom, madd, msub, mult, mdiv, min, mout, pow, gcd, rpow − multiple precision integer arithmetic

SYNTAX

typedef struct { int len; short *val; } mint;

madd(a, b, c) msub(a, b, c) mult(a, b, c) mdiv(a, b, q, r) min(a) mout(a) pow(a, b, m, c) gcd(a, b, c) rpow(a, b, c) msqrt(a, b, r) mint *a, *b, *c, *m, *q, *r;

sdiv(a, n, q, r) mint *a, *q; short *r;

mint *itom(n)

DESCRIPTION

These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type *mint.* Pointers to a *mint* should be initialized using the function *itom* , which sets the initial value to *n* . After that space is managed automatically by the routines.

madd , *msub* , *mult* , assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. *mdiv* assigns the quotient and remainder, respectively, to its third and fourth arguments. *sdiv* is like *mdiv* except that the divisor is an ordinary integer. *msqrt* produces the square root and remainder of its first argument. *rpow* calculates *a* raised to the power *b* , while *pow* calculates this reduced modulo *m* . *min and mout* do decimal input and output.

The functions are obtained with the loader option *-lmp* .

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

NLIST(3)

NAME

nlist – get entries from name list

SYNTAX

```
#include <a.out.h>
nlist(filename, nl)
char *filename;
struct nlist nl[ ];
```

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See *a.out (5)* for the structure declaration.

This subroutine is useful for examining the system name list kept in the file /unix . In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

PERROR(3)

NAME

perror, sys_errlist, sys_nerr – system error messages

SYNTAX

```
perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];
```

DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable *errno* (see *intro (2)),* which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the newline. *Sys_nerr* is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2)

PKOPEN(3)

NAME

pkopen, pkclose, pkread, pkwrite, pkfail – packet driver simulator

SYNTAX

char *pkopen(fd)

pkclose(ptr)
char *ptr;

pkread(ptr, buffer, count)
char *ptr, *buffer;

pkwrite(ptr, buffer, count)
char *ptr, *buffer;

pkfail()

DESCRIPTION

These routines are a user-level implementation of the full-duplex end-to-end communication protocoi described in *pk (4).* If *fd* is a file descriptor open for reading and writing, *pkopen* carries out the initial synchronization and returns an identifying pointer. The pointer is used as the first parameter to *pkread, pkwrite,* and *pkclose.*

Pkread, pkwrite and *pkclose* behave analogously to *read, write* and *close (2).* However, a write of zero bytes is meaningful and will produce a corresponding read of zero bytes.

SEE ALSO

pk(4), pkon(2)

DIAGNOSTICS

Pkfail is called upon persistent breakdown of communication. *Pkfail* must be supplied by the user.

Pkopen returns a null (0) pointer if packet protocol can not be established.

Pkread returns −1 on end of file, 0 in correspondence with a 0-length write.

NOTES

This simulation of *pk (4)* leaves something to be desired in needing special read and write routines, and in not being inheritable across calls of *exec (2).* Its prime use is on systems that lack *pk.*
These functions use *alarm (2);* simultaneous use of *alarm* for other puposes may cause trouble.

PLOT(3X)

NAME

plot: openpl et al. – graphics interface

SYNTAX

openpl( )

erase( )

label(s) char s[ ];

line(x1, y1, x2, y2)

circle(x, y, r)

arc(x, y, x0, y0, x1, y1)

move(x, y)

cont(x, y)

point(x, y)

linemod(s) char s[ ];

space(x0, y0, x1, y1)

closepl( )

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. See *plot (5)* for a description of their effect. *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

String arguments to *label* and *linemod* are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following *ld (1)* options:

–lplot	device-independent graphics stream on standard output for *plot (1)* filters
–l300	GSI 300 terminal
–l300s	GSI 300S terminal
–l450	DASI 450 terminal
–l4014	Tektronix 4014 terminal

SEE ALSO

plot(5), plot(1), graph(1)

POPEN/PCLOSE(3S)

NAME

popen, pclose – initiate I/O to/from a process

SYNTAX

#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;

DESCRIPTION

The arguments to *popen* are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by *popen* should be closed by *pclose,* which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

SEE ALSO

pipe(2), fopen(3), fclose(3), system(3), wait(2)

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

Pclose returns −1 if *stream* is not associated with a 'popened' command.

NOTES

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, e.g. with *fflush,* see *fclose (3).*

PRINTF/FPRINT/SPRINTF(3S)

NAME

printf, fprintf, sprintf – formatted output conversion

SYNTAX

```
#include <stdio.h>

printf(format [ , arg ] ... )
char *format;

fprintf(stream, format [ , arg ] ... )
FILE *stream;
char *format;

sprintf(s, format [ , arg ] ... )
char *s, format;
```

DESCRIPTION

Printf places output on the standard output stream *stdout*. *Fprintf* places output on the named output *stream*. *Sprintf* places 'output' in the string *s*, followed by the character '\0'.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive *arg printf*.

Each conversion specification is introduced by the character **%**. Following the **%**, there may be

– an optional minus sign '–' which specifies *left adjustment* of the converted value in the indicated field;

– an optional digit string specifying a *field width;* if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;

– an optional period ' . ' which serves to separate the field width from the next digit string;

– an optional digit string specifying a *precision* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

– the character l specifying that a following **d** , **o** , **x** , or **u** corresponds to a long integer *arg*. (A capitalized conversion code accomplishes the same thing.)

– a character which indicates the type of conversion to be applied.

A field width or precision may be '*' instead of a digit string. In this case an integer *arg* supplies the field width or precision.

The conversion characters and their meanings are

dox The integer *arg* is converted to decimal, octal, or hexadecimal notation respectively.

f The float or double *arg* is converted to decimal notation in the style '[−]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e The float or double *arg* is converted in the style '[−]d.ddd**e**± dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.

g The float or double *arg* is printed in style **d** , in style **f** , or in style **e** , whichever gives full precision in minimum space.

c The character *arg* is printed. Null characters are ignored.

s *Arg* is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.

u The unsigned integer *arg* is converted to decimal and printed (the result will be in the range 0 to 65535).

% Print a '%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by *putc (3).*

Examples
To print a date and time in the form 'Sunday, July 3, 10:02', where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print π to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO
putc(3), scanf(3), ecvt(3)

NOTES
Very wide fields (> 128 characters) fail.

PUTC/PUTCHAR/FPUTC/PUTW(3S)

NAME

putc, putchar, fputc, putw – put character or word on a stream

SYNTAX

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;
```

DESCRIPTION

Putc appends the character *c* to the named output *stream* . It returns the character written.

Putchar(c) is defined as *putc(c, stdout).*

Fputc behaves like *putc,* but is a genuine function rather than a macro. It may be used to save on object text.

Putw appends word (i.e. **int**) *w* to the output *stream* . It returns the word written. *Putw* neither assumes nor causes special alignment in the file.

The standard stream *stdout* is normally buffered if and only if the output does not refer to a terminal; this default may be changed by *setbuf (3).* The standard stream *stderr* is by default unbuffered unconditionally, but use of *freopen* (see *fopen (3))* will cause it to become buffered; *setbuf ,* again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. *Fflush* (see *fclose (3))* may be used to force the block out early.

SEE ALSO

fopen(3), fclose(3), getc(3), puts(3), printf(3), fread(3)

DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, *ferror (3)* should be used to detect *putw* errors.

NOTES

Because it is implemented as a macro, *putc* treats a *stream* argument with side effects improperly. In particular 'putc(c, *f++);' doesn't work sensibly.

PUTS/FPUTS(3S)

NAME

puts, fputs – put a string on a stream

SYNTAX

#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;

DESCRIPTION

Puts copies the null-terminated string *s* to the standard output stream *stdout* and appends a newline character.

Fputs copies the null-terminated string *s* to the named output *stream* .

Neither routine copies the terminal null character.

SEE ALSO

fopen(3), gets(3), putc(3), printf(3), ferror(3)
fread(3) for *fwrite*

NOTES

Puts appends a newline, *fputs* does not, all in the name of backward compatibility.

QSORT(3)

NAME

qsort – quicker sort

SYNTAX

qsort(base, nel, width, compar)
char *base;
int (*compar)( );

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO

sort(1)

RAND(3)

NAME

rand, srand – random number generator

SYNTAX

srand(seed)
int seed;

rand( )

DESCRIPTION

Rand uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range from 0 to $2^{15}-1$.

The generator is reinitialized by calling *srand* with 1 as argument. It can be set to a random starting point by calling *srand* with whatever you like as argument.

SCANF/FSCANF/SSCANF(3S)

NAME

scanf, fscanf, sscanf – formatted input conversion

SYNTAX

#include <stdio.h>

scanf(format [, pointer] . . .)
char *format;

fscanf(stream, format [, pointer] . . .)
FILE *stream;
char *format;

sscanf(s, format [, pointer] . . .)
char *s, *format;

DESCRIPTION

Scanf reads from the standard input stream *stdin* . *Fscanf* reads from the named input *stream* . *Sscanf* reads from the character string *s* . Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string *format,* described below, and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.
2. An ordinary character (not %) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character **%** , an optional assignment suppressing character ***** , an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by ***** . An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

%	a single '%' is expected in the input at this point; no assignment is done.
d	a decimal integer is expected; the corresponding argument should be an integer pointer.
o	an octal integer is expected; the corresponding argument should be a integer pointer.
x	a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating '\0', which will be added. The input field is terminated by a space character or a newline.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

e
f a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.

[indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d** , **o** and **x** may be capitalized or preceeded by **l** to indicate that a pointer to **long** rather than to **int** is in the argument list. Similarly, the conversion characters **e** or f may be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float** . The conversion characters **d** , **o** and **x** may be preceeded by **h** to indicate a pointer to **short** rather than to **int** .

The *scanf* functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant **EOF** is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
10      int i; float x; char name[50];
        scanf( "%d%f%s", &i, &x, name);
```

with the input line

```
        25  54.32E−1  thompson
```

will assign to *i* the value 25, *x* the value 5.432, and *name* will contain *'thompson\0'*. Or,

```
        int i; float x; char name[50];
        scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

    56789 0123 56a72

will assign 56 to *i*, 789.0 to *x*, skip '0123', and place the string '56\0' in *name*. The next call to *getchar* will return 'a'.

SEE ALSO

atof(3), getc(3), printf(3)

DIAGNOSTICS

The *scanf* functions return EOF on end of input, and a short count for missing or illegal data items.

NOTES

The success of literal matches and suppressed assignments is not directly determinable.

SETBUF(3S)

NAME

setbuf – assign buffering to a stream

SYNTAX

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;
```

DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array *buf* to be used instead of an automatically allocated buffer. If *buf* is the constant pointer **NULL**, input/output will be completely unbuffered.

A manifest constant **BUFSIZ** tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from *malloc (3)* upon the first *getc* or *putc (3)* on the file, except that output streams directed to terminals, and the standard error stream *stderr* are normally not buffered.

SEE ALSO

fopen(3), getc(3), putc(3), malloc(3)

SETJMP/LONGJMP(3)

NAME

setjmp, longjmp – non-local goto

SYNTAX

#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;

DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in *env* for later use by *longjmp.* It returns value 0.

Longjmp restores the environment saved by the last call of *setjmp* . It then returns in such a way that execution continues as if the call of *setjmp* had just returned the value *val* to the function that invoked *setjmp,* which must not itself have returned in the interim. All accessible data have values as of the time *longjmp* was called.

SEE ALSO

signal(2)

SIN/COS/TAN/ASIN/ACOS/ATAN/ATAN2(3M)

NAME

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNTAX

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(x, y)
double x, y;
```

DESCRIPTION

Sin, cos and *tan* return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.

Acos returns the arc cosine in the range 0 to π.

Atan returns the arc tangent of *x* in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arc tangent of *x/y* in the range $-\pi$ to π.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause *asin* and *acos* to return value 0; *errno* is set to EDOM. The value of *tan* at its singular points is a huge number, and *errno* is set to ERANGE.

NOTES

The value of *tan* for arguments greater than about 2**31 is garbage.

SINH/COSH/TANH(3M)

NAME

sinh, cosh, tanh – hyperbolic functions

SYNTAX

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;
```

DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and *cosh* return a huge value of appropriate sign when the correct value would overflow.

SLEEP(3)

NAME

sleep – suspend execution for interval

SYNTAX

sleep(seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

alarm(2), pause(2)

STDIO(3S)

NAME

stdio – standard buffered input/output package

SYNTAX

#include <stdio.h>

FILE ***stdin;**
FILE ***stdout;**
FILE ***stderr;**

DESCRIPTION

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros *getc* and *putc (3)* handle characters quickly. The higher level routines *gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite* all use *getc* and *putc;* they can be freely intermixed.

A file with associated buffering is called a *stream,* and is declared to be a pointer to a defined type FILE. *Fopen (3)* creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant 'pointer' NULL (0) designates no stream at all.

An integer constant EOF (–1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros; redeclaration of these names is perilous: *getc, getchar, putc, putchar, feof, ferror, fileno .*

SEE ALSO

open(2), close(2), read(2), write(2)

DIAGNOSTICS

The value EOF is returned uniformly to indicate that a FILE pointer has not been initialized with *fopen,* input (output) has been attempted on an output (input) stream, or a FILE pointer designates corrupt or otherwise unintelligible FILE data.

STRING(3)

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex – string operations

SYNTAX

```
char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s;
```

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string *s2* to the end of string *s1* . *Strncat* copies at most *n* characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as *s1* is lexicographically greater than, equal to, or less than *s2* . *Strncmp* makes the same comparison but looks at at most *n* characters.

Strcpy copies string *s2* to *s1,* stopping after the null character has been moved. *Strncpy* copies exactly *n* characters, truncating or null-padding *s2;* the target may not be null-terminated if the length of *s2* is *n* or more. Both return *s1* .

Strlen returns the number of non-null characters in *s* .

Index (rindex) returns a pointer to the first (last) occurrence of character *c* in string *s,* or zero if *c* does not occur in the string.

NOTES

Strcmp uses native character comparison, which is signed on PDP11's, unsigned on other machines.

SWAB(3)

NAME

swab – swap bytes

SYNTAX

swab(from, to, nbytes)
char *from, *to;

DESCRIPTION

Swab copies *nbytes* bytes pointed to by *from* to the position pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP11's and other machines. *Nbytes* should be even.

SYSTEM(3)

NAME

system – issue a shell command

SYNTAX

system(string)
char *string;

DESCRIPTION

System causes the *string* to be given to *sh (1)* as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO

popen(3), exec(2), wait(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

TTYNAME/ISATTY/TTYSLOT(3)

NAME

ttyname, isatty, ttyslot – find name of a terminal

SYNTAX

char *ttyname(fildes)

isatty(fildes)

ttyslot()

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *fildes* .

Isatty returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

Ttyslot returns the number of the entry in the *ttys (5)* file for the control terminal of the current process.

FILES

/dev/*
/etc/ttys

SEE ALSO

ioctl(2), ttys(5)

DIAGNOSTICS

Ttyname returns a null pointer (0) if *fildes* does not describe a terminal device in directory '/dev'.

Ttyslot returns 0 if '/etc/ttys' is inaccessible or if it cannot determine the control terminal.

NOTES

The return value points to static data whose content is overwritten by each call.

UNGETC(3S)

NAME

ungetc – push character back into input stream

SYNTAX

#include <stdio.h>

ungetc(c, stream)
FILE *stream;

DESCRIPTION

Ungetc pushes the character *c* back on an input stream. That character will be returned by the next *getc* call on that stream. *Ungetc* returns *c* .

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek (3) erases all memory of pushed back characters.

SEE ALSO

getc(3), setbuf(3), fseek(3)

DIAGNOSTICS

Ungetc returns **EOF** if it can't push a character back.

Section 4 Special Files

INTRODUCTION

This section discusses data transfer and access to/from special files. *Special files* are hardware input/output devices that appear to the user as regular files. Thus, inter-device file/data transfer within the **TNIX** operating system is straightforward. Terminals, printers, and disk drives are examples of some special files.

AUX(4)

NAME

aux - line printer

DESCRIPTION

Aux1 and **Aux2** are special files which access the line printers. They will provide tab to space substitution for printers without hardware tabs, and will provide end of line substitution. The most common use of end of line substitution is to change a <CR><LF> sequence to just <CR> or <LF>.

The command slp(1) can be used to set the end of line and tab options.

The ioctls which are needed for setting the end of line string and turning the tab expansion on and off are LPSETNL, LPTABON, and LPTABOFF, respectively. The formats of these requests are:

```
#include <lp.h>
ioctl(fildes,LPSETNL,buf)
char buf[8];
```

and

```
#include <lp.h>
ioctl(fildes,LPTABON)
```

and

```
#include <lp.h>
ioctl(fildes,LPTABOFF)
```

where buf is a null terminated string to substitute for the end of line character, linefeed. The string may be up to 8 characters long. Null characters may be inserted in the string by setting the high bit. This high bit will not be sent to the line printer.

FILES

/dev/aux1
/dev/aux2

SEE ALSO

slp(1)

FD(4)

NAME

fd—flexible disk

DESCRIPTION

The file **fd0** refers to the flexible disk drive. The **fd0** file accesses the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records.

There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation; therefore, raw I/O is considerably more efficient (for multiple block buffers when many words are transmitted). The name of the raw **fd0** file is **rfd0**. In raw I/O, the buffer must begin on a word boundary.

The **ioctl***(2)* device control system calls are available for the raw device. GETSIZ and PMSCMD are functions executed by the **ioctl***(2)* system call. The GETSIZ **ioctl***(2)* request returns the maximum block number on the device into a long integer. The PMSCMD **ioctl***(2)* request sends a direct command to the low-level Winchester disk controller and returns the response in the same location.

The GETSIZ and PMSCMD **ioctl***(2)* requests use the form:

```
#include <disk.h>
ioctl(fildes,GETSIZ,buf)
long *buf;
```

and

```
#include <disk.h>
ioctl(fildes,PMSCMD,buf)
unsigned buf[10];
```

If the drive number in the PMSCMD request does not correspond to one of the disk drives associated with the open disk device, **ioctl***(2)* returns an error. This keeps people from accessing any location on the fixed disk by opening the flexible disk or another fixed disk as a device and issuing an **ioctl***(2)* call.

FILES

/dev/fd0
/dev/rfd0

SEE ALSO

hd*(4)*, **ioctl***(2)*

NOTES

In raw I/O, **read***(2)* and **write***(2)* truncate file offsets to 512-byte block boundaries, and *write(2)* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read(2)*, *write(2)*, and **lseek***(2)* should always deal in 512-byte multiples.

HD(4)

NAME

hd—Winchester fixed disks

DESCRIPTION

The files *[r]***hd****, for example, hd0, rhd0, and hd03 refer to the Winchester fixed disk drives.

Files without a leading **r** (e.g., **hd03**) access the disks via the system's normal buffering mechanism and may be read and written without regard to physical disk records. Files containing a leading **r** (e.g., **rhd03**) are the 'raw' devices, providing direct transmission between a disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation; therefore, raw I/O is considerably more efficient when many words are transmitted.

Because a TNIX disk device can include one or more physical disk drives, the two final digits of the filename are the drive numbers of the first and last fixed disk drives associated with the TNIX device. The second digit is omitted if the TNIX device occupies only one fixed disk drive. For example, the special file /dev/hd1 occupies one fixed disk drive (drive 1); the special file /dev/hd02 includes the first three fixed disk drives (0—2, inclusive).

The minor device numbers of an **hd** device reflect this organization: the low two bits (0 and 1) give the first drive of the TNIX disk device; the next **bit***(2)* is always zero; the next two bits (3 and 4) give the final drive of the TNIX disk device.

The **ioctl***(2)* system calls are available for the raw disk devices. GETSIZ and PMSCMD are functions executed by the **ioctl***(2)* system call. GETSIZ will return the maximum block number on the disk device. PMSCMD will give a direct command to the Winchester disk controller and will return the response in the same location.

The GETSIZ and PMSCMD requests use the following form:

```
#include <disk.h>
ioctl(fildes,GETSIZ,buf)
long *buf;
```

and

```
#include <disk.h>
ioctl(fildes,PMSCMD,buf)
unsigned buf[10];
```

If the drive number in the PMSCMD request does not correspond to one of the physical disk drives associated with the open disk device, the **ioctl***(2)* system call returns an error. This keeps people from accessing any location on the fixed disk by opening the flexible disk or another fixed disk as a device and issuing an **ioctl***(2)* system call.

FILES

/dev/[r]hd*

SEE ALSO

fd*(4)*, **ioctl***(2)*.

NOTES

In raw I/O, **read***(2)* and **write***(2)* truncate file offsets to 512-byte block boundaries, and *write(2)* scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, *read(2)*, *write(2)*, and **lseek***(2)* should always deal in 512-byte multiples.

In raw I/O, the buffer must begin on a word boundary.

The result of a "get max disk block number" PMSCMD request will differ from the result of a GETSIZ operation. This is because:

- the last track of every fixed disk is reserved for diagnostics and therefore is not included in what GETSIZ returns,
- the Winchester disk controller thinks that each logical device occupies only one physical drive.

[This page intentionally left blank.]

HSI(4)

NAME

hsi, hsix – high speed interface protocols

DESCRIPTION

Hsi is the device name for the HSI protocol. The details of the HSI protocol are not available at this printing. The HSI protocol allows an error free transfer of blocks of data of up to 600 bytes in a block. This protocol is used in communication with the 8540. The device, hsix, uses the hsi protocol with a few changes to allow an end-of-file indication (a 0 length read) to be sent across the link. When the hsix device is closed after being open for write only, a zero length chunk is sent to indicate the end of file.

The hsix device may be used for transfering data from one 8560 to another. For example, the command 'cat </dev/hsix3 >filename' on one 8560, with the command 'cat filename >/dev/hsix3' on another 8560, would transfer a file from one 8560 to the other through the hsi cable connected to port 3 on both 8560s. The zero length block sent when the sending 8560 closes the hsix device will correctly terminate the command on the recieving 8560. The block size sent over the hsix device can be up to 598 bytes. The ports being used for this hsi transfer must be either strapped for hsi, or forced to hsi mode with a *stty(1)* command, such as 'stty IU >/dev/ttyn'.

FILES

/dev/hsi, /dev/hsix

SEE ALSO

stty(1), tty(4).

MEM(4)

NAME

mem, kmem – memory

DESCRIPTION

Mem is a special file that is an image of the memory of the computer. It may be used, for example, to examine, and even to patch the system. **Kmem** is the same as **mem** except that kernel virtual memory rather than physical memory is accessed.

Byte addresses are interpreted as memory addresses. References to non-existent locations return errors.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

On the 8560, the I/O page begins at location 0160000 of **kmem** and per-process data for the current process begins at 0140000.

FILES

/dev/mem, /dev/kmem

NOTES

On the 8560, memory files are accessed one byte at a time, an inapproriate method for some device registers.

NULL(4)

NAME

null – data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

TTY(4)

NAME

tty – general terminal interface

DESCRIPTION

This section describes both a particular special file, and the general nature of the terminal interface.

The file */dev/tty* is, in each process, a synonym for the control terminal associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

As for terminals in general: all of the low-speed asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of the interface.

When a terminal file is opened, it causes the process to wait until a connection is established. In practice user's programs seldom open these files; they are opened by *init* and become a user's input and output file. The very first terminal file open in a process becomes the *control terminal* for that process. The control terminal plays a special role in handling quit or interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*, even if the control terminal is closed. The set of processes that thus share a control terminal is called a *process group;* all members of a process group receive certain signals together, see ETX below and *kill(2).*

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program.

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. There are special modes, discussed below, that permit the program to read each character as typed without waiting for a full line.

During input, erase and kill processing is normally done. By default, the character <bs> erases the last character typed, except that it will not erase beyond the beginning of a line or an EOT. By default, the character <^U> kills the entire line up to the point where it was typed, but not beyond an EOT. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Either <^U> or <bs> may be entered literally by preceding it by '\'; the erase or kill character remains, but the '\' disappears.

The character <^R> will retype the line as input so far.

The character <^K> will retype successive characters of the previous line.

Certain ASCII control characters have special meaning. These characters are not passed to a reading program except in raw mode where they lose their special meaning. Also, it is possible to change these characters from the default; see below.

EOT (Control–D) may be used to generate an end of file from a terminal. When an EOT is received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOT is discarded. Thus if there are no characters waiting, which is to say the EOT occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file indication.

ETX (Control–C) is not passed to a program but generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. See *signal(2).*

FS (Control–\ or control–shift–L) generates the *quit* signal. Its treatment is identical to the interrupt signal except that unless a receiving process has made other arrangements it will not only be terminated but a memory image file will be generated.

DC3 (Control–S) delays all printing on the terminal until a DC1 is typed in (or when an interrupt or quit signal is generated by typing ETX or FS).

DC1 (Control–Q) restarts printing after DC3 without generating any input to a program.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a *hangup* signal is sent to all processes with the terminal as control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file on their input can terminate appropriately when hung up on.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Full documentation on the details of character processing is not available at this printing. The EOT character is not transmitted (except in raw mode) to prevent terminals that respond to it from hanging up.

Several *ioctl(2)* calls apply to terminals. Most of them use the following structure, defined in *<sgtty.h>:*

```
struct sgttyb {
        char sg_ispeed;
        char sg_ospeed;
        char sg_erase;
        char sg_kill;
        int  sg_flags;
};
```

This is the structure which is pointed to by *argp*. The sg_ispeed field describes the input and output speeds of the device according to the following table. The sg_ospeed field is no longer used. Symbolic values in this table are as defined in <sgtty.h>.

B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud

The sg_erase and sg_kill fields of the argument structure specify the erase and kill characters respectively. Setting the high bit of the erase or kill character will cause the erase or kill function to be the non-fancy variety. (Defaults are ^H and ^U.) Full documentation on the descriptions of fancy versus non-fancy features is not available at this printing.

The sg_flags field contains several bits that determine the system's treatment of the terminal. Bit fields within sg_flags will be followed by the values it can take on.

```
TANDEM      03
        possible values:
        00 off
        01 XON/XOFF flagging
        02 DTR flagging
CBREAK      04
ECHO  010
CRMOD 020
RAW   040
XTABS 0100
ARESM 0200
CTSFLG      0400
IPRTY 07000
        possible values:
        00000 don't care
        01000 even parity
        02000 odd
        03000 no parity (zero)
        04000 mark (one)
        05000 data (8th bit not stripped)
OPRTY 070000
        possible values:
        000000 don't care
        010000 even parity
        020000 odd
        030000 no parity (zero)
        040000 mark (one)
        050000 data (8th bit not stripped)
CHNGED      0100000
```

Characters with the wrong parity are ignored.

In raw mode, every character is passed immediately to the program without waiting until a full line has been typed. There is no special meaning associated with any character. For example, no erase or kill processing is done; the end-of-file indicator (EOT), the interrupt character (DEL) and the quit character (FS) are not treated specially. There is no echoing, and no replacement of one character for another; characters are a full 8 bits for both input and output (parity is up to the program).

CRMOD causes input carriage returns to be turned into new-lines; input of either CR or LF causes a settable string of up to eight characters to be output in place of the new-line character. A typical string would be "<CR><LF>".

CBREAK is a sort of half cooked mode. Programs can read each character as soon as typed, instead of waiting for a full line, but quit, DC1, DC3, and interrupt work; CRMOD, XTABS, ECHO, and parity work normally. On the other hand, there is no erase or kill, and no special treatment of the escape character, <^R>, <^K>, or EOT.

ARESM turns on auto resume, which causes any input character to resume the output if it has been suspended by a DC3.

XTABS turns on tab expansion, which causes tab characters to be expanded to the appropriate number of spaces on output. Tabs are fixed at every eight columns, starting at column nine.

The TANDEM field activates flow-control on input. When enabled, the IOP signals whatever is on the terminal line to stop sending when its buffers begin to get full, and signals it to start again when its buffer gets sufficiently empty.

CTSFLG turns on CTS flagging, which causes output to be stopped when CTS goes low and to be restarted when CTS goes high.

CHNGED is set when the large characteristic table has been changed and cleared when the data structure described above is set (since this resets the characteristic table to a standard state).

Several **ioctl** calls have the form:

```
#include <sgtty.h>
ioctl(fildes,code,arg)
struct sgttyb *arg;
```

The applicable codes are:

TIOPGETP - Fetch the parameters assoaiated with the terminal, and store in the pointed to sgttyb structure described earlier.

TIOPSETP - Set the parameters according to the pointed-to sgttyb structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOPSETN - Set the parameters but do not delay or flush input. Switching out of RAW or CBREAK mode may cause some garbage input.

TIOCGETP, TIOCSETP, and TIOCSETN perform the same functions as TIOPGETP, TIOPSETP, and TIOPSETN but interpret the flag word of the sgttyb structure as the old unix tty driver did. These are included for compatability with old object code.

With the following four codes the arg is ignored.

TIO - Turn on support for block mode communication (HSI) with an . This mode may not be turned off once it is enabled.

TIOCEXCL - Set "exclusive-use" mode: no further opens are permitted until the file has been closed.

TIOCNXCL - Turn of "exclusive-use" mode.

TIOCHPCL - When the file is closed for the last time, hang up the terminal.

TIOCNHPCL - When the file is closed for the last time, do not hang up the terminal.

TIOCFLUSH - All characters waiting in input or output queues are flushed.

The large terminal characteristic table may be set with the following calls:

TIOTSET - Fill in the table with the values at arg. Note that this also sets the CHNGED bit in the sg_flags field of the sgttyb structure to alert the user when s/he does a TIOPGETP that the table has been changed. This bit is cleared by a TIOPSET request. The TIOTSET command can change the function of any input character. Thus, all of the characters with special meaning like DC1, DC3, <^R>, <^K>, EOT, and <^C>, may be changed to another character or removed.

TIOTGET - Fetch the values in the table and store them at arg.

Full documentation on the large charcteristics table is not available at this printing.

Section 5
File Formats and Conventions

INTRODUCTION

This section describes some of the data structures, file structures, and file formats that are visible to the user. Information necessary for system accounting, such as the password file layout, can be found in this section. Disk addressing conventions used by the operating system are also described in this section.

A.OUT(5)

NAME

a.out – assembler and link editor output

SYNTAX

#include <a.out.h>

DESCRIPTION

A.out is the output file of the assembler *as (1)* and the link editor *ld (1).* Both programs make *a.out* executable if there were no errors and no unresolved external references. Layout information as given in the include file for the LSI 11/23 is:

```
/*
 * Header prepended to each a.out file.
 */
struct exec {
          long    a_magic;        /* magic number */
unsigned long    a_text;         /* size of text segment */
unsigned long    a_data;         /* size of initialized data */
unsigned long    a_bss;          /* size of uninitialized data */
unsigned long    a_syms;         /* size of symbol table */
unsigned long    a_entry;/* entry point */
unsigned long    a_trsize;       /* size of text relocation */
unsigned long    a_drsize;       /* size of data relocation */
};

#define OMAGIC         0407           /* old impure format */
#define NMAGIC         0410           /* read-only text */
#define ZMAGIC         0413           /* demand load format */

/*
 * Macros which take exec structures as arguments and tell whether
 * the file has a reasonable magic number or offsets to text|symbols|strings.
 */
#define N_BADMAG(x)    (((x).a_magic)!=OMAGIC && ((x).a_magic)!=NMAGIC\
          &&  ((x).a_magic= ZMAGIC)

#define N_TXTOFF(x)    ((x).a_magic==ZMAGIC ? 1024 : sizeof (struct exec))
#define N_SYMOFF(x)    (N_TXTOFF(x) + (x).a_text+(x).a_data +\
          (x).a_trsize+(x).a_drsize)
#define N_STROFF(x)    (N_SYMOFF(x) + (x).a_syms)


/*
 * Format of a relocation datum.
 */
struct relocation_info {
          int       r_address;        /* address which is relocated */
unsigned int    r_symbolnum:24,        /* local symbol ordinal */
                r_pcrel:1,      /* was relocated pc relative already */
                r_length:2,     /* 0=byte, 1=word, 2=long */
                r_extern:1,     /* does not include value of sym referenced */
                :4;             /* nothing, yet */
};

/*
 * Format of a symbol table entry; this file is included by <a.out.h>
 * and should be used if you aren't interested the a.out header
 * or relocation information.
 */
struct  nlist {
        union {
                char    *n_name;        /* for use when in-core */
                long    n_strx;         /* index into file string table */
        } n_un;
unsigned char   n_type;         /* type flag, i.e. N_TEXT etc; see below */
        char    n_other;/* unused */
        short   n_desc;         /* see <stab.h> */
unsigned long   n_value;        /* value of this symbol (or sdb offset) */
};
#define n_hash n_desc           /* used internally by ld */

/*
 * Simple values for n_type.
 */
#define N_UNDF          0x0             /* undefined */
#define N_ABS  0x2              /* absolute */
#define N_TEXT 0x4              /* text */
#define N_DATA          0x6             /* data */
#define N_BSS  0x8              /* bss */
#define N_COMM          0x12            /* common (internal to ld) */
#define N_FN   0x1f             /* file name symbol */

#define N_EXT  01               /* external bit, or'ed in */
#define N_TYPE 0x1e             /* mask for all the type bits */

/*
 * Sdb entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */
#define N_STAB          0xe0            /* if any of these bits set, a SDB entry */

/*
 * Format for namelist values.
 */
#define N_FORMAT        "%08x"
```

The file has four sections: a header, the program and data text, relocation information, and a symbol table (in that order). The last two may be empty if the program was loaded with the '−s' option of *ld* or if the symbols and relocation have been removed by *strip (1)*.

In the header the sizes of each section are given in bytes, but are even. The size of the header is not included in any of the other sizes.

When an *a.out* file is loaded into core for execution, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0 in the core image; the header is not loaded. If the magic number in the header is 0407(8), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. If the magic number is 0410, the data segment begins at the first 0 mod 8K byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. If the magic number is 411, the text segment is again pure, write-protected, and shared, and moreover instruction and data space are separated; the text and data segment both begin at location 0. If the magic number is 0405, the text segment is overlaid on an existing (0411 or 0405) text segment and the existing data segment is preserved.

The stack will occupy the highest possible locations in the core image: from 0177776(8) and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by *brk (2)*.

The start of the text segment in the file is 020(8); the start of the data segment is $020+S_t$ (the size of the text) the start of the relocation information is $020+S_t+S_d$; the start of the symbol table is $020+2(S_t+S_d)$ if the relocation information is present, $020+S_t+S_d$ if not.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file. Other flag values may occur if an assembly language program defines machine instructions.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader *ld* as the name of a common region whose size is indicated by the value of the symbol.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in core when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the 'relocation info stripped' flag in the header is on.

Bits 3-1 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

000	absolute number
002	reference to text segment
004	reference to initialized data
006	reference to uninitialized data (bss)
010	reference to undefined external symbol

Bit 0 of the relocation word indicates, if 1, that the reference is relative to the pc (e.g. 'clr x'); if 0, that the reference is to the actual symbol (e.g., 'clr *$x').

The remainder of the relocation word (bits 15-4) contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

SEE ALSO

as(1), ld(1), nm(1)

AR(5)

NAME

ar – archive (library) file format

SYNTAX

#include <ar.h>

DESCRIPTION

The archive command *ar* is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor *ld.*

A file produced by *ar* has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG "!<arch>0
#define SARMAG8

#define ARFMAG"'0

struct ar_hdr {
        char    ar_name[16];
        char    ar_date[12];
        char    ar_uid[6];
        char    ar_gid[6];
        char    ar_mode[8];
        char    ar_size[10];
        char    ar_fmag[2];
};
```

The name is a null-terminated string; the date is in the form of *time (2);* the user ID and group ID are numbers; the mode is a bit pattern per *chmod (2);* the size is counted in bytes.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

SEE ALSO

ar(1), ld(1), nm(1)

NOTES

Coding user and group IDs as characters is a botch.

BANPROTO(5)

NAME

/etc/banproto1 /etc/banproto2 – spooler banner page prototype files

DESCRIPTION

Prior to printing a file on a line printer the printer spooler prints a banner page, used to separate printouts and to identify the owner of the printout. The format of this banner page is controlled by the file **/etc/banproto***n*, where *n* is the number of the associated printer (1 or 2).

The prototype file contains text to be literally copied to the banner page, line formatting characters, and string substitution characters.

The only line formatting character is the uparrow (^).

^	causes the remainder of the line (excluding any other line formatting characters) to be printed in large type on the banner page.

String substitution characters can appear anywhere in a line and are replaced in the banner page by the corresponding string:

\nnn	A backslash immediately followed by one to three octal digits is replaced by the corresponding ASCII character. For example, \014 prints a formfeed character on the banner.
\u	is replaced by the username of the user who spooled the file to be printed.
\g	is replaced by the groupname of the user who spooled the file to be printed.
\d	is replaced by the current date, as output by the **ctime(3)** subroutine.
\newline	A backslash at the end of a line causes the end-of-line character to be ignored. This is used to break a long prototype file line into several shorter ones.
\^	is replaced by an uparrow character (^). This is used to escape the interpretation of the uparrow character.
\\\\	is replaced by a backslash (\\).
\p(*fieldspec*...)	is replaced by a field from the entry in /etc/passwd associated with the user who spooled the file to be printed.

Fieldspec consists of a field delimiter character (e.g. a colon) immediately followed by one decimal digit. The delimiter character is the character that delimits fields of the line or field; The decimal digit selects one field from that line or field — 0 selects the first field.

For example, a banner prototype file might contain the following line:

```
printed by \p(:4)
```

The fieldspec ":4" is read as "field[4] of the password file line". The fifth field (field[4]) of the password file is the comment field.

Some installations further divide the password comment field into subfields separated by semicolons. One such comment field is shown below:

```
Alfred Fetucchini;IH-1113;60-300
```

The first field is the user's real name; the second, his telephone extension; the third, his mail station. A banner prototype line to print his mail station would look like:

```
mail station \p(:4;2)
```

The fieldspec ":4" specifies the password file comment field, just as it did in the previous example. The fieldspec ";2" specifies the third semicolon-separated field of the password comment field (i.e. field[2], Alfred's mail station).

CHECKLIST(5)

NAME

checklist—list of normally mounted filesystems

DESCRIPTION

The "/etc/checklist" file contains, for each typically mounted filesystem, one line showing of the name of the *raw* special device that corresponds to that filesystem. The "/etc/checklist" file is used by several programs as a list of default filesystems to operate on, and is useful as a record of how your fixed disk drives are arranged (for systems that use 8503 Disk Expansion Units).

For example, consider the following "/etc/checklist" file:

```
/dev/rhd01
/dev/rhd23
```

This file describes one filesystem occupying fixed disk drives 0 and 1, and another filesystem occupying drives 2 and 3.

FILES

/etc/checklist

NOTES

The "/etc/checklist" file should be edited after you install any 8503 Disk Expansion Units or reconfigure your filesystems to reflect the new organization of the filesystems.

[This page intentionally left blank.]

CORE(5)

NAME

core – format of core image file

DESCRIPTION

TNIX writes out a core image of a terminated process when any of various errors occur. See *signal (2)* for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called 'core' and is written in the process's working directory (provided it can be; normal access controls apply).

The first 1024 bytes of the core image are a copy of the system's per-user data for the process, including the registers as they were at the time of the fault; see the system listings for the format of this area. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is write-protected and shared, it is not dumped; otherwise the entire address space is dumped.

In general the debugger *adb (1)* is sufficient to deal with core images.

SEE ALSO

adb(1), signal(2)

CVT(5)

NAME

cvt—structure of kernel CVT table

SYNTAX

#include sys/cvt.h

DESCRIPTION

The **cvt** data structure in the TNIX kernel contains the commonly-referenced, changeable parameters of the kernel. This allows you to reconfigure the way that the kernel uses its data space, without recompiling the kernel.

A pointer to the **cvt** structure is located at kernel address CVT_LOC. This value is defined in the <sys/cvt.h> include file. The pointer to the actual location of the **cvt** structure is the first address of the structure located at CVT_LOC.

Here is a listing of the <sys/cvt.h> include file:

```
#define CVT_LOC 050
struct cvt_ptr {                /* this is at location CVT_LOC */
        unsigned cp cvt;        /* pointer to real CVT */
        unsigned cp kend;       /* end of kernel instruction/data memory */
        unsigned cp end;        /* end of used memory */
};
struct cvt {                    /* this is at location CVT_LOC->cp_cvt */
        unsigned        c_nbuf;         /* number of buffers */
        unsigned        c_sbuf;         /* size of a buffer entry */
        struct buf      *c_buf;         /* start of buf table */
        struct buf      *c_ebuf;        /* end of buf table */
        char            *c_bstart;      /* start of actual buffers */
        unsigned        c_iocache;      /* number of i/o page buffers */
        unsigned        c_ninode;       /* number of inodes */
        unsigned        c_sinode;       /* size of an inode entry */
        struct inode    *c_inode;       /* start of inode table */
        struct inode    *c_einode;      /* end of inode table */
        unsigned        c_nfile;        /* number of file table entries */
        unsigned        c_sfile;        /* size of a file table entry */
        struct file     *c_file;        /* start of file table */
        struct file     *c_efile;       /* end of file table */
        unsigned        c_nproc;        /* number of process table entries */
        unsigned        c_sproc;        /* size of a process table entry */
        struct proc     *c_proc;        /* start of proc table */
        struct proc     *c_eproc;       /* end of proc table */
        unsigned        c_nmount;       /* number of mountable filesystems */
        unsigned        c_smount;       /* size of a mount table entry */
        struct mount    *c_mount;       /* start of mount table */
        struct mount    *c_emount;      /* end of mount table */
        unsigned        c_ntext;        /* number of text table entries */
        unsigned        c_stext;        /* size of a text table entry */
        struct text     *c_text;        /* start of text table */
        struct text     *c_etext;       /* end of text table */
        unsigned        c_nsmap;        /* number of swapmap table entries */
        unsigned        c_ssmap;        /* size of a swapmap table entry */
        struct map      *c_smap;        /* start of swapmap table */
        struct map      *c_esmap;       /* end of swapmap table */
        unsigned        c_ncmap;        /* number of coremap table entries */
        unsigned        c_scmap;        /* size of a coremap table entry */
        struct map      *c_cmap;        /* start of coremap table */
        struct map      *c_ecmap;       /* end of coremap table */
        dev_t           rootdev;        /* root device (major/minor) */
        dev_t           swapdev;        /* swap device (major/minor) */
        dev_t           pipedev;        /* pipe device (major/minor) */
        unsigned        nswap;          /* amount of swap space */
        daddr_t         swplo;          /* location of swap space */
        time_t          c_time;         /* system time of day */
        char            tz_name[4];     /* time zone name (3 char plus null) */
        char            tz_dayname[4]; /* daylight time zone name */
        short           tz_offgmt;      /* minutes west of GMT */
        short           tz_flag;        /* time zone flags */
                                        /* bit 0: 1 --> daylight time in effect (US rules) */
        unsigned        maxmem;         /* maximum user process memory (clicks) */
        unsigned        mem_start;      /* start of user memory (clicks) */
        unsigned        mem_end;        /* end of user memory (clicks) */
        char    *c_pmsg;        /* pointer into buffer */
        char    *c_msgbuf;      /* start of msg buffer */
        char    *c_emsgbuf;     /* end of msg buffer */
        struct IO_Info {        /* buffer cache profiling (bio.c) */
                long    nread;  /* # blocks read */
                long    nreada; /* # blocks read ahead */
                long    ncache; /* # blocks found in cache */
                long    nwrite; /* # blocks written */
                long    *bufcount; /* # free cache blocks (histogram) */
        } io_info;
};
```

FILES

/tnix—a copy of <cvt.h> is located here.
/dev/kmem—a copy of <cvt.h> is located here.

SEE ALSO

cvt*(8)*

DIR(5)

NAME

dir – format of directories

SYNTAX

#include <sys/dir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry see, *filsys (5).* The structure of a directory entry as given in the include file is:

```
/*      dir.h    4.2     81/02/19        */

#ifndef DIRSIZ
#define DIRSIZ  14
#endif
struct  direct
{
        ino_t   d_ino;
        char    d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. Therefore in both cases '..' has the same meaning as '.'.

SEE ALSO

filsys(5)

DUMP(5)

NAME

dump, ddate—incremental dump format

SYNTAX

#include <sys/types.h>
#include <sys/ino.h>
# include <dumprestor.h>

DESCRIPTION

Flexible disks used by *dump(8)* and **restor***(8)* contain:

- a header record,
- two groups of bit map records,
- a group of records describing directories, and
- a group of records describing files.

Here is the format of the header record and of the first record of each description, as given in the *<dumprestor.h>* include file:

```
#define NTREC          10
#define MLEN           16
#define MSIZ           4096
#define TS_TAPE        1
#define TS_INODE       2
#define TS_BITS        3
#define TS_ADDR        4
#define TS_END         5
#define TS_CLRI        6
#define MAGIC          (int)60011
#define CHECKSUM       (int)84446
struct  spcl {
        int     c_type;
        time_t  c_date;
        time_t  c_ddate;
        int     c_volume;
        daddr_t c_tapea;
        ino_t   c_inumber;
        int     c_magic;
        int     c_checksum;
        struct  dinode  c_dinode;
        int     c_count;
        char    c_addr[BSIZE];
} spcl;
struct  idates {
        char    id_name[16];
        char    id_incno;
        time_t  id_ddate;
};
#define DUMPOUTFMT      "%-16s %c %s"           /* for printf */
                                                /* name, incno, ctime(date) */
#define DUMPINFMT       "%16s %c %[ ^00"     /* inverse for scanf */
```

DEFINITIONS

Here are the definitions of the symbolic constants defined with the "# define" statement:

NTREC	the number of 512-byte records in a physical tape block.
MLEN	the number of bits in a bit map word.
MSIZ	the number of bit map words.

The *TS* entries are used in the *c_type* field to indicate what sort of header this is. The data types and their definitions are:

TS_TAPE	tape volume label
TS_INODE	a file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.
TS_BITS	a bit map follows. This bit map has a one bit for each inode that was dumped.
TS_ADDR	a subrecord of a file description. See *c_addr*, below.
TS_END	end of tape record.
TS_CLRI	a bit map follows. This bit map contains a "0" bit for all inodes that were empty on the file system when dumped.
MAGIC	all header records have this number in *c_magic*.
CHECKSUM	header records checksum to this value.

The fields of the header structure are as follows:

c_type	the type of the header.
c_date	the date the dump was taken.
c_ddate	the date the file system was dumped from.
c_volume	the current volume number of the dump.
c_tapea	the current number of this 512-byte record.
c_inumber	the number of the inode being dumped if this is of type *TS_INODE*.
c_magic	this contains the value *MAGIC* above, truncated as needed.
c_checksum	this contains whatever value is needed to make the record sum to *CHECKSUM*.
c_dinode	this is a copy of the inode as it appears on the file system; see **filsys***(5)*.
c_count	the count of characters in *c_addr*.
c_addr	an array of characters describing the blocks of the dumped file. A character is non-zero if the block associated with that character was present on the filesystem; otherwise, the character is 0. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, *TS_ADDR* records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a *TS_END* record and then the tapemark.

The structure *idates* describes an entry of the file */etc/ddate* where dump history is kept. The fields of the structure are:

id_name	the dumped filesystem is *'/dev/id_nam'*.
id_incno	the level number of the dump tape; see **dump***(8)*.
id_ddate	the date of the incremental dump in system format; see **types***(5)*.

FILES

/etc/ddate

SEE ALSO

dump*(5)*, **dump***(8)*, **dumpdir***(8)*, **filsys***(5)*, **restor***(8)*, **types***(5)*

[This page intentionally left blank.]

ENVIRON(5)

NAME

environ – user environment

SYNTAX

extern char **environ;

DESCRIPTION

An array of strings called the 'environment' is made available by **exec (2)** when a process begins. By convention these strings have the form 'name=value'. The following names are set automatically and are used by various commands:

PATH	The sequence of directory prefixes that **sh** applies in searching for a file known by an incomplete path name. The prefixes are separated by ':'. **Login (1)** sets PATH=:/bin:/usr/bin.
HOME	A user's login directory, set by **login (1)** from the password file **passwd (5)**.
PS1	**Shell** initial prompt string.
PS2	**Shell** secondary prompt string.
IFS	**Shell** field separators.

Further names may be placed in the environment by the **export** command and 'name=value' arguments in **sh (1)**, or by **exec (2)**. It is unwise to conflict with certain Shell variables that are frequently exported by '.profile' files:

TERM	The kind of terminal for which output is to be prepared.
	The 8540 associated with this user.
UP	The currently selected target microprocessor.
MAIL	The pathname of incoming mail. Used by **sh (1)**.

SEE ALSO

exec(2), sh(1), login(1)

FBR(5)

NAME

fbr – file backup and restore format

SYNTAX

```
#include <sys/types.h>
#include <fbr.h>
```

DESCRIPTION

Fbr saves and restores directories and files on a floppy disk archive, preserving aliases (multiple links to the same file). The archive consists of:

> Boot block - Block zero is reserved for a copy of a stand-alone boot program.
>
> Directory area - This area contains directory information for all files and directories on the archive. The first entry in this area is dedicated to the archive label.
>
> Data area - The data of all files on the archive is stored here.

The format of a directory entry (as given in the include file) follows. To increase the portability of archives, this structure is read and written in PDP-11 format regardless of what machine **fbr** is running on.

```
#define PATHLEN              106     /* space for path        */

struct fbrent
{
        char           fbr_path[PATHLEN]; /* pathname        */
        unsigned short fbr_mode;
        short          fbr_uid;  /* owner's userID      */
        short          fbr_gid;  /* owner's groupID     */
        off_t          fbr_size; /* size in bytes */
        time_t         fbr_acct; /* access date-time    */
        time_t         fbr_modt; /* modify date-time    */
        unsigned       fbr_fblk; /* first data block    */
        char           fbr_zero; /* unused              */
        char           fbr_chks; /* checksum            */
};
```

The path field is the pathname of the file when archived, less any redundant slashes or initial './'. It is null-terminated if less than PATHLEN bytes long. The mode, uid, gid, size, and access and modification date-times are in the same format as their corresponding i-node fields. As in the file system, an available entry has a mode of zero. The fblk field contains the block number of the first data block allocated to this file. The zero field is unused. The checksum field contains a number such that the sum of the first 127 bytes of the directory entry and the complement of the checksum is zero. Each file's data starts on a block boundary and occupies max(1,((size + 511) / 512) contiguous blocks. At least one archive block is allocated to each archived file or directory so that aliases (multiple links) can be properly recorded. Each directory archived is treated as an empty file (I.E. no directory's data is stored). All entries representing links to a given file are identical. In particular, the fblk field in each link's entry contains the same block number.

The archive label directory entry contains the following information:

path A comment about this archive.

mode
Set by **fbr** to a file readable/writeable/executable by all. Otherwise unused.

uid The userID of the user who created this archive.

gid The groupID of the same.

size The total number of bytes in the data area of the archive. This field in combination with the **fblk** field, records the size of both the directory area and the entire archive.

acct The date-time that this archive was created - *not* its access time.

modt
The date-time that this archive was last altered.

fblk The block number of the first archive block following the directory area.

checksum
(same as any other directory entry.)

SEE ALSO
fbr(1), stat(2).

FILSYS(5)

NAME

filsys, flblk, ino – format of file system volume

SYNTAX

#include <sys/types.h>
#include <sys/flbk.h>
#include <sys/filsys.h>
#include <sys/ino.h>

DESCRIPTION

Every file system storage volume (e.g. disk) has a common format for certain vital information. Every such volume is divided into a certain number of 512-byte blocks. Block 0 is unused and is available to contain a bootstrap program, pack label, or other information.

Block 1 is the *super block.* The layout of the super block as defined by the include file *<sys/filsys.h>* is:

```
/*
 * Structure of the super-block
 */
struct filsys {
        unsigned short s_isize;       /* size in blocks of i-list */
        daddr_t s_fsize;              /* size in blocks of entire volume */
        short   s_nfree;              /* number of addresses in s_free */
        daddr_t s_free[NICFREE];/* free block list */
        short   s_ninode;             /* number of i-nodes in s_inode */
        ino_t   s_inode[NICINOD];/* free i-node list */
        char    s_flock;              /* lock during free list manipulation */
        char    s_ilock;              /* lock during i-list manipulation */
        char    s_fmod;               /* super block modified flag */
        char    s_ronly;              /* mounted read-only flag */
        time_t  s_time;               /* last super block update */
};
```

S_isize is the address of the first block after the i-list, which starts just after the super-block, in block 2. Thus i-list is *s_isize–2* blocks long. *S_fsize* is the address of the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block addresses; if an 'impossible' block address is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The *s_free* array contains, in *s_free[1], ... , s_free[s_nfree–1],* up to NICFREE free block numbers. NICFREE is a configuration constant. *S_free[0]* is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in the include file <*sys/fblk.h*> is:

```
struct fblk
{
        int      df_nfree;
        daddr_t df_free[NICFREE];
};
```

The fields *df_nfree* and *df_free* in a free block are used exactly like *s_nfree* and *s_free* in the super block. To allocate a block: decrement *s_nfree*, and the new block number is *s_free[s_nfree]*. If the new block address is 0, there are no blocks left, so give an error. If *s_nfree* became 0, read the new block into *s_nfree* and *s_free*. To free a block, check if *s_nfree* is NICFREE; if so, copy *s_nfree* and the *s_free* array into it, write it out, and set *s_nfree* to 0. In any event set *s_free[s_nfree]* to the freed block's address and increment *s_nfree*.

S_ninode is the number of free i-numbers in the *s_inode* array. To allocate an i-node: if *s_ninode* is greater than 0, decrement it and return *s_inode[s_ninode]*. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the *s_inode* array, then try again. To free an i-node, provided *s_ninode* is less than NICINODE, place its number into *s_inode[s_ninode]* and increment *s_ninode*. If *s_ninode* is already NICINODE, don't bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

S_flock and *s_ilock* are flags maintained in the memory copy of the file system while it is mounted and their values on disk are immaterial. The value of *s_fmod* on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information. *S_ronly* is a write-protection indicator; its disk value is also immaterial.

S_time is the last time the super-block of the file system was changed. During a reboot, *s_time* of the super-block for the root file system is used to set the system's idea of the time.

The fields *s_tfree, s_tinode, s_fname, s_m, s_n* and *s_fpack* are not currently maintained.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-nodes are 64 bytes long, so 8 of them fit into a block. I-node 1 is used to group bad blocks found on the disk. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node as given in the include file <*sys/ino.h*> is:

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode
```

```
{
        unsigned short      di_mode;     /* mode and type of file */
        short  di_nlink;    /* number of links to file */
        short  di_uid;      /* owner's user id */
        short  di_gid;      /* owner's group id */
        off_t  di_size;     /* number of bytes in file */
        char   di_addr[40]; /* disk block addresses */
        time_t di_atime;    /* time last accessed */
        time_t di_mtime;    /* time last modified */
        time_t di_ctime;    /* time created */
};
#define INOPB 8             /* 8 inodes per block */
/*
 * the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

Di_mode tells the kind of file; it is encoded identically to the *st_mode field of stat (2). Di_nlink* is the number of directory entries (links) that refer to this i-node. *Di_uid* and *di_gid* are the owner's user and group IDs. *Size* is the number of bytes in the file. *Di_atime* and *di_mtime* are the times of last access and modification of the file contents (read, write or create) (see *times (2)); Di_ctime* records the time of last modification to the inode or to the file, and is used to determine whether it should be dumped.

Special files are recognized by their modes and not by i-number. A block-type special file is one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the *di_addr* field is occupied by the device code (see *types (5)).* The device codes of block and character special files overlap.

Disk addresses of plain files and directories are kept in the array *di_addr* packed into 3 bytes each. The first 10 addresses specify device blocks directly. The last 3 addresses are singly, doubly, and triply indirect and point to blocks of 128 block pointers. Pointers in indirect blocks have the type *daddr_t* (see *types (5)).*

For block *b* in a file to exist, it is not necessary that all blocks less than *b* exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

SEE ALSO

icheck(1), dcheck(1), dir(5), stat(2), types(5)

GROUP(5)

NAME

group – group file

DESCRIPTION

Group contains for each group the following information:

group name
encrypted password
numerical group ID
a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; Each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

newgrp(1), crypt(3), passwd(1), passwd(5)

GUIDE(5)

NAME

guide – guide file formats

DESCRIPTION

GUIDE is a shell program that executes the program **/usr/lib/guide/menu.** The *menu* program displays a menu or list of options that *GUIDE* will execute for you. Some of the options allow you to select specific tasks to perform; others allow you to select another menu.

The options displayed by the *menu* program are controlled by a file named *main.* Each directory in */usr/lib/guide* contains a file named *main,* allowing for as many different menus as there are directories in */usr/lib/guide.*

Two different kinds of files are used by *GUIDE:* the files named *main* within each directory in */usr/lib/guide;* and shell command files that execute the command you have selected. *GUIDE* uses the **TNIX** directory structures to group together logically related tasks under one menu. Generally, both the *main* file which provides the list of options to the *menu* program, and the shell command files that execute the selected options, are located in the same directory. For consistency, the names of the shell command files all begin with the letter "x".

DIRECTORY STRUCTURES

The directory structures used by *GUIDE* consist of directories in */usr/lib/guide* that contain a file called *main* and shell command files designated by the *main* file. The name of each shell command file begins with the letter "x". For example, the shell command file that executes the *pwd* command is named *xpwd.* The directory structure used by *GUIDE* includes:

```
/usr/lib/guide/
          menu    - menu driver program
          main    - top level menu
          xintro  - instructions on how to use GUIDE
          xprompt - determines user prompt level
          xshell  - shell command file that allows a
                    temporary escape to the shell

/usr/lib/guide/advan/      - directory containing the main file
                   and shell command files that provide
                   directory manipulation procedures.
             main   - contains a list of the directory
                   manipulation procedures and allows
                   a return to the top level menu or
                   to the file manipulation menu.
             xpwd   - executes a pwd command
             xcd    - executes a cd command
             xmkdir - executes a mkdir command
             xrmdir - executes a rmdir command

/usr/lib/guide/file/       - directory containing file manipulation
                   procedures.
             main   - contains a list of the file manipulation
                   procedures and allows a return to the
                   top level menu or to the directory
                   manipulation menu.
             xed    - executes a ed command
             xace   - executes a ace command
             xcp    - executes a cp command
             xcat   - executes a cat command
             xls    - executes a ls command
             xmv    - executes a mv command
             xrm    - executes a rm command
             xterm  - executes a term command
             xlpr   - executes a lpr command
          .
          .
          .

/usr/lib/guide/(dir_n)/    - user-defined directory containing
                   user-defined procedures and
                   shell command files.
             main    - user-defined menu
             xcmnd1  - user-defined shell file
             xcmnd2  - user-defined shell file
             xcmnd3  - user-defined shell file
```

To add a new menu, you create a new directory in the */usr/lib/guide* directory, then add the *main* file. The *main* file contains the names of the shell command files located in the newly created directory and the menu program to return to when you finish working in the newly created directory. Also, an existing *main* file must be modified to allow access to the newly created menu.

FILE FORMAT--MAIN

The basic format of the *main* file is:

```
main text block
end of main text block
:
description:new directory:shell program:program parameters
description:new directory:shell program:program parameters
description:new directory:shell program:program parameters
:
optional trailing text block
end of optional trailing text block
```

The fields in the *main* file are:

1. The main text describing the menu printed on your screen.
2. A line consisting of a single colon (:) separating the descriptive text from the commands available to you.
3. A description of a user-selectable command, displayed literally, with the following exceptions:
   - "\:" generates a colon character within the displayed description.
   - "\n" generates a newline followed by a tab within the displayed description, allowing for multiple-line descriptions for single commands.
   - "\t" generates a tab character within the displayed description.
   - "\\" generates a backslash (\) charater within the displayed description.
4. The name of a directory to change to before executing the command (current directory if no directory is specified).
5. The file name of a shell command file that executes the selected command.
   - If this field is empty, a new menu program is executed with arguments drawn from the *main* file located in the directory specified in the *new directory* field.
   - If the program name begins with a minus (–), the menu program does *not* resume executing when the shell program you have selected finishes executing.

- If the name consists of a single hyphen (–), the menu program reads the *main* file located in the directory specified by the *new directory* field.
- Otherwise, the shell program is run, after which the **menu** program regains control.

6. The parameters passed to the selected shell command file (*main* is the parameter passed to the **menu** program).
7. A line consisting of a single colon (:) that terminates the list of user-selectable commands.
8. A final block of descriptive text that is displayed after the available commands are listed.

Items 3-5 in the above list are separated from each other by a colon (:) -- each line contains one command.

A typical file format for a *main* file is:

```
          -- Directory Manipulation --

:
Top Level Menu:..:-:main
File Manipulation Menu\n:../file:-:main
Display the name of the current directory::xpwd:
Change the current directory::xcd:
Make a new directory\n::xmkdir:
Remove an empty directory::xrmdir:
Temporary escape to command language::../xshell:
:
```

When the *main* file shown above is used as the argument passed to the **menu** program, the following text is displayed on your screen:

```
                   -- Directory Manipulation --

        1) Top Level Menu
        2) File Manipulation Menu

        3) Display the name of the current directory
        4) Change the current directory
        5) Make a new directory

        6) Remove an empty directory
        7) Temporary escape to command language

Select by entering a number from 1 to 7:
```

If you enter a 3, the shell command file *xpwd* is executed.

SHELL COMMAND FILES

You should be familiar with the operation of the **TNIX** *shell* before attempting to write a shell command file for use with **GUIDE.** Refer to the *Shell* section of the *8560 MUSDU System Users Manual* for further information on shell procedures.

The shell command files executed by the **menu** program must begin with the line:

```
eval "'cat $GL'"
```

This command synchronizes the current directory and environment with your current directory and environment. Certain variables are made available to the executing shell program, specifically:

$GL	is the name of the temporary file containing a global environment consisting of the "$level" variable, the current directory, and the "$uP" variable, all stored as shell commands.
$level	is set equal to *a* or *b* depending on the selected prompting level.
$uP	is a global environment variable used by **8540.**
$IU	is set to the tty number of the 8540 or 8550, if you have selected it.

A simple example of a shell program that executes the **pwd** command is:

```
eval "'cat $GL'"
sh -cx "pwd"
case "$level" in
        a) echo -n "[press return to continue]";;
        b) echo -n "[continue]";;
esac
read x
```

The first line synchronizes **TNIX's** idea of the global environment with your selected environment. The second line executes a **pwd** command while displaying the command on your terminal. The *case* statement writes a message, based on the value of the global variable *$level,* on your screen. The read statement returns program control to the calling menu program when the RETURN key is pressed.

This shell command file is a simple example of a program that can be used with GUIDE. More complex programs are possible; their implementation depends on your knowledge of the shell command language. Examine the shell command files provided with the 8560 as needed for specific examples of the types of command files used by guide.

FILES

/usr/bin/guide - the top level shell program
/usr/lib/guide/menu - the menu file interpreter
/usr/lib/guide/main - the top level menu
/tmp/gl$$ - temporary file
/usr/lib/*/main - other menus
/usr/lib/*/x* - other shell programs

SEE ALSO

guide*(6)*
The *Shell* section of the *8560 MUSDU System Users Manual.*

MTAB(5)

NAME

mtab—mounted file system table

DESCRIPTION

/etc/mtab is a table of currently mounted filesystems. The **mount** command adds entries to this table; **umount** removes them.

Each entry is 64 bytes long. The first 32 are the null-padded name of the file the filesystem is mounted on (e.g. "/usr1"); the last 32 are the null-padded name of the special file involved (less any directory names: /dev/hd0 would be stored as "hd0").

FILES

/etc/mtab

SEE ALSO

mount*(2)*, **mount***(8)*

[This page intentionally left blank.]

PASSWD(5)

NAME

passwd – password file

DESCRIPTION

Passwd contains for each user the following information:

login name
encrypted password
numerical user ID
numerical group ID
a spare field for future use
initial working directory
program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The spare field can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

FILES

/etc/passwd

SEE ALSO

getpwent(3), login(1), crypt(3), passwd(1), group(5)

PLOT(5)

NAME

plot – graphics interface

DESCRIPTION

Files of this format are produced by routines described in *plot (3),* and are interpreted for various devices by commands described in *plot (1).* A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the 'current point' for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *plot (3).*

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes. See *plot (1).*

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

a arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c circle: The first four bytes give the center of the circle, the next two the radius.

e erase: Start another frame of output.

f linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are 'dotted,' 'solid,' 'longdashed,' 'shortdashed,' and 'dotdashed.' Effective only in *plot 4014* and *plot ver.*

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *plot (1).* The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face isn't square.

4014 space(0, 0, 3120, 3120);
ver space(0, 0, 2048, 2048);
300, 300s
space(0, 0, 4096, 4096);
450 space(0, 0, 4096, 4096);

SEE ALSO

plot(1), plot(3), graph(1)

TTYS(5)

NAME

ttys – terminal initialization data

DESCRIPTION

The *ttys* file is read by the *init* program and specifies which terminal special files are to have a process created for them which will allow people to log in. It contains one line per special file.

The first character of a line is either '0' or '1'; the former causes the line to be ignored, the latter causes it to be effective. The second character is used as an argument to **getty (8)**, which performs such tasks as baud-rate recognition, reading the login name, and calling **login**. The characters recognized by **getty(8)** are included with the description of **getty**.

FILES

/etc/ttys

SEE ALSO

init(8), getty(8), login(1)

UTMP(5)

NAME

utmp, wtmp – login records

SYNTAX

#include <utmp.h>

DESCRIPTION

The *utmp* file allows one to discover information about who is currently using **TNIX**. The file is a sequence of entries with the following structure declared in the include file:

```
struct utmp {
        char    ut_line[8];             /* tty name */
        char    ut_name[8];             /* user id */
        long    ut_time;        /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of *time (2).*

The *wtmp* file records all logins and logouts. Its format is exactly like *utmp* except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name '~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names '|' and '}' indicate the system-maintained time just before and just after a *date* command has changed the system's idea of the time.

Wtmp is maintained by *login (1)* and *init (8).* Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by *ac (1).*

FILES

/etc/utmp
/usr/adm/wtmp

SEE ALSO

login(1), init(8), who(1)

Section 6
Category C Software

INTRODUCTION

This Section describes the Category C software supplied by Tektronix. The three software packges which are described in this section and which comprise the class of software refered to as Category C software include:

1. Optional Text Processing Package 8560U01
2. Optional Native Programming Package 8560U02
3. Optional Auxiliary Utilities Package 8560U03

This section also describes software products such as GUIDE and ATO-BASM which are included as a part of the base software package.

NOTE

The products described in this section are provided by Tektronix as Category C software. Tektronix makes no warranty, expressed or implied, that this software is suitable for a specific purpose or that it performs any specific function correctly. Tektronix has determined that this software can be installed and will execute in the specified environment, but the suitability of the software and its correct operation are the customer's responsibility. Tektronix assumes no liability for any damage resulting from the use of this software, either directly or indirectly.

[This page intentionally left blank.]

ADB(6)

NAME

adb – debugger

SYNTAX

adb [–**w**] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of TNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is **a.out.** *Corfil* is assumed to be a core image file produced after executing *objfil* ; the default for *corfil* is **core.**

Requests to *adb* are read from the standard input and responses are to the standard output. If the –**w** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb* . *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

[*address*] [, *count*] [*command*] [;]

If *address* is present then *dot* is set to *address* . Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see ADDRESSES.

EXPRESSIONS

. The value of *dot* .

+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

integer
An octal number if *integer* begins with a 0; a hexadecimal number if preceded by # ; otherwise a decimal number.

integer . fraction
A 32 bit floating point number.

′cccc′ The ASCII value of up to 4 characters. \ may be used to escape a ′.

< **name**
The value of *name* , which is either a variable name or a register name. *Adb* maintains a number of variables (see VARIABLES) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil* . The register names are r0 ... **r5 sp pc ps** .

symbol
A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. **The value of the** *symbol* is taken from the symbol table in *objfil*. An initial _ or ~ will be prepended to *symbol* if needed.

_ symbol
In C, the 'true name' of an external symbol begins with _. It may be necessary to utter this name to disinguish it from internal or hidden variables of a program.

routine . name
The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*.

(exp)
The value of the expression *exp*.

Monadic operators

*** exp** The contents of the location addressed by *exp* in *corfil*.

@ exp The contents of the location addressed by *exp* in *objfil*.

– exp Integer negation.

~ exp Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1 + e2
Integer addition.

e1 – e2
Integer subtraction.

e1 * e2
Integer multiplication.

e1 % e2
Integer division.

e1 & e2
Bitwise conjunction.

e1 | e2
Bitwise disjunction.

e1 # e2
E1 rounded up to the next multiple of *e2*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '*'; see ADDRESSES for further details.)

? f Locations starting at *address* in *objfil* are printed according to the format *f* .

/ f Locations starting at *address* in *corfil* are printed according to the format *f* .

= f The value of *address* itself is printed in the styles indicated by the format *f* . (For i format '?' is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented temporarily by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

o 2
Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.

O 4
Print 4 bytes in octal.

q 2
Print in signed octal.

Q 4
Print long signed octal.

d 2
Print in decimal.

D 4
Print long decimal.

x 2
Print 2 bytes in hexadecimal.

X 4
Print 4 bytes in hexadecimal.

u 2
Print as an unsigned decimal number.

U 4
Print long unsigned decimal.

f 4
Print the 32 bit value as a floating point number.

F 8
Print double floating point.

b 1
Print the addressed byte in octal.

c 1
Print the addressed character.

C 1
Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.

s	n	Print the addressed characters until a zero character is reached.
S	n	Print a string using the @ escape convention. *n* is the length of the string including its zero terminator.
Y	4	Print 4 bytes in date format (see *ctime (3)).*
i	n	Print as PDP11 instructions. *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a	0	Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/	local or global data symbol
?	local or global text symbol
=	local or global absolute symbol

p	2	Print the addressed value in symbolic form using the same rules for symbol lookup as **a** .
t	0	When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.
r	0	Print a space.
n	0	Print a newline.
"..."	0	Print the enclosed string.
^		*Dot* is decremented by the current increment. Nothing is printed.
+		*Dot* is incremented by 1. Nothing is printed.
−		*Dot* is decremented by 1. Nothing is printed.

newline

If the previous command temporarily incremented *dot*, make the increment permanent. Repeat the previous command with a *count* of 1.

[?/] l value mask

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If L is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted then −1 is used.

[?/] w value ...

Write the 2-byte *value* into the addressed location. If the command is **W** , write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m b1 e1 f1[?/]
New values for *(b1, e1, f1)* are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the '?' or '/' is followed by '*' then the second segment (*b2* ,*e2* ,*f2*) of the mapping is changed. If the list is terminated by '?' or '/' then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, '/m?' will cause '/' to refer to *objfil* .)

> name
Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following '!'.

$ modifier
Miscellaneous commands. The available *modifiers* are:

< f	Read commands from the file *f* and return.
> f	Send output to the file *f*, which is created if it does not exist.
r	Print the general registers and the instruction addressed by **pc** . *Dot* is set to **pc**.
f	Print the floating registers in single or double length. If the floating point status of **ps** is set to double (0200 bit) then double length is used anyway.
b	Print all breakpoints and their associated counts and commands.
a	ALGOL 68 stack backtrace. If *address* is given then it is taken to be the address of the current frame (instead of **r4**). If *count* is given then only the first *count* frames are printed.
c	C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **r5**). If **C** is used then the names and (16 bit) values of all automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.
e	The names and values of external variables are printed.
w	Set the page width for output to *address* (default 80).
s	Set the limit for symbol matches to *address* (default 255).
o	All integers input are regarded as octal.
d	Reset integer input as described in EXPRESSIONS.
q	Exit from *adb* .
v	Print all non zero variables in octal.
m	Print the address map.

: modifier
Manage a subprocess. Available modifiers are:

b c	Set breakpoint at *address* . The breakpoint is executed *count −1* times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero then the breakpoint causes a stop.
d	Delete breakpoint at *address* .

r	Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.
c s	The subprocess is continued with signal *s* c *s*, see *signal (2).* If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for r .
s s	As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for r . In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
k	The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

0	The last value printed.
1	The last offset part of an instruction source.
2	The previous value of variable 1.

On entry the following are set from the system header in the *corfil* . If *corfil* does not appear to be a **core** file then these values are set from *objfil* .

b	The base address of the data segment.
d	The data segment size.
e	The entry point.
m	The 'magic' number (0405, 0407, 0410 or 0411).
s	The stack segment size.
t	The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples *(b1, e1, f1)* and *(b2, e2, f2)* and the *file address* corresponding to a written *address* is calculated as follows.

$$b1 \leqslant address < e1 \implies file\ address = address + f1 - b1,$$

otherwise,

$$b2 \leqslant address < e2 \implies file\ address = address + f2 - b2,$$

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a ? or / is followed by an * then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

So that *adb* may be used on large files all appropriate values are kept as signed 32 bit integers.

FILES

/dev/mem
/dev/swap
a.out
core

SEE ALSO

ptrace(2), a.out(5), core(5)

DIAGNOSTICS

'Adb' when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

NOTES

A breakpoint set at the entry point is not effective on initial entry to the program.
When single stepping, system calls do not count as an executed instruction.
Local variables whose names are the same as an external variable may foul up the accessing of the external.

[This page intentionally left blank.]

AR(6)

NAME

ar – archive and library maintainer

SYNTAX

ar key [posname] 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 **drqtpmx**, optionally concatenated with one or more of **vualbcl**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname* . Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print 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.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

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

v 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. When used with **t**, it gives a long listing of all information about the files. When used with **p** , it precedes each file with a name.

c Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally *ar* places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

FILES

/tmp/v*	temporaries

SEE ALSO

ld(1), ar(5), lorder(1)

NOTES

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

ARCV(6)

NAME

arcv – convert archives to new format

SYNTAX

arcv file ...

DESCRIPTION

Arcv converts archive files (see *ar (1), ar (5))* from 6th edition to 7th edition format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177555 at the start; new archives have 0177545.

FILES

/tmp/v*, temporary copy

SEE ALSO

ar(1), ar(5)

[This page intentionally left blank.]

AS(6)

NAME
as – assembler

SYNTAX
as [–] [**–o** objfile] file ...

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 *objfile;* if that is omitted, **a.out** is used. 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(1), nm(1), adb(1), a.out(5)
TNIX Assembler Manual by D. M. Ritchie

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

NOTES
Syntax errors can cause incorrect line numbers in following diagnostics.

[This page intentionally left blank.]

AT(6)

NAME
at – execute commands at a later time

SYNTAX
at time [day] [file]

DESCRIPTION
At squirrels away a copy of the named *file* (standard input default) to be used as input to *sh (1)* at a specified later time. A *cd (1)* command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the user and group ID of the creator of the copy file.

The *time* is 1 to 4 digits, with an optional following 'A', 'P', 'N' or 'M' for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional *day* is either (1) a month name followed by a day number, or (2) a day of the week; if the word 'week' follows invocation is moved seven days further off. Names of months and days may be recognizably truncated. Examples of legitimate commands are

```
at 8am jan 24
at 1530 fr week
```

At programs are executed by periodic execution of the command */usr/lib/atrun* from *cron (8)*. The granularity of *at* depends upon how often *atrun* is executed.

Standard output or error output is lost unless redirected.

FILES
/usr/spool/at/yy.ddd.hhhh.uu
activity to be performed at hour *hhhh* of year day *ddd* of year *yy*. *uu* is a unique number.
/usr/spool/at/lasttimedone contains *hhhh* for last hour of activity.
/usr/spool/at/past directory of activities now in progress
/usr/lib/atrun program that executes activities that are due
pwd(1)

SEE ALSO
calendar(1), cron(8)

DIAGNOSTICS
Complains about various syntax errors and times out of range.

NOTES
Due to the granularity of the execution of */usr/lib/atrun,* there may be bugs in scheduling things almost exactly 24 hours into the future.

[This page intentionally left blank.]

ATOBASM(6)

NAME

atobasm—A series to B series conversion program for non-microprocessor-specific source code.

SYNTAX

atobasm [**–o** *newfile*] [**–n**] [**–m**] *infile*

DESCRIPTION

Atobasm converts non-microprocessor-specific A series assembler source code into B series assembler source code where possible, generating conversion messages or error messages where manual intervention is necessary.

OPTIONS

–m	Merge the message file with the output source.
–n	Turn off the conversion for reserved words and labels.
–o	Output the B series source to newfile; otherwise, sends the B series source to the standard output.

SOURCE DIFFERENCES

Here is a list of source differences between the A series and B series assemblers:

1. Text substitution delimiter

   A series: delimiter = '

   B series: delimiter = "

2. String delimiter

   A series: delimiter = "

   B series: delimiter = '

   Instances of ^' and ^" will be kept in their original (correct) form.

3. TRM list option

   A series: available

   B series: replaced by the LINE directive

4. Numeric operand set to a string

   A series: Numeric converted to a string; truncated on right.

   B series: Numeric treated as a literal; truncated on left.

   Example of A series: STRING **A,B***(3)*

```
A    SET     6 ;  sets A to "000006"
B    SET    -3 ;  sets B to "-00"
B    SET  1234 ;  sets B to "001"
```

   Example of B series: STRING **A,B***(3)*

```
A    SET     6 ;  sets A to "6"
B    SET    -3 ;  sets B to "-3"
B    SET  1234 ;  sets B to "123" and a truncation error
```

5. Labels allowed on the ELSE, ENDIF, ENDR, EXITM, LIST, MACRO, NAME, NOLIST, PAGE, REPEAT, SPACE, STITLE, STRING, TITLE, and WARNING instructions

   A series: labels are permitted

   B series: labels are not permitted

6. Reserved words—ADDRESS, ALIGN, BITS, CLASS, ELSEIF, EXITR, FLOAT, LONG, LINE, MACLIB, STOPS, STRINGOF, TIMES, XREF

   A series: not used as reserved words

   B series: reserved words

7. Variable name length

   A series: only the first 8 characters of a variable name are recognized

   B series: the first 16 characters of a variable name are recognized

   Variable name length can be a problem if you define a variable named "VERYLONGLABEL" in an A series program, but later refer to it within the same program as "VERYLONG". Since the B series assembler would treat them as two distinct variables, "VERYLONG" would be undefined in the B series assembler.

EXAMPLES

Here are some typical command invocations:

```
$ atobasm oldfile <CR>
```

This uses the file "oldfile" as input and produces a file on standard output with the messages and source code merged.

```
$ atobasm -o newfile oldfile <CR>
```

The file "oldfile" will be used as input. The source code will be output to "newfile" and the messages will be sent to standard output.

```
$ atobasm -o newfile -m oldfile <CR>
```

or

```
$ atobasm oldfile >newfile <CR>
```

The file "oldfile" will be used as input. The messages and source code will be merged into "newfile"

```
$ atobasm oldfile 2>errfile <CR>
```

The file "oldfile" will be used as input. The conversion messages and source code merged will be sent to standard output. The error messages will be output into "errfile".

CONVERSION MESSAGES

The following conversion messages may be generated:

```
atobasm:   line nn Invalid A series syntax; remainder of line skipped
atobasm:   line nn Include file must be converted also, check file name
atobasm:   line nn <symbol> changed to <symbol>
atobasm:   Substituted values may be keywords on the following lines:
           nn, nn, nn, . . .
atobasm:   String delimiter ' is converted to " on the following lines:
           nn, nn, nn, . . .
```

atobasm:	String delimiter " is converted to ' on the following lines: nn, nn, nn, . . .
atobasm:	Numberic operand conversion done on the following lines: nn, nn, nn, . . .
atobasm:	Labels conversion done on the following lines: nn, nn, nn, . . .
atobasm:	TRM conversion done on the following lines: nn, nn, nn, . . .
atobasm:	String delimiter " is converted to ^ " on the following lines: nn, nn, nn, . . .

nn is the line number of the B series assembler output file.

ERROR MESSAGES

The following error messages may be generated:

atobasm:	file read error
atobasm:	file write error
atobasm:	memory overflow
atobasm:	cannot open file
atobasm:	input <file> is output
atobasm:	usage: atobasm [–o outfile] [–n] [–m] infile

NOTES

Atobasm works for all legal A series assembler programs, subject to the following constraints:

a. **Atobasm** will:
   - change the text substitution delimiter to its equivalent B series form ("); and will
   - truncate the variable name within the text substitution delimiters to 8 characters.

b. **Atobasm** will not:
   - convert the variable name within the text substitution delimiters if it is a B series reserved word;
   - convert a B series reserved word when it is composed of substituted text and leading and/or trailing text; or
   - truncate a variable to 8 characters when it is composed of substituted text and leading and/or trailing text.

c. **Atobasm** is restricted to source code that is not microprocessor-specific. For microprocessor-specific differences between the A series and B series assemblers, refer to the appropriate Assembler Specifics supplement to your 8500 Modular MDL Series B Series Assembler Users Manual.

d. **Atobasm** will not check TRM options placed in the operand field of a LIST or NOLIST statement via text substitution. You must manually convert the statement.

e. A numeric value set to a string variable may use an expression or constant (decimal, binary, octal, or hexadecimal) format, but must yield a valid A series constant (a value between –32768 and 32767).

f. The string delimiter in the comment field is converted to ^ ".
g. Variable names in the comment field are not truncated to 8 characters.
h. You must convert *include* files before you assemble the program. You must convert library files before link time.

FILES

###.atob.tmp—temporary file
###.atob.tpmesfile—temporary file

AWK(6)

NAME

awk – pattern scanning and processing language

SYNTAX

awk [**−F c**] [prog] [file] ...

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog,* or in a file specified as −f *file*.

Files are read in order; if there are no files, the standard input is read. The file name '−' means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS.) The fields are denoted $1, $2, ... ; $0 refers to the entire line.

A pattern-action statement has the form

```
pattern { action }
```

A missing { action } means print the line; a missing pattern always matches.

An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next     # skip remaining patterns on this input line
exit     # skip the rest of the input
```

Statements are terminated by semicolons, newlines or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The C operators ++, −−, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted "...".

The *print* statement prints its arguments on the standard output (or on a file if >*file* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format (see *printf (3)).*

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp, log, sqrt,* and *int*. The last truncates its argument to an integer. *substr(s, m, n)* returns the *n -character* substring of *s* that begins at position *m*. The function *sprintf(fmt, expr, expr, ...)* formats the expressions according to the *printf (3)* format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep*. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

where a relop is any of the six relational operators in C, and a matchop is either ~ (for contains) or !~ (for does not contain). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with

```
BEGIN { FS = "c" }
```

or by using the **−F c** option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default newline); and OFMT, the output format for numbers (default %.6g).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
            { s += $1 }
    END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
    { for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
    /start/, /stop/
```

Print all lines whose first field is different from previous one:

```
    $1 != prev { print; prev = $1 }
```

SEE ALSO

lex(1), sed(1)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, *Awk – a pattern scanning and processing language*

NOTES

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate "" to it.

[This page intentionally left blank.]

BAS(6)

NAME

bas – basic

SYNTAX

bas [file]

DESCRIPTION

Bas is a dialect of Basic. If a file argument is provided, the file is used for input before the terminal 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. Interrupts suspend computation.

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.

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

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

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,

& | & (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
/bin/ed	for **edit**

DIAGNOSTICS

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

NOTES

Has been known to give core images.
Catches interrupts even when they are turned off.

BC(6)

NAME

bc – arbitrary-precision arithmetic language

SYNTAX

bc [**−c**] [**−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 an arbitrary precision math library. The syntax for *bc* programs is as follows; L means letter a-z, E means expression, S means statement.

Comments

```
are enclosed in /* and */.
```

Names

```
simple variables: L
array elements: L [ E ]
The words 'ibase', 'obase', and 'scale'
```

Other operands

```
arbitrarily long numbers with optional sign and decimal point.
( E )
sqrt ( E )
length ( E )    number of significant decimal digits
scale ( E )     number of digits right of decimal point
L ( E , ... , E )
```

Operators

```
+  −  *  /  %  ^  (% is remainder; ^ is power)
++  −−            (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

```
define L ( L ,..., L ) {
        auto L, ... , L
        S; ... S
        return ( E )
}
```

Functions in −l math library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

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 in the manner of *dc (1).* Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. 'Auto' variables are pushed down during function calls. 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.

Bc is actually a preprocessor for *dc (1),* which it invokes automatically, unless the −c (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

FILES

/usr/lib/lib.b	mathematical library
dc(1)	desk calculator proper

SEE ALSO

dc(1)
L. L. Cherry and R. Morris, *BC – An arbitrary precision desk-calculator language*

NOTES

No &&, ||, or ! operators.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

[This page intentionally left blank.]

CAL(6)

NAME

cal – print calendar

SYNTAX

cal [month] year

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

NOTES

The year is always considered to start in January even though this is historically naive.
Beware that 'cal 78' refers to the early Christian era, not the 20th century.

[This page intentionally left blank.]

CALENDAR(6)

NAME

calendar – reminder service

SYNTAX

calendar [–]

DESCRIPTION

Calendar consults the file 'calendar' in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as 'Dec. 7,' 'december 7,' '12/7,' etc., are recognized, but not '7 December' or '7/12'. On weekends 'tomorrow' extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file 'calendar' in his login directory and sends him any positive results by *mail (1).* Normally this is done daily in the wee hours under control of *cron (8).*

FILES

calendar
/usr/lib/calendar to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
egrep, sed, mail subprocesses

SEE ALSO

at(1), cron(8), mail(1)

NOTES

Your calendar must be public information for you to get reminder service.
Calendar's extended idea of 'tomorrow' doesn't account for holidays.

[This page intentionally left blank.]

CB(6)

NAME

cb – C program beautifier

SYNTAX

cb

DESCRIPTION

Cb places a copy of the C program from the standard input on the standard output with spacing and indentation that displays the structure of the program.

[This page intentionally left blank.]

CC(6)

NAME

cc, pcc – C compiler

SYNTAX

cc [option] ... file ...

pcc [option] ... file ...

DESCRIPTION

Cc is the TNIX C compiler. It accepts several 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.

In the same way, arguments whose names end with '.s' are taken to be assembly source programs and are assembled, producing a '.o' file.

The following options are interpreted by *cc* . See *id (1)* for load-time options.

–q Turn off the printing of the name of each pass of the compiler as it is called by *cc (1).* The default mode is to print them. This is sometimes called noisy and quiet mode, depending on the the value of this switch.

–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 *monitor (3)* 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 (1).*

–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 and place the result for each '.c' file in a corresponding '.i' file and has no '#' lines in it.

–E Run only the macro preprocessor and send the result to the standard output. The output is intended for compiler debugging; it is unacceptable as input to *cc* .

–o output

Name the final output file *output* . If this option is used the file 'a.out' will be left undisturbed.

−D name=def
−D name
Define the *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as 1.

−U name
Remove any initial definition of *name* .

−I dir '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **−I** options, then in directories on a standard list.

−B string
Find substitute compiler passes in the files named *string* with the suffixes cpp, c0, c1 and c2. If *string* is empty, use a standard backup version.

−t [p012]
Find only the designated compiler passes in the files whose names are constructed by a **−B** option. In the absence of a **−B** option, the *string* is taken to be '/usr/c/'.

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

The major purpose of the 'portable C compiler', *pcc,* is to serve as a model on which to base other compilers. *Pcc* does not support options **−f** , **−E** , **−B** , and **−t** . It provides, in addition to the language of *cc,* unsigned char type data and initialized bit fields.

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporaries for *cc*
/lib/cpp	preprocessor
/lib/c[01]	compiler for *cc*
/lib/bc0	large pass 0.
/usr/c/oc[012]	backup compiler for *cc*
/usr/c/ocpp	backup preprocessor
/lib/fc[01]	floating-point compiler
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/fcrt0.o	startoff for floating-point interpretation
/lib/libc.a	standard library, see *intro (3)*
/usr/include	standard directory for '#include' files
/tmp/pc*	temporaries for *pcc*
/usr/lib/ccom	compiler for *pcc*

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, 1978
D. M. Ritchie, *C Reference Manual*
monitor(3), prof(1), adb(1), ld(1)

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, *as (1),* in particular 'm', which means a multiply-defined external symbol (function or data).

NOTES

Pcc is little tried on the PDP11; specialized code generated for that machine has not been well shaken down. The −O optimizer was designed to work with *cc ;* its use with *pcc* is suspect.

[This page intentionally left blank.]

COL(6)

NAME

col – filter reverse line feeds

SYNTAX

col [–bfx]

DESCRIPTION

Col reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ESC-7 in ASCII) and by forward and reverse half line feeds (ESC-9 and ESC-8). *Col* is particularly useful for filtering multicolumn output made with the '.rt' command of *nroff* and output resulting from use of the *tbl (1)* preprocessor.

Although *col* accepts half line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full line boundary. This treatment can be suppressed by the **–f** (fine) option; in this case the output from *col* may contain forward half line feeds (ESC-9), but will still never contain either kind of reverse line motion.

If the **–b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

The control characters SO (ASCII code 017), and SI (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

Col normally converts white space to tabs to shorten printing time. If the **–x** option is given, this conversion is suppressed.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by one of 789, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

SEE ALSO

troff(1), tbl(1), greek(1)

NOTES

Can't back up more than 128 lines.
No more than 800 characters, including backspaces, on a line.

[This page intentionally left blank.]

CRYPT(6)

NAME

crypt – encode/decode

SYNTAX

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; 'sneak paths' by which keys or cleartext can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps (1)* or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of *crypt.*

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(8)

NOTES

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, Tektronix, Inc. assumes no responsibility for their use by the recipient. Further, Tektronix, Inc. assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

[This page intentionally left blank.]

DC(6)

NAME

dc – desk calculator

SYNTAX

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. **P** interprets the top of the stack as an ascii string, removes it, and prints it.

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.

X — replaces the number on the top of the stack with its scale factor.

[...] — 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 TNIX 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. **I** pushes the input base on the top of the stack.

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

O pushes the output base on the top of the stack.

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.

Z replaces the number on the top of the stack with its length.

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

; : are used by *bc* for array operations.

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

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

SEE ALSO

bc(1), 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 is unimplemented' where x is an octal number.
'stack empty' for not enough elements on the stack to do what was asked.
'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.

DD(6)

NAME

dd – convert and copy a file

SYNTAX

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= n
input block size *n* bytes (default 512)
obs= n
output block size (default 512)
bs= n 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
files= n
copy *n* files from (tape) input
seek= n
seek *n* records from beginning of output file before copying
count=n
copy only *n* input records
conv=ascii
convert EBCDIC to ASCII
ebcdic
convert ASCII to EBCDIC
ibm
slightly different map of 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; 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.

To skip over a file before copying from magnetic tape do

```
(dd of=/dev/null; dd of=x) </dev/rmt0
```

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

f+p records in(out): numbers of full and partial records read(written)

NOTES

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The 'ibm' conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

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

DEROFF(6)

NAME

deroff – remove nroff, troff, tbl and eqn constructs

SYNTAX

deroff [**−w**] file ...

DESCRIPTION

Deroff reads each file in sequence and removes all *nroff* and *troff* command lines, backslash constructions, macro definitions, *eqn* constructs (between '.EQ' and '.EN' lines or between delimiters), and table descriptions and writes the remainder on the standard output. *Deroff* follows chains of included files ('.so' and '.nx' commands); if a file has already been included, a '.so' is ignored and a '.nx' terminates execution. If no input file is given, *deroff* reads from the standard input file.

If the **−w** flag is given, the output is a word list, one 'word' (string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed) per line, and all other characters ignored. Otherwise, the output follows the original, with the deletions mentioned above.

SEE ALSO

troff(1), eqn(1), tbl(1)

NOTES

Deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

[This page intentionally left blank.]

DIFF3(6)

NAME

diff3 – 3-way differential file comparison

SYNTAX

diff3 [**−ex3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====
: all three files differ

====1
: *file1 is different*

====2
: *file2 is different*

====3
: *file3 is different*

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f : n1 a
: Text is to be appended after line number *n1* in file *f,* where *f* = 1, 2, or 3.

f : n1 , n2 c
: Text is to be changed in the range line *n1* to line *n2* . If *n1* = *n2,* the range may be abbreviated to *n1* .

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the **−e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3, i.e* . the changes that normally would be flagged ==== and ====3. Option **−x** (**−3**) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to 'file1'.

```
(cat script; echo '1,$p') | ed − file1
```

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(1)

NOTES

Text lines that consist of a single '.' will defeat **−e**.
Files longer than 64K bytes won't work.

[This page intentionally left blank.]

EHEX(6)

NAME

ehex—multi-vendor interface program

SYNTAX

ehex [–m] [–s] [–i] [–**n** *number*] *file*

OPTIONS

–i	Converts **file** to Intel hexadecimal load module format.
–m	Converts **file** to Motorola S-records.
–s	Converts **file** to Standard Tekhex.
(default)	If no modifier is specified, the file is converted to Extended Tekhex format.
–n number	Specifies the number of characters (including EOF) per output record. *Number* can be a maximum of 72 characters (30 data bytes) for standard Tekhex; 256 characters for Intel hexadecimal load module format, Motorola S-record, and Extended Tekhex formats. If you do not specify *number*, the default value is 72 for standard Tekhex, and 80 for all other formats. The default value for *number* includes the end-of-line character.
file	Contains the Tektronix binary load module to convert.

DESCRIPTION

The **ehex** command converts a file in Tektronix binary load module format to the specified format.

All hexadecimal output is written to standard output.

The Intel and Motorola formats allow addresses larger than 16 bits, as does extended TEKHEX.

Program symbols are limited to 16 characters. Symbol values (including address-space bytes) are limited to 32 bits, and the total number of sections may not exceed 300.

EXTENDED TEKHEX

The distinction between "code" and "data" addresses (available in Extended Tekhex) is not implemented. Any address-space byte included with an address is passed straight through as part of the address (used with chips such as the Z8001 and MC68000).

EXAMPLES

```
$ ehex -i prog.load <CR>
```

Converts the file "prog.load" from Tektronix binary load module format to Intel hexadecimal load module format.

[This page intentionally left blank.]

EQN/NEQN/CHECKEQN(6)

NAME

eqn, neqn, checkeqn – typeset mathematics

SYNTAX

eqn [**−d** xy] [**−p** n] [**−s** n] [**−f** n] [file] ...
checkeq [file] ...

DESCRIPTION

Eqn is a troff(1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, *neqn* on terminals. Usage is almost always

```
eqn file ... | troff
neqn file ... | nroff
```

If no files are specified, these programs 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, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **−d xy** or (more commonly) with 'delim *xy* ' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ˜ represents a full space in the output, circumflex ˆ half as much.

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* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 +bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim\limits_{n \to \infty} \sum\limits_0^n x_i$ 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. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

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.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~=~ n under* is $\bar{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \overleftrightarrow{y}$.

Sizes and font can be changed with **size** *n* or **size** ± **n, roman**, **italic**, **bold**, and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s n** and **−f n.**

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **−p n.**

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**: *define thing % replacement %* defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement.*

Keywords like *sum*	$(\sum)$
int	$(\int)$
inf	(∞)
and shorthands like >=	$(\geqslant)$
−>	$(\rightarrow)$,
and !=	$(\neq)$

@

are recognized. Greek letters are spelled out in the desired case, as in *alpha* or *GAMMA.* Mathematical words like sin, cos, log are made Roman automatically. *Troff (1)* four-character escapes like \(bs (@) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

SEE ALSO

troff(1), tbl(1), ms(7), eqnchar(7)

B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics—User's Guide*

J. F. Ossanna, *NROFF/TROFF User's Manual*

NOTES

To embolden digits, parens, etc., it is necessary to quote them, as in 'bold "12.3"'.

[This page intentionally left blank.]

F77(6)

NAME

f77 – Fortran 77 compiler

SYNTAX

f77 [option] ... file ...

DESCRIPTION

F77 is the TNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by f77.

In the same way, arguments whose names end with '.c' or '.s' are taken to be C or assembly source programs and are compiled or assembled, producing a '.o' file.

The following options have the same meaning as in *cc (1)*. See *ld (1)* for load-time options.

−c Suppress loading and produce '.o' files for each source file.

−p Prepare object files for profiling, see *prof (1)*.

−O Invoke an object-code optimizer.

−S Compile the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

−f Use a floating point interpreter (for PDP11's that lack 11/70-style floating point).

−o output
Name the final output file *output* instead of 'a.out'.

The following options are peculiar to *f77*.

−onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

−u Make the default type of a variable 'undefined' rather than using the default Fortran rules.

−C Compile code to check that subscripts are within declared array bounds.

−w Suppress all warning messages. If the option is '−w66', only Fortran 66 compatibility warnings are suppressed.

−F Apply EFL and Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to '.f', but do not compile.

−m	Apply the M4 preprocessor to each '.r' or '.e' file before transforming it with the Ratfor or EFL preprocessor.
−E x	Use the string *x* as an EFL option in processing '.e' files.
−R x	Use the string *x* as a Ratfor option in processing '.r' files.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-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.[fresc]	input file
file.o	object file
a.out	loaded output
/lib/f77pass1	compiler
/lib/c1	pass 2
/lib/c2	optional optimizer
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library, see section 3

SEE ALSO

S. I. Feldman, P. J. Weinberger, *A Portable Fortran 77 Compiler*
prof(1), cc(1), ld(1)

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

NOTES

The Fortran 66 subset of the language has been exercised extensively; the newer features have not.

FACTOR/PRIMES(6)

NAME

factor, primes – factor a number, generate large primes

SYNTAX

factor [number]

primes

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about 7.2×10^{16}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to $\sqrt{n}$ and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 10^{14} on a PDP11.

When *primes* is invoked, it waits for a number to be typed in. If you type in a positive number less than 2^{56} it will print all primes greater than or equal to this number.

DIAGNOSTICS

'Ouch.' for input out of range or for garbage input.

[This page intentionally left blank.]

GRAPH(6)

NAME

graph – draw a graph

SYNTAX

graph [option] ...

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *plot (1)* filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "...", in which case they may be empty or contain blanks and numbers; labels never contain newlines.

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

−a	Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).
−b	Break (disconnect) the graph after each label in the input.
−c	Character string given by next argument is default label for each point.
−g	Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
−l	Next argument is label for graph.
−m	Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.
−s	Save screen, don't erase before plotting.
−x [l]	If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.
−y [l]	Similarly for *y* .
−h	Next argument is fraction of space for height.
−w	Similarly for width.
−r	Next argument is fraction of space to move right before plotting.
−u	Similarly to move up before plotting.
−t	Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1), plot(1)

NOTES

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

GUIDE(6)

NAME

guide – menu-driven software development tool

SYNTAX

guide

DESCRIPTION

GUIDE is a software development tool designed to give you immediate command of the entire range of capabilities built into the 8560 and associated workstations. The main feature of *GUIDE* is a menu-driven front-end, a program that offers you a menu consisting of a list of options that *GUIDE* executes for you. Some of the options allow you to select specific tasks to be performed; others allow you to select another menu. When you select an opotion, the menu program requests additional information where necessary, then executes the task.

GUIDE issues the appropriate **TNIX** commands to execute the task selected by you. Each **TNIX** command is displayed on your screen preceded by a "+", to help you learn the actual **TNIX** commands required.

The menu driver approach makes *GUIDE* a very powerful tool--it can easily be extended and modified to adapt to the individual needs of specific programming environments. Knowledge of the *Shell* command language is required for the modification of *GUIDE*.

USING GUIDE

GUIDE is a shell program that executes the program **/usr/lib/guide/menu**. To use *GUIDE*, enter the command line:

```
$ guide
```

GUIDE then prompts you through the execution of each command or procedure desired until you exit *GUIDE*.

FILES

/usr/bin/guide - the top level shell program.
/usr/lib/guide/menu - the menu file interpreter
/usr/lib/guide/main - the top level menu
/tmp/gl$$ - temporary file
/usr/lib/*/main - other menus
/usr/lib/*/x* - other shell programs

SEE ALSO

guide(*5*)
The *Shell* section of the *8560 MUSDU System Users Manual.*
The *Operating Procedures* section of the *8560 MUSDU System Users Manual.*

[This page intentionally left blank.]

IOSTAT(6)

NAME

iostat – report I/O statistics

SYNTAX

iostat [option] ... [interval [count]]

DESCRIPTION

Iostat delves into the system and reports certain statistics kept about input-output activity. Information is kept about up to three different disks (RF, RK, RP) and about typewriters. For each disk, IO completions and number of words transferred are counted; for typewriters collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. The tally goes into one of four categories, depending on whether the system is executing in user mode, in 'nice' (background) user mode, in system mode, or idle. From all these numbers and from the known transfer rates of the devices it is possible to determine information such as the degree of IO overlap and average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

With no option argument *iostat* reports for each disk the number of transfers per minute, the milliseconds per average seek, and the milliseconds per data transfer exclusive of seek time. It also gives the percentage of time the system has spend in each of the four categories mentioned above.

The following options are available:

–t Report the number of characters of terminal IO per second as well.

–i Report the percentage of time spend in each of the four categories mentioned above, the percentage of time each disk was active (seeking or transferring), the percentage of time any disk was active, and the percentage of time spent in 'IO wait:' idle, but with a disk active.

–s Report the raw timing information: 32 numbers indicating the percentage of time spent in each of the possible configurations of 4 system states and 8 IO states (3 disks each active or not).

–b Report on the usage of IO buffers.

FILES

/dev/mem, /unix

[This page intentionally left blank.]

JOIN(6)

NAME

join – relational database operator

SYNTAX

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2* . If *file1* is '–', the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1* , then the rest of the line from *file2* .

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

–a n	In addition to the normal output, produce a line for each unpairable line in file *n* , where *n* is 1 or 2.
–e s	Replace empty output fields by string *s* .
–j n m	Join on the *m th* field of file *n* . If *n* is missing, use the *m th* field in each file.
–o **list**	Each output line comprises the fields specifed in *list* , each element of which has the form *n . m* , where *n* is a file number and *m* is a field number.
–t c	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

SEE ALSO

sort(1), comm(1), awk(1)

NOTES

With default field separation, the collating sequence is that of *sort –b ;* with **–t** , the sequence is that of a plain sort.

The conventions of *join, sort, comm, uniq, look* and *awk (1)* are wildly incongruous.

[This page intentionally left blank.]

LD(6)

NAME

ld – loader

SYNTAX

ld [option] file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* 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, and the library has not been processed by *ranlib (1),* the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named '__.SYMDEF', then it is understood to be a dictionary for the library such as produced by *ranlib ;* the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for –l , they should appear before the file names.

–s 'Strip' 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 (1).*

–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 x This option is an abbreviation for the library name *'/lib/lib x .a',* where *x* is a string. If that does not exist, *ld* tries *'/usr/lib/lib x .a'.* 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 (1)* 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 to 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.
−o	The *name* argument after **−o** is used as the name of the *ld* output file, instead of **a.out** .
−e	The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.
−O	This is an overlay file, only the text segment will be replaced by *exec (2).* Shared data must have the same layout as in the program overlaid.
−D	The next argument is a decimal number that sets the size of the data segment.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), ranlib(1)

NOTES

LEARN(6)

NAME

learn – computer aided instruction about TNIX

SYNOPSIS

learn [–directory] [subject [lesson [speed]]]

DESCRIPTION

Learn gives CAI courses and practice in the use of TNIX. To get started simply type 'learn'. The program will ask questions to find out what you want to do. The questions may be bypassed by naming a *subject,* and the last *lesson* number that *learn* told you in the previous session. You may also include a *speed* number that was given with the lesson number (but without the parentheses that *learn* places around the speed number). If *lesson* is '–', *learn* prompts for each lesson; this is useful for debugging.

The *subjects* presently handled are

editor
eqn
files
macros
morefiles
C

The special command 'bye' terminates a *learn* session.

The *–directory* option allows one to exercise a script in a nonstandard place.

FILES

/usr/learn and all dependent directories and files

BUGS

The main strength of *learn,* that it asks the student to use the real TNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a TNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Such lessons may be skipped, but it takes some sophistication to recognize the situation.

[This page intentionally left blank.]

LEX(6)

NAME

lex – generator of lexical analysis programs

SYNTAX

lex [**−tvfn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analyis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, 'lex.yy.c' is generated, to be compiled thus:

```
cc lex.yy.c −ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The following *lex* program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A−Z] putchar(yytext[0]+'a'−'A');
[ ]+$
[ ]+   putchar(' ');
```

The options have the following meanings.

−t Place the result on the standard output instead of in file 'lex.yy.c'.

−v Print a one-line summary of statistics of the generated analyzer.

−n Opposite of −v ; −n is default.

−f 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

SEE ALSO

yacc(1)
M. E. Lesk and E. Schmidt, *LEX – Lexical Analyzer Generator*

[This page intentionally left blank.]

LINT(6)

NAME

lint – a C program verifier

SYNTAX

lint [**–abchnpuvxB**] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint ;* these libraries are referred to by a conventional name, such as '-lm', in the style of *ld (1).*

Any number of the options in the following list may be used. The –D , –U , and –I options of *cc (1)* are also recognized as separate arguments.

p	Attempt to check portability to the *IBM* and *GCOS* dialects of C.
h	Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.
b	Report *break* statements that cannot be reached. (This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.)
v	Suppress complaints about unused arguments in functions.
x	Report variables referred to by extern declarations, but never used.
a	Report assignments of long values to int variables.
c	Complain about casts which have questionable portability.
u	Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).
n	Do not check compatibility against the standard library.
B	Use the backup C preprocessor which allows a greater number of defines.

Exit (2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint :*

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*VARARGS n */
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*NOSTRICT*/
shuts off strict type checking in the next expression.

/*ARGSUSED*/
turns on the −v option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

FILES

/usr/lib/lint[1 2] programs
/usr/lib/llib-lc declarations for standard functions
/usr/lib/llib-port declarations for portable functions

SEE ALSO

cc(1)
S. C. Johnson, *Lint, a C Program Checker*

LOOK(6)

NAME

look – find lines in a sorted list

SYNTAX

look [**–df**] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options **d** and **f** affect comparisons as in *sort (1):*

d	'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.
f	Fold. Upper case letters compare equal to lower case.

If no *file* is specified, */usr/dict/words* is assumed with collating sequence **–df.**

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

[This page intentionally left blank.]

LOOKALL(6)

NAME

lookall – look through all text files on TNIX

SYNTAX

lookall [−C n]

DESCRIPTION

Lookall accepts keywords from the standard input, performs a search similar to that of *refer (1),* and writes the result on the standard output. *Lookall* consults, however, an index to all the text files on the system rather than just bibliographies. Only the first 50 words of each file (roughly) were used to make the indexes. Blank lines are taken as delimiters between queries.

The **-C** n option specifies a coordination level search: up to *n* keywords may be missing from the answers, and the answers are listed with those containing the most keywords first.

The command sequence in */usr/dict/lookall/makindex* regenerates the index.

FILES

The directory */usr/dict/lookall* contains the index files.

DIAGNOSTICS

'Warning: index precedes file ...' means that a file has been changed since the index was made and it may be retrieved (or not retrieved) erroneously.

NOTES

Coordination level searching doesn't work as described: only those acceptable items with the smallest number of missing keywords are retreived.

[This page intentionally left blank.]

LORDER(6)

NAME

lorder – find ordering relation for an object library

SYNTAX

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see *ar (1)) files.* The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort (1)* to find an ordering of a library suitable for one-pass access by *ld (1).*

This brash one-liner intends to build a new library from existing '.o' files.

```
ar cr library ` lorder *.o | tsort`
```

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld(1), ar(1)

NOTES

The names of object files, in and out of libraries, must end with '.o'; nonsense results otherwise.

[This page intentionally left blank.]

M4(6)

NAME

m4 – macro processor

SYNTAX

m4 [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no arguments, or if an argument is '–', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

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

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

Left and right single quotes (`') are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n -th* argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string.

undefine
removes the definition of the macro named in its argument.

ifdef If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on TNIX versions of *m4* .

changequote
Change quote characters to the first and second arguments. *Changequote* without arguments restores the original values (i.e., `').

divert *M4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert
causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum
returns the value of the current output stream.

dnl reads and discards characters up to and including the next newline.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation); relationals; parentheses.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

substr returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit
transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include
returns the contents of the file named in the argument.

sinclude
is identical to *include,* except that it says nothing if the file is inaccessible.

syscmd
executes the TNIX command given in the first argument. No value is returned.

maketemp
fills in a string of XXXXX in its argument with the current process id.

errprint
prints its argument on the diagnostic output file.

dumpdef
prints current names and definitions, for the named items, or for all if no arguments are given.

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The M4 Macro Processor*

[This page intentionally left blank.]

NM(6)

NAME

nm – print name list

SYNTAX

nm [**−gnopruh**] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in 'a.out' are listed.

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.

Options are:

−g	Print only global (external) symbols.
−n	Sort numerically rather than alphabetically.
−o	Prepend file or archive element name to each output line rather than only once.
−p	Don't sort; print in symbol-table order.
−r	Sort in reverse order.
−u	Print only undefined symbols.
−h	Print values in hex notation instead of octal.

SEE ALSO

ar(1), ar(5), a.out(5)

[This page intentionally left blank.]

PLOT(6)

NAME

plot – graphics filters

SYNTAX

plot [**–T terminal** [raster]]

DESCRIPTION

These commands read plotting instructions (see *plot (5))* from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter $TERM (see *environ (5))* is used. Known *terminals* are:

4014	Tektronix 4014 storage scope.
450	DASI Hyterm 450 terminal (Diablo mechanism).
300	DASI 300 or GSI terminal (Diablo mechanism).
300S	DASI 300S terminal (Diablo mechanism).
ver	Versatec D1200A printer-plotter. This version of *plot* places a scan-converted image in '/usr/tmp/raster' and sends the result directly to the plotter device rather than to the standard output. The optional argument causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/bin/tek
/bin/t450
/bin/t300
/bin/t300s
/bin/vplot
/usr/tmp/raster

SEE ALSO

plot(3), plot(5)

NOTES

There is no lockout protection for /usr/tmp/raster.

[This page intentionally left blank.]

PREP(6)

NAME

prep – prepare text for statistical processing

SYNTAX

prep [**–dio**] file ...

DESCRIPTION

Prep reads each *file* in sequence and writes it on the standard output, one 'word' to a line. A word is a string of alphabetic characters and imbedded apostrophes, delimited by space or punctuation. Hyphented words are broken apart; hyphens at the end of lines are removed and the hyphenated parts are joined. Strings of digits are discarded.

The following option letters may appear in any order:

–d Print the word number (in the input stream) with each word.

–i Take the next *file* as an 'ignore' file. These words will not appear in the output. (They will be counted, for purposes of the **–d** count.)

–o Take the next *file* as an 'only' file. Only these words will appear in the output. (All other words will also be counted for the **–d** count.)

–p Include punctuation marks (single nonalphanumeric characters) as separate output lines. The punctuation marks are not counted for the **–d** count.

Ignore and only files contain words, one per line.

SEE ALSO

deroff(1)

[This page intentionally left blank.]

PROF(6)

NAME

prof – display profile data

SYNTAX

prof [**−v**] [**−a**] [**−l**] [**−** **low** [*−high*]] [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 graphic version of the profile is produced on the standard output for display by the *plot (1)* filters. The numbers *low* and *high,* by default 0 and 100, cause a selected percentage of the profile to be plotted with accordingly higher resolution.

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

SEE ALSO

monitor(3), profil(2), cc(1), plot(1)

NOTES

Beware of quantization errors.

[This page intentionally left blank.]

PSTAT(6)

NAME

pstat – print system facts

SYNOPSIS

pstat [**–aixptuf**] [suboptions] [file]

DESCRIPTION

Pstat interprets the contents of certain system tables. If *file* is given, the tables are sought there, otherwise in */dev/mem.* The required namelist is taken from */unix.* Options are

–a	Under **–p**, describe all process slots rather than just active ones.
–i	Print the inode table with the these headings:
LOC	The core location of this table entry.
FLAGS	Miscellaneous state variables encoded thus:
L	locked
U	update time *filsys*(5)) must be corrected
A	access time must be corrected
M	file system is mounted here
W	wanted by another process (L flag is on)
T	contains a text file
C	changed time must be corrected
CNT	Number of open file table entries for this inode.
DEV	Major and minor device number of file system in which this inode resides.
INO	I-number within the device.
MODE	Mode bits, see *chmod*(2).
NLK	Number of links to this inode.
UID	User ID of owner.
SIZ/DEV	Number of bytes in an ordinary file, or major and minor device of special file.
–x	Print the text table with these headings:
LOC	The core location of this table entry.
FLAGS	Miscellaneous state variables encoded thus:
T	*ptrace*(2) in effect
W	text not yet written on swap device
L	loading in progress
K	locked
w	wanted (L flag is on)
DADDR	Disk address in swap, measured in multiples of 512 bytes.
CADDR	Core address, measured in multiples of 64 bytes.
SIZE	Size of text segment, measured in multiples of 64 bytes.
IPTR	Core location of corresponding inode.
CNT	Number of processes using this text segment.

CCNT	Number of processes in core using this text segment.
−p	Print process table for active processes with these headings:
LOC	The core location of this table entry.
S	Run state encoded thus:
0	no process
1	waiting for some event
3	runnable
4	being created
5	being terminated
6	stopped under trace
F	Miscellaneous state variables, or-ed together:
01	loaded
02	the scheduler process
04	locked
010	swapped out
020	traced
040	used in tracing
0100	locked in by *lock*(2).
PRI	Scheduling priority, see *nice*(2).
SIGNAL	Signals received (signals 1-16 coded in bits 0-15),
UID	Real user ID.
TIM	Time resident in seconds; times over 127 coded as 127.
CPU	Weighted integral of CPU time, for scheduler.
NI	Nice level, see *nice*(2).
PGRP	Process number of root of process group (the opener of the controlling terminal).
PID	The process ID number.
PPID	The process ID of parent process
ADDR	If in core, the physical address of the 'u-area' of the process measured in multiples of 64 bytes. If swapped out, the position in the swap area measured in multiples of 512 bytes.
SIZE	Size of process image in multiples of 64 bytes.
WCHAN	Wait channel number of a waiting process.
LINK	Link pointer in list of runnable processes.
TEXTP	If text is pure, pointer to location of text table entry.
CLKT	Countdown for *alarm*(2) measured in seconds.
−t	Print table for terminals (only DH11 and DL11 handled) with these headings:
RAW	Number of characters in raw input queue.
CAN	Number of characters in canonicalized input queue.
OUT	Number of characters in putput queue.
MODE	See *tty*(4).
ADDR	Physical device address.
DEL	Number of delimiters (newlines) in canonicalized input queue.
COL	Calculated column position of terminal.
STATE	Miscellaneous state variables encoded thus:
W	waiting for open to complete

O	open
S	has special (output) start routine
C	carrier is on
B	busy doing output
A	process is awaiting output
X	open for exclusive use
H	hangup on close
PGRP	Process group for which this is controlling terminal.

−u print information about a user process; the next argument is its address as given by *ps*(1). The process must be in main memory, or the file used can be a core image and the address 0.

−f Print the open file table with these headings:

LOC	The core location of this table entry.
FLG	Miscellaneous state variables encoded thus:
R	open for reading
W	open for writing
P	pipe
CNT	Number of processes that know this open file.
INO	The location of the inode table entry for this file.
OFFS	The file offset, see *lseek*(2).

FILES

/unix	namelist
/dev/mem	default source of tables

SEE ALSO

ps(1), stat(2), filsys(5)

[This page intentionally left blank.]

PTX(6)

NAME

ptx – permuted index

SYNTAX

ptx [option] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an *nroff* or *troff (1)* macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head,* at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

–f Fold upper and lower case letters for sorting.

–t Prepare the output for the phototypesetter; the default line length is 100 characters.

–w n Use the next argument, *n,* as the width of the output line. The default line length is 72 characters.

–g n Use the next argument, *n,* as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

–o only
Use as keywords only the words given in the *only* file.

–i **ignore**
Do not use as keywords any words given in the *ignore* file. If the *-i* and *-o* options are missing, use /usr/lib/eign as the *ignore* file.

–b **break**
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

–r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

FILES

/bin/sort
/usr/lib/eign

NOTES

Line length counts do not account for overstriking or proportional spacing.

PUBINDEX(6)

NAME

pubindex – make inverted bibliographic index

SYNTAX

pubindex [file] ...

DESCRIPTION

Pubindex makes a hashed inverted index to the named *files* for use by *refer (1).* The *files* contain bibliographic references separated by blank lines. A bibliographic reference is a set of lines that contain bibliographic information fields. Each field starts on a line beginning with a '%', followed by a key-letter, followed by a blank, and followed by the contents of the field, which continues until the next line starting with '%'. The most common key-letters and the corresponding fields are:

A	Author name
B	Title of book containing article referenced
C	City
D	Date
d	Alternate date
E	Editor of book containing article referenced
G	Government (CFSTI) order number
I	Issuer (publisher)
J	Journal
K	Other keywords to use in locating reference
M	Technical memorandum number
N	Issue number within volume
O	Other commentary to be printed at end of reference
P	Page numbers
R	Report number
r	Alternate report number
T	Title of article, book, etc.
V	Volume number
X	Commentary unused by *pubindex*

Except for 'A', each field should only be given once. Only relevant fields should be supplied. An example is:

```
%T 5-by-5 Palindromic Word Squares
%A M. D. McIlroy
%J Word Ways
%V 9
%P 199-202
%D 1976
```

FILES

x.ia, x.ib, x.ic where *x* is the first argument.

SEE ALSO

refer(1)

[This page intentionally left blank.]

RANLIB(6)

NAME

ranlib – convert archives to random libraries

SYNTAX

ranlib archive ...

DESCRIPTION

Ranlib converts each *archive* to a form which can be loaded more rapidly by the loader, by adding a table of contents named __.**SYMDEF** to the beginning of the archive. It uses *ar (1)* to reconstruct the archive, so that sufficient temporary file space must be available in the file system containing the current directory.

SEE ALSO

ld(1), ar(1)

NOTES

Because generation of a library by *ar* and randomization by *ranlib* are separate, phase errors are possible. The loader *ld* warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

[This page intentionally left blank.]

RATFOR(6)

NAME

ratfor – rational Fortran dialect

SYNTAX

ratfor [option ...] [filename ...]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:
{ statement; statement; statement }

decision-making:
if (condition) statement [else statement]
switch (integer value) { case integer:statement ... [default:]statement }

loops: while (condition) statement for (expression; condition; expression) statement do limits statement repeat statement [until (condition)] break [n] next [n]

and some syntactic sugar to make programs easier to read and write:

free form input:
multiple statements/line; automatic continuation

comments:
# this is a comment

translation of relationals:
>, > =, etc., become .GT., .GE., etc.

return (expression)
returns expression to caller from function

define:
define name replacement

include:
include filename

The option **–h** causes quoted strings to be turned into 27H constructs. **–C** copies comments to the output, and attempts to format it neatly. Normally, continuation lines are marked with a & in column 1; the option **–6x** makes the continuation character **x** and places it in column 6.

Ratfor is best used with *f77 (1).*

SEE ALSO

f77(1)
B. W. Kernighan and P. J. Plauger, *Software Tools* , Addison-Wesley, 1976.

[This page intentionally left blank.]

REFER/LOOKBIB(6)

NAME

refer, lookbib – find and insert literature references in documents

SYNTAX

refer [option] ...

lookbib [file] ...

DESCRIPTION

Lookbib accepts keywords from the standard input and searches a bibliographic data base for references that contain those keywords anywhere in title, author, journal name, etc. Matching references are printed on the standard output. Blank lines are taken as delimiters between queries.

Refer is a preprocessor for *nroff* or *troff (1)* that finds and formats references. The input files (standard input default) are copied to the standard output, except for lines between .[and .] command lines, which are assumed to contain keywords as for *lookbib,* and are replaced by information from the bibliographic data base. The user may avoid the search, override fields from it, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms (7)* print the finished reference text from these strings. A flag is placed in the text at the point of reference; by default the references are indicated by numbers.

The following options are available:

–a r Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted all author names are reversed.

–b Bare mode: do not put any flags in text (neither numbers nor labels).

–c string
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *string* .

–e Instead of leaving the references where encountered, accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

–k x Instead of numbering references, use labels as specified in a reference data line beginning *%x;* by default *x* is L.

–l m , n
Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or , **n** is omitted the entire name or date respectively is used.

−p Take the next argument as a file of references to be searched. The default file is searched last.

−n Do not search the default file.

−s keys
Sort references by fields whose key-letters are in the *keys* string; permute reference numbers in text accordingly. Implies **−e** . The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with + taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort, for example, on all authors and then title use **-sA+T** .

To use your own references, put them in the format described in *pubindex (1)* They can be searched more rapidly by running *pubindex (1)* on them before using *refer;* failure to index results in a linear search.

When *refer* is used with *eqn, neqn* or *tbl, refer* should be first, to minimize the volume of data passed through pipes.

FILES
/usr/dict/papers directory of default publication lists and indexes
/usr/lib/refer directory of programs

SEE ALSO

REV(6)

NAME
rev − reverse lines of a file

SYNTAX
rev [file] ...

DESCRIPTION
Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

SED(6)

NAME

sed – stream editor

SYNTAX

sed [−n] [−**e** script] [−**f** sfile] [file] ...

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The −**f** option causes the script to be taken from file *sfile ;* these options accumulate. If there is just one −**e** option and no −**f** '**s**, the flag −**e** may be omitted. The −n option suppresses the default output.

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a 'D' command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under −n) and deletes the pattern space.

An *address* is either a decimal number that counts input lines cumulatively across files, a '$' that addresses the last line of input, or a context address, '/regular expression/', in the style of *ed (1)* modified thus:

> The escape sequence '\n' matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function '!' (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

An argument denoted *text* consists of one or more lines, all but the last of which end with '\' to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an 's' command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An argument denoted *rfile* or *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a
text
Append. Place *text* on the output before reading the next input line.

(2) b label
Branch to the ':' command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) c
text
Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2) D Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) g Replace the contents of the pattern space by the contents of the hold space.

(2) G Append the contents of the hold space to the pattern space.

(2) h Replace the contents of the hold space by the contents of the pattern space.

(2) H Append the contents of the pattern space to the hold space.

(1) i
text Insert. Place *text* on the standard output.

(2) l List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two digit ascii, and long lines are folded.

(2) n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) N Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) p Print. Copy the pattern space to the standard output.

(2) P Copy the initial segment of the pattern space through the first newline to the standard output.

(1) q Quit. Branch to the end of the script. Do not start a new cycle.

(2) r rfile
Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) s /regular expression/replacement/flags
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of '/'. For a fuller description see *ed (1). Flags* is zero or more of

- **g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
- **p** Print the pattern space if a replacement was made.
- **w wfile**
  Write. Append the pattern space to *wfile* if a replacement was made.

(2) t label
Test. Branch to the ':' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a 't'. If *label* is empty, branch to the end of the script.

(2) w wfile
Write. Append the pattern space to *wfile* .

(2) x Exchange the contents of the pattern and hold spaces.

(2) y /string1/string2/
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2.* The lengths of *string1* and *string2* must be equal.

(2)! function
Don't. Apply the *function* (or group, if *function* is '{') only to lines *not* selected by the address(es).

(0) : label
This command does nothing; it bears a *label* for 'b' and 't' commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching '}' only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO
ed(1), grep(1), awk(1)

[This page intentionally left blank.]

SIZE(6)

NAME

size – size of an object file

SYNTAX

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.

SEE ALSO

a.out(5)

[This page intentionally left blank.]

SPELL(6)

NAME

spell, spellin, spellout – find spelling errors

SYNTAX

spell [option] ... [file] ...

/usr/dict/spellin [list]

/usr/dict/spellout [**−d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most *troff, tbl* and *eqn (1)* constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the **−b** option, British spelling is checked. Besides preferring *centre, colour, speciality, travelled,* etc., this option insists upon *-ise* in words like *standardise,* Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with '=' for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g. thier=thy−y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell.* Both expect a list of words, one per line, from the standard input. *Spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option **−d**) the hash list.

FILES

D=/usr/dict/hlist[ab]: hashed spelling lists, American & British
S=/usr/dict/hstop: hashed stop list
H=/usr/dict/spellhist: history file
/usr/lib/spell
deroff(1), sort(1), tee(1), sed(1)

NOTES

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.
British spelling was done by an American.

[This page intentionally left blank.]

SPLINE(6)

NAME

spline – interpolate smooth curve

SYNTAX

spline [option] ...

DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph (1).*

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

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

−k The constant k used in the boundary value computation

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument. By default $k = 0$.

−n Space output points so that approximately n intervals occur between the lower and upper x limits. (Default $n = 100$.)

−p Make output periodic, i.e. match derivatives at ends. First and last input values should normally agree.

−x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1)

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

NOTES

A limit of 1000 input points is enforced silently.

[This page intentionally left blank.]

SPLIT(6)

NAME

split – split a file into pieces

SYNTAX

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.

[This page intentionally left blank.]

STRIP(6)

NAME

strip – remove symbols and relocation bits

SYNTAX

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

FILES

/tmp/stm?	temporary file

SEE ALSO

ld(1)

[This page intentionally left blank.]

STRUCT(6)

NAME

struct – structure Fortran programs

SYNTAX

struct [option] ... file

DESCRIPTION

Struct translates the Fortran program specified by *file* (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original Fortran. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (.e.g. '.GT.' into '>'). The output is appropriately indented.

The following options may occur in any order.

–s Input is accepted in standard format, i.e. comments are specified by a c, C, or * in column 1, and continuation lines are specified by a nonzero, nonblank character in column 6. Normally, a statement whose first nonblank character is not alphanumeric is treated as a continuation.

–i Do not turn computed goto statements into switches. (Ratfor does not turn switches back into computed goto statements.)

–a Turn sequences of else ifs into a non-Ratfor switch of the form

```
switch {
            case pred1: code
            case pred2: code
            case pred3: code
            default: code
        }
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

–b Generate goto's instead of multilevel break statements.

–n Generate goto's instead of multilevel next statements.

–e n If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit code segments with fewer than *n* statements to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop.

FILES

/tmp/struct*
/usr/lib/struct/*

SEE ALSO

f77(1)

NOTES

Struct knows Fortran 66 syntax, but not full Fortran 77 (alternate returns, IF...NMEN...ELSE, etc.)

If an input Fortran program contains identifiers which are reserved words in Ratfor, the structured version of the program will not be a valid Ratfor program.

Extended range DO's generate cryptic errors.

Columns 73-80 are not special even when **−s** is in effect.

Will not generate Ratfor FOR statements.

SUM(6)

NAME

sum – sum and count blocks in a file

SYNTAX

sum file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinuishable from end of file on most devices; check the block count.

[This page intentionally left blank.]

TABS(6)

NAME

tabs – set terminal tabs

SYNTAX

tabs [−n] [terminal]

DESCRIPTION

Tabs sets the tabs on a variety of terminals. Various of the terminal names given in *term (7)* are recognized; the default is, however, suitable for most 300 baud terminals. If the −n flag is present then the left margin is not indented as is normal.

SEE ALSO

stty(1), term(7)

[This page intentionally left blank.]

TAR(6)

NAME

tar – tape archiver

SYNTAX

tar [key] [name ...]

DESCRIPTION

Tar saves and restores files on magtape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped or restored. 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 end of the tape. The **c** function implies this.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape.

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies r.

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

0,...,7 This modifier selects the drive on which the tape is mounted. The default is 1 .

v Normally *tar* 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.

w causes *tar* to print the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is performed. Any other input means don't do it.

f causes *tar* to use the next argument as the name of the archive instead of /dev/mt?. If the name of the file is '–', tar writes to standard output or reads from standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain *Tar* can also be used to move hierarchies with the command

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

b causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (See f above). The block size is determined automatically when reading tapes (key letters 'x' and 't').

l tells *tar* to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m tells *tar* to not restore the modification times. The mod time will be the time of extraction.

FILES

/dev/mt?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

NOTES

There is no way to ask for the *n -th* occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated. The current magtape driver cannot backspace raw magtape. If the archive is on a disk file the **b** option should not be used at all, as updating an archive stored in this manner can destroy it.
The current limit on file name length is 100 characters.

TBL(6)

NAME

tbl – format tables for nroff or troff

SYNTAX

tbl [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff (1).* The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and reformatted. Details are given in the reference manual.

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

```
      Household Population
     Town        Households
               Number   Size
Bedminster       789    3.26
Bernards Twp.   3087    3.74
Bernardsville   2018    3.30
Bound Brook     3425    3.04
Branchburg      1644    3.49
Bridgewater     7897    3.81
Far Hills        240    3.19
```

If no arguments are given, *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

SEE ALSO

troff(1), eqn(1)
M. E. Lesk, *TBL.*

[This page intentionally left blank.]

TC(6)

NAME

tc – photypesetter simulator

SYNTAX

tc [**–t**] [**–sN**] [**–pL**] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Graphic Systems phototypesetter (cat). The standard output of *tc* is intended for a Tektronix 4015 (a 4014 teminal with ASCII and APL character sets). The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary. Typical usage:

```
15      troff –t file | tc
```

At the end of each page *tc* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will suppress the screen erase before the next page; **s**N will cause the next N pages to be skipped; and !line will send line to the shell.

The command line options are:

–t Don't wait between pages; for directing output into a file.

–s N Skip the first N pages.

–p L Set page length to L. L may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

'**– l w** '
Multiply the default aspect ratio, 1.5, of a displayed page by *l/w.*

SEE ALSO

troff(1), plot(1)

NOTES

Font distinctions are lost.
The aspect ratio option is unbelievable.

[This page intentionally left blank.]

TK(6)

NAME

tk – paginator for the Tektronix 4014

SYNTAX

tk [−**t**] [−*N*] [−**p***L*] [file]

DESCRIPTION

The output of *tk* is intended for a Tektronix 4014 terminal. *Tk* arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page *tk* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command ! **command** will send the *command* to the shell.

The command line options are:

−t	Don't wait between pages; for directing output into a file.
− N	Divide the screen into *N* columns and wait after the last column.
−p L	Set page length to *L* lines.

SEE ALSO

pr(1)

[This page intentionally left blank.]

TROFF/NROFF(6)

NAME

troff, nroff – text formatting and typesetting

SYNTAX

nroff [option] ... [file] ...

troff [option] ... [file] ...

DESCRIPTION

Troff formats text in the named *files* for printing on a Graphic Systems C/A/T phototypesetter; *nroff* for typewriter-like devices. Their capabilities are described in the *Nroff/Troff User's Manual. Nroff* and *troff* are normally used in conjunction with a predefined macro package; for example, see *ms (1).*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

–o list Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N* – *M* means pages *N* through *M* ; an initial –*N* means from the beginning to page *N* ; and a final *N* – means from *N* to the end.

–n N Number first generated page *N* .

–s N Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N =1)* to allow paper loading or changing, and will resume upon receipt of a newline. *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

–m name
Prepend the macro file **/usr/lib/tmac/tmac***name* to the input *files* .

–r aN Set register *a* (one-character) to *N* .

–i Read standard input after the input files are exhausted.

–q Invoke the simultaneous input-output mode of the **rd** request.

Nroff only

–T name
Prepare output for specified terminal. For example, –**Tq** specifies a Qume Sprint-5 printer and –**T4025** a Tektronix 4025 terminal. See */usr/lib/nterm/list* for a complete list.

–e Produce equally-spaced words in adjusted lines, using full terminal resolution.

–h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

Troff only

−t	Direct output to the standard output instead of the phototypesetter.
−f	Refrain from feeding out paper and stopping phototypesetter at the end of the run.
−w	Wait until phototypesetter is available, if currently busy.
−b	Report whether the phototypesetter is busy or available. No text processing is done.
−a	Send a printable ASCII approximation of the results to the standard output.
−p N	Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elasped time.
−g	Prepare output for a GCOS phototypesetter and direct it to the standard output (see *gcat (1)).*

If the file */usr/adm/tracct* is writable, *troff* keeps phototypesetter accounting records there. The integrity of that file may be secured by making *troff* a 'set user-id' program.

FILES

/usr/lib/suftab(?)	suffix hyphenation tables
/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/nterm/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

SEE ALSO

J. F. Ossanna, *Nroff/Troff User's Manual*
B. W. Kernighan, *A TROFF Tutorial*
R. A. LeFaivre, *Addendum: Tektronix Modifications to Nroff/Troff*
eqn(1), tbl(1), ms(1)
col(1), tk(1) *(nroff* only)
tc(1), gcat(1), vcat(1) *(troff* only)

TSORT(6)

NAME

tsort – topological sort

SYNTAX

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file* . If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

NOTES

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

[This page intentionally left blank.]

UNITS(6)

NAME

units – conversion program

SYNTAX

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.54000e+00
        / 3.93701e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 pounds force/in2
You want: atm
        * 1.02069e+00
        / 9.79730e-01
```

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

NOTES

Don't base your financial plans on the currency conversions.

[This page intentionally left blank.]

XSEND/XGET/ENROLL(6)

NAME

xsend, xget, enroll – secret mail

SYNOPSIS

xsend person
xget
enroll

DESCRIPTION

These commands implement a secure communication channel; it is like *mail*(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use *enroll*; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use *xget*. It asks for your password, then gives you the messages.

To send secret mail, use *xsend* in the same manner as the ordinary mail command. (However, it will accept only one target). A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

/usr/spool/secretmail/*.key: keys /usr/spool/secretmail/*.[0-9]: messages

SEE ALSO

mail (1)

BUGS

It should be integrated with ordinary mail. The announcement of secret mail makes traffic analysis possible.

[This page intentionally left blank.]

YACC(6)

NAME

yacc – yet another compiler-compiler

SYNTAX

yacc [**–vd**] 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 file, *y.tab.c* , must be compiled by the C compiler to produce a program *yyparse* . This program must be loaded with the lexical analyzer program, *yylex* , as well as *main* and *yyerror* , an error handling routine. These routines must be supplied by the user; *Lex (1)* is useful for creating lexical analyzers usable by *yacc* .

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.

If the **–d** flag is used, the file *y.tab.h* is generated with the *define* statements that associate the *yacc*-assigned 'token codes' with the user-declared 'token names'. This allows source files other than *y.tab.c* to access the token codes.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs
/lib/liby.a	library with default 'main' and 'yyerror'

SEE ALSO

lex (1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC – Yet Another Compiler Compiler by S. C. Johnson.

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. Similarly, if some rules are not reachable from the start symbol, this is also reported.

NOTES

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

[This page intentionally left blank.]

Section 7
Macros and other Packages

INTRODUCTION

This section briefly describes the various macro packages intended for use with the nroff/troff text processors. This section includes other useful information, such as an ASCII character set and a listing of the **TNIX** directory hierarchy.

ASCII(7)

NAME

ascii – map of ASCII character set

SYNTAX

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel|
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb|
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us |
|040 sp |041  ! |042  " |043  # |044  $ |045  % |046  & |047  ´ |
|050  ( |051  ) |052  * |053  + |054  , |055  - |056  . |057  / |
|060  0 |061  1 |062  2 |063  3 |064  4 |065  5 |066  6 |067  7 |
|070  8 |071  9 |072  : |073  ; |074  < |075  = |076  > |077  ? |
|100  @ |101  A |102  B |103  C |104  D |105  E |106  F |107  G |
|110  H |111  I |112  J |113  K |114  L |115  M |116  N |117  O |
|120  P |121  Q |122  R |123  S |124  T |125  U |126  V |127  W |
|130  X |131  Y |132  Z |133  [ |134  \ |135  ] |136  ^ |137  _ |
|140  ` |141  a |142  b |143  c |144  d |145  e |146  f |147  g |
|150  h |151  i |152  j |153  k |154  l |155  m |156  n |157  o |
|160  p |161  q |162  r |163  s |164  t |165  u |166  v |167  w |
|170  x |171  y |172  z |173  { |174  | |175  } |176  ~ |177 del|
```

FILES

/usr/pub/ascii

EQNCHAR(7)

Name

eqnchar – special character definitions for eqn

SYNTAX

eqn /usr/pub/eqnchar [files] **|** **troff** [options]

neqn /usr/pub/eqnchar [files] **|** **nroff** [options]

DESCRIPTION

Eqnchar contains *troff* and *nroff* character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with *eqn* and *neqn* . It contains definitions for the following characters

ciplus	⊕	||	∥	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	▣
-wig	≃	*hbar*	ħ	*bullet*	●
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	<->	↔	*empty*	∅
=wig	≅	<=>	⇔	*member*	∈
star	*	|	≮	*nomem*	∉
bigstar	✳	|>	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	∟	*incl*	⊑
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
		angstrom	Å	*degree*	°

FILES

/usr/pub/eqnchar

SEE ALSO

troff(1), eqn(1)

HIER(7)

NAME

hier – file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative directory hierarchy.

/	root	
/dev/	devices (4)	
	console	main console, *tty (4)*
	tty*	terminals, *tty (4)*
	...	
/bin/	utility programs, cf /usr/bin/ (1)	
	as	assembler first pass, cf /usr/lib/as2
	cc	C compiler executive, cf /usr/lib/c[012]
	...	
/lib/	object libraries and other stuff, cf /usr/lib/	
	libc.a	system calls, standard I/O, etc. (2,3,3S)
	libm.a	math routines (3M)
	libplot.a	plotting routines, *plot (3)*
	libF77.a	Fortran runtime support
	libI77.a	Fortran I/O
	...	
	as2	second pass of *as (1)*
	c[012]	passes of *cc (1)*
	...	
/etc/	essential data and dangerous maintenance utilities	
	passwd	password file, *passwd (5)*
	group	group file, *group (5)*
	motd	message of the day, *login (1)*
	mtab	mounted file table, *mtab (5)*
	ddate	dump history, *dump (1)*

ttys properties of terminals, *ttys (5)*

getty part of *login , getty (8)*

init the father of all processes, *init (8)*

rc shell program to bring the system up

cron the clock daemon, *cron (8)*

mount *mount (1)*

wall *wall (1)*

...

/tmp/ temporary files, usually on a fast device, cf /usr/tmp/

e* used by *ed (1)*

ctm* used by *cc (1)*

...

/usr/ general-pupose directory, usually a mounted file system

/usr **/bin**
utility programs, to keep /bin/ small

tmp/ temporaries, to keep /tmp/ small

stm* used by *sort (1)*

raster used by *plot (1)*

dict/ word lists, etc.

words principal word list, used by *look (1)*

spellhist
history file for *spell (1)*

include/
standard #include files

a.out.h object file layout, *a.out (5)*

stdio.h
standard I/O, *stdio (3)*

math.h
(3M)

...

sys/ system-defined layouts, cf /usr/sys/h

acct.h process accounts, *acct (5)*

buf.h internal system buffers

...

lib/ object libraries and stuff, to keep /lib/ small

- **lint[12]** subprocesses for *lint (1)*
- **llib-lc** dummy declarations for /lib/libc.a, used by *lint (1)*
- **llib-lm** dummy declarations for /lib/libc.m
- **atrun** scheduler for *at (1)*
- **struct/** passes of *struct (1)*
- ...
- **tmac/** macros for *troff (1)*
  - **tmac.an** macros for *man (7)*
  - **tmac.s** macros for *ms (7)*
  - ...
- **font/** fonts for *troff (1)*
  - **R** Times Roman
  - **B** Times Bold
  - ...
- **uucp/** programs and data for *uucp (1)*
  - **L.sys** remote system names and numbers
  - **uucico** the real copy program
  - ...
- **suftab** table of suffixes for hyphenation, used by *troff (1)*
- **units** conversion tables for *units (1)*
- **eign** list of English words to be ignored by *ptx (1)*

/usr/ **man/** on-line reference manual, *man (1)*

- **cat1/** preprinted pages for man1/
  - **as.1**
  - **mount.1m**
- ...

spool/ delayed execution files

- **at/** used by *at (1)*
- **lpd/** used by *lpr (1)*
  - **lock** present when line printer is active
  - **cf*** copy of file to be printed, if necessary
  - **df*** daemon control file, *lpd (8)*
  - **tf*** transient control file, while *lpr* is working

mail/ mailboxes for *mail (1)*

- **uid** mail file for user *uid*
- **uid .lock**
  lock file while *uid* is receiving mail

wd initial working directory of a user, typically *wd* is the user's login name

- **. profile**
  set environment for *sh (1), environ (5)*
- **calendar**
  user's datebook for *calendar (1)*

SEE ALSO

ls(1), ncheck(1), find(1), grep(1)

NOTES

The position of files is subject to change without notice.

MAN(7)

NAME

man – macros to typeset manual

SYNTAX

nroff −man file ...

troff −man file ...

DESCRIPTION

These macros are used to lay out pages of this manual. A skeleton page may be found in the file /usr/man/man0/xx.

Any text argument *t* may be zero to six words. Quotes may be used to include blanks in a 'word'. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way . **I** may be used to italicize a whole line, or . **SM** followed by . **B** to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

~~These strings are predefined by −man :~~

\*R	'.ft R ,if t .I nroff.
\*S	Change to default type size.

FILES

/usr/lib/tmac/tmac.an
/usr/man/man0/xx

SEE ALSO

troff(1), man(1)

NOTES

Relative indents don't nest.

REQUESTS

Request	Cause Break	If no Argument	Explanation
.Bo *t*	no	*t*=n.t.l.*	Text *t* is bold.
.Bi *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and italic.
.Br *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and Roman.
.Dt	no	.5i 1i...	Restore default tabs.
.Hp *i*	yes	*i*=p.i.*	Set prevailing indent to *i*. Begin paragraph with hanging indent.
.Io *t*	no	*t*=n.t.l.	Text *t* is italic.
.Ib *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and bold.
.Ir *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and Roman.
.Lp	yes	-	Same as .Pp.
.lo d.			

.Pp	-	Begin paragraph. Set prevailing indent to .5i.	
.RE	yes	-	End of relative indent. Set prevailing indent to amount of starting .in +5.
.Rb *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and bold.
.Ri *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, move left margin in distance *i*. Set prevailing indent to .5i for nested indents.
.Sh *t*	yes	*t*=n.t.l.	Subhead.
.Sm *t*	no	*t*=n.t.l.	Text *t* is small.
.NM *n c x*	yes	-	Begin page named *n* of chapter *c; x* is extra commentary, e.g. 'local', for page foot. Set prevailing indent and tabs to .5i.
.Hn i.			Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

MS(7)

NAME

ms – macros for formating manuscripts

SYNTAX

nroff −ms [options] file ...
troff −ms [options] file ...

or

ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a canned formating facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col (1).* Output of the *eqn (1), neqn (1), refer (1),* and *tbl (1)* preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.s*

SEE ALSO

nroff/troff(1), ms(1), eqn(1), refer(1), tbl(1)

REQUESTS

Following is a complete list of available ms formating requests. See the manual(s) for detailed descriptions of the various requests.

.1C	One column format on a new page.
.2C	Two column format.
.AB	Begin abstract.
.AD *f*	Set right margin adjustment on (*f*=1 or missing) or off (*f*=0).
.AE	End abstract.
.AI	Author's institution follows.
.AN *c*	Define auto increment number *c*.
.AU	Author's name follows.
.B *x*	Print *x* in boldface; if no argument switch to boldface.
.BC	Begin new column when in .2C mode.
.BD	Start centered block display which may extend over page boundaries.
.BE	End text to be boxed; print it (also known as .B2).
.BP	Begin a new page.
.BR	Begin a new line ("break" the line).
.BS	Start text to be enclosed in a box (also known as .B1).
.BT	Print page footer at bottom of page. May be redefined.
.BU	Start a bullet item (indented paragraph with bullet label).
.BX *word*	Print *word* in a box.
.CD	Start centered display which may extend over page boundaries.
.COL	Pipe output through *col*(1) if necessary. Must be first.
.CS *f*	Enter constant spacing mode if *f* is missing; leave constant spacing mode if *f* is 0. Ignored in *nroff*.
.DA	Place current date at bottom of each page.
.DE	End displayed text.

.DR	This is a draft document.
.DS *x*	Start of displayed text, to appear verbatim line-by-line. *x*=I for indented display (default), *x*=L for left-justified on the page, *x*=C for centered, *x*=B for make left-justified block, then center whole block. Implies .KS.
.EH	End heading.
.EN	End equation.
.EQ *x y*	Begin an equation. Equation number is *y*. The optional argument *x* may be I to indent the equation, L to left-justify the equation, or C to center the equation (default).
.FE	End footnote.
.FS *x*	Start footnote. *x* is optional label to be placed to the left of the footnote.
.HL	Draw a horizontal line across the page.
.HS *x y*	Specify heading style. O indicates outline form; I indicates indented numbered sections.
.HY *f*	Set hyphenation on (*f*=1 or missing) or off (*f*=0).
.I *x*	Italicize *x*. If no argument switch to italics. Underline in *nroff*.
.ID	Start indented display which may extend over page boundaries.
.IE	End an indented section.
.IOC	IOC style. Must be first. .TO, .FR, .CC, .SU, .DA, .TI, .PL give information for IOC header.
.IP *l x y*	Start indented paragraph, with hanging label *l*. Text indentation is *x* spaces; label is indented *y* spaces.
.IS	Start indented section.
.JU *'l'c'r'*	Justify line, with *l* left-justified, *c* centered, and *r* right-justified.
.KE	End keep. Put preceding text on next page if not enough room.
.KF	Start floating keep. If the kept text must be moved to the next page, float later text back to this page.
.KS	Start keeping following text.
.LD	Start left-justified display which may extend over page boundaries.
.LG	Make letters larger. Ignored in *nroff*.
.LP	Start left-blocked paragraph.
.LS *n*	Set line spacing to *n* lines (2 for double-spacing).
.LT	Business letter style. Must be first.
.ND *date*	Use date supplied in place of actual date.
.NE *n*	Need *n* lines on page; page eject if not enough.
.NH *n*	Same as .SH, with section number supplied automatically. Numbers are multilevel, like 1.2.3, where *n* tells what level is wanted (default is 1).
.NL	Make letters normal size.
.P1	Include header at top of page 1 (normally suppressed).
.PC	Print header preceding table of contents. May be redefined.
.PN *n*	Set page number of next page to *n*.
.PP	Start paragraph. First line indented.
.PT	Print page header at top of page. May be redefined.
.PX	Print header preceding index. May be redefined.
.QE	End quoted material.
.QP	Start quoted paragraph (indented and shorter).
.QS	Start quoted material (indented and shorter).
.R	Roman text follows.
.RD *file*	Read input from *file*.

.RE	End relative indent section.
.RP	Released paper style. Must be first.
.RS	Start level of relative indentation. Following .AR's are measured from current indentation.
.SE	End a section of text to be sorted.
.SH	Section head follows; font automatically bold.
.SM	Make letters smaller. Ignored in *nroff*.
.SO	Sort following text.
.SP *n*	Space *n* lines (1 if missing).
.SZ *n*	Set character size. Ignored in *nroff*.
.TA *x...*	Set tabs.
.TC *text*	Place *text* in the table of contents and also include in text.
.TE	End table.
.TH	End heading section of table.
.TL	Title of document follows.
.TM *x*	Technical memo style, with optional number *x*. Must be first.
.TR *x*	Technical report style, with optional number *x*. Must be first.
.TS *x*	Start table; if *x* is H table has repeated heading.
.UL *word*	Underline argument (even in troff).
.UX	'UNIX'; first time used, add footnote 'UNIX is a trademark of Bell Laboratories.'
.XN *text*	Add *text* to index without a page number.
.XX *text*	Add *text* to index with current page number.

IN-LINE COMMANDS

\space	Unpaddable Space Character
\e	Echo Backslash Character
\%	Suppress Hyphenation
\F*x*	Switch to Font *x* (Also \f)
\F(*xy*	Switch to Font *xy* (Also \f)
\s*n*	Set Character Size to *n* Points
\s±*n*	Increase/Decrease Size by *n* Points
\(*xy*	Special Character *xy*
\o'...'	Overstrike Characters
\"	Ignore Rest of Input Line (For Comments)
*{	Start Superscript
*}	End Superscript
*[	Start Subscript
*]	End Subscript
**x*	Increment and Print Auto Number *x*
\n*x*	Print Auto Number *x* (no incr.)
*(DT	Today's Date
*(DY	Today's Date (Changeable via .ND)
*(DW	Day of the Week
\n(PN	Current Page Number

STRING/NUMBER REGISTERS

.ds LH	Left Portion of Page Header (Initially Null)
.ds CH - \\n(PN -	Center Portion of Page Header
.ds RH	Right Portion of Page Header

.ds LF	Left Portion of Page Footer
.ds CF	Center Portion of Page Footer (\\*(DY if .DA)
.ds RF	Right Portion of Page Footer
.ds NF R	Normal Text Font
.ds HF B	Heading Font (.SH/.NH)
.ds PD 1v	Paragraph Separation (.P/.DS/.SP -- 0.5v if **−Tvpr**)
.ds DI Distribution	Default for Missing .TO Argument in IOC
.nr LL 6i	Line Length (6.5i for IOC)
.nr LT 6i	Header/Footer Length (6.5i for IOC)
.nr FL 6i-3n	Footnote Line Length
.nr PO 0	Page Offset (Appropriate Value if **−Tvpr**)
.nr HM 1i	Top Margin (Header in Middle of Margin)
.nr FM 1i	Bottom Margin (Footer in Middle of Margin)
.nr PI 5n	Paragraph (.P/.AR/.IS) Indent
.nr QI 5n	Quoted Section (.QP/.QS) Indent
.nr NI 4n	Auto Indent for Numbered Sections (.HS I)
.nr PS 10	Character Point Size (Range 6 to about 18)
.nr VS 12	Vertical Spacing (Normally PS+2)
.nr CS 24	Constant spacing character width (.CS)

TERM(7)

NAME

terminals– conventional names

DESCRIPTION

These names are used by certain commands and are maintained as part of the shell environment (see *sh (1), environ (5)).*

1620	DIABLO 1620 (and others using HyType II)
1620–12	same, in 12-pitch mode
300	DASI/DTC/GSI 300 (and others using HyType I)
300–12	same, in 12-pitch mode
300s	DASI/DTC 300/S
300s–12	same, in 12-pitch mode
33	TELETYPE.ft R
Model 33	
37	TELETYPE Model 37
40–2	TELETYPE Model 40/2
43	TELETYPE Model 43
450	DASI 450 (same as Diablo 1620)
450–12	same, in 12-pitch mode
450–12–8	same, in 12-pitch, 8 lines/inch mode
735	Texas Instruments TI735 (and TI725)
745	Texas instruments Ti745
dumb	terminals with no special features
hp	Hewlett-Packard HP264? series terminals
4014	Tektronix 4014
tn1200	General Electric TermiNet 1200
tn300	General Electric TermiNet 300
vt05	Digital Equipment Corp. VT05

Commands whose behavior may depend on the terminal accept arguments of the form **–Tterm**, where *term* is one of the names given above. If no such argument is present, a command may consult the shell environment for the terminal type.

SEE ALSO

stty(1), tabs(1), plot(1), sh(1), environ(5)
troff(1) for *nroff*

NOTES

The programs that ought to adhere to this nomenclature do so only fitfully.

Section 8
System Maintenance

INTRODUCTION

This section describes commands intended for use by the system manager in performing routine software maintenance and problem analysis. Also described here are commands for disaster recovery, system accounting and administration, and those commands which help you to keep your system alive and well. For procedural and tutorial information consult your *8560 MUSDU System Users Manual.*

CRASH(8)

NAME

crash – what to do if the system crashs

DESCRIPTION

This section will give an idea of what went wrong if the system crashes. These errors should never happen. Also in this section is an explanation of how to take a disc dump of the system to send in along with the standard problem report.

TNIX will write a message to the console terminal when it detects a fatal error before crashing. Following is a list of these messages.

Messages with * are indicative of a possible system software problem. A dump of the system at the time of the panic is desirable. The other errors can be caused by such things as a too heavily loaded system, and if they happed too frequently may require system tuning.

tnix: panic: blkdev *

The *getblk* routine was called with a nonexistent major device as an argument. This is definitely a hardware or software error.

tnix: panic: devtab *

The device table entry for the device passed to getblk is null. Definitely a hardware or software error.

tnix: panic: iinit

An I/O error occured while reading the super-block for the root file system during initialization.

tnix: panic: IO err in swap

An unrecoverable I/O error occured during a swap.

tnix: panic: no fs *

A device has disappeared from the mounted-device table. Definitely a hardware or software error.

tnix: panic: no imt *

This is like 'no fs' but it is produced elsewhere. Definitely a hardware or software error.

tnix: panic: no procs *

A process slot has disappeared after just being checked for. Definitely a hardware or software error.

tnix: panic: Out of swap

There are no available entries in the swap map.

tnix: panic: out of swap space

A program needs to be swapped out, and there is no more swap space.

tnix: panic: parity

A memory parity error has occured while in system space.

tnix: panic: Running a dead proc *

A dead process was about to be set running. Definiitely a hardware or software error.

tnix: panic: Sleeping on wchan 0 *

An invalid 0 parameter was passed to sleep. Definiitely a hardware or software error.

tnix: panic: trap *

An unexpected trap has occurred within the system. This is accompanied by three numbers: a 'ka6', which is the contents of the segmentation register for the area in which the system's stack is kept; 'aps', which is the location where the hardware stored the program status word during the trap; 'pc' and 'ps', which are the program counter and program status at the time of the trap; and a 'trap type' which encodes which trap occured. The trap types are:

0	bus error
1	illegal instruction
2	BPT/trace
3	IOT
4	power fail
5	EMT
6	recursive sytem call
10	floating point trap
11	segmentation violation

In some of these cases it is possible for octal 20 to be added to the trap type; this indicates that the processor was in user mode when the trap occured.

NOTE: The following should be done on a scratch floppy disk.

To get a dump of the system at the time of the crash:

Halt the system and plug a terminal into the line printer 2 port. (do not power down or toggle restart!)

Type 'R7/' and then '44', followed by a carriage return.

Type 'P'. The system will now write to the floppy disc for about 40 seconds. When this is done the dump is complete and you may restart the system as ususal, being sure to check the file system for possible errors.

CRON(8)

NAME

cron – clock daemon

SYNTAX

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file /usr/lib/crontab. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file /etc/rc; see *init (8).*

Crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=monday). Each of these patterns may contain a number in the range above; two numbers separated by a minus meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the Shell at the specified times. A percent character in this field is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the Shell. The other lines are made available to the command as standard input.

Crontab is examined by *cron* every minute.

FILES

/usr/lib/crontab

CVT(8)

cvt - examine or alter kernel parameters

SYNTAX
cvt [**-w**] [**-f** tnixname] [script]

PARAMETERS

-w Write the new parameters. Normally, **cvt** can only be used to examine the current TNIX configuration or test the feasibility of a new configuration (e.g., "Will I run out of memory if I add three cache buffers?"). **-w** makes any changes to TNIX permanent.

-f tnixname
Examine/modify tnixname rather than the default kernel, "/tnix".

script A file that contains lines directing **cvt** to examine or set various configuration parameters. If script cannot be found in the current directory, and does not begin with a slash, the file "/etc/cvtscript/script" is checked for. Standard-input is used if no script is given. (Terminate standard input by typing a CTRL-D.)

Any blank lines in the script and any lines beginning with a pound-sign (#) are ignored. A line beginning with "!" is passed to the shell to be executed. Other lines are of the form:

```
parameter
   or
parameter value
```

where parameter is the name of the parameter to be set/examined and value (if specified) is the new value to give the parameter. If value is not specified, the current value of parameter is printed.

EXPLANATION
Cvt alters various TNIX kernel parameters. TNIX must be reconfigured whenever you change the size of the root filesystem (the one beginning on fixed disk 0). The following table lists the parameters that you must set when you change the size of the root filesystem:

Root Filesystem	device name	swplo	nswap	swapdev	pipedev
13.6 MByte	/dev/rhd0	24360	2250	0 0	0 0
35.6 MByte	/dev/rhd0	65000	4576	0 0	0 0
71.2 MByte	/dev/rhd01	134576	4576	0 8	0 8
106.8 MByte	/dev/rhd02	204152	4576	0 16	0 16
142.4 MByte	/dev/rhd03	273728	4576	0 24	0 24

It is also desirable to reconfigure to take full advantage of newly extended memory or to tune the system for optimal performance.

Cvt is available both as a standalone command (to be used when a kernel reconfiguration must be synchronized with a change in the size of the root filesystem) and as a normal command (to be used in other circumstances). See **standalone**(8).

SCRIPT PARAMETERS

You can specify the following parameters:

buffer	*Value* is the total number of cache buffers.
inode	*Value* is the number of inode table entries -- the maximum number of inodes that can be in use at any one time.
file	*Value* is the number of open file table entries -- the maximum number of files that can be open at any one time.
proc	*Value* is the number of process table entries -- the maximum number of active processes.
mount	*Value* is the number of mount table entries -- the maximum number of devices that can be mounted at any one time.
text	*Value* is the number of text table entries -- the maximum number of read-only text segments that can be active at any one time.
swapmap	*Value* is the number of swap map entries -- the maximum allowable fragmentation of the available swap space. Each discontiguous section of unused swap space requires one swap map entry.
coremap	*Value* is the number of core map entries -- the maximum allowable fragmentation of the available memory. Each discontiguous section of unused user memory requires one core map entry.
rootdev	*Value* is the major and minor device numbers of the root device -- two integers separated by blanks or tabs.
swapdev	*Value* is the major and minor device numbers of the swap device -- two integers separated by blanks or tabs.
pipedev	*Value* is the major and minor device numbers of the pipe device (the place to store data piped between processes) -- two integers separated by blanks or tabs.
nswap	*Value* is the size of the swap space, in blocks.
swplo	*Value* is the block number of the first block of the swap space.
timezone	*Value* is the offset in minutes from Greenwich Mean Time to the current timezone.
tzname	*Value* is the three-character name of the current timezone.

tzdstname
: _Value_ is the three-character name of the current timezone during daylight savings time.

tzflag
: _Value_ is either '1', indicating that the current timezone follows the U.S. rules for daylight time, or '0', indicating that it does not.

*
: Display the values of all of the above parameters. No _value_ field is allowed for this _parameter._

ERROR MESSAGES

preposterous cvt address
: if _tnixname_ is not in the appropriate format.

SYSTEM TOO BIG: number BYTES OVER
: if the new configuration will not fit in the available memory.

Other messages are self-explanatory.

FILES

/tnix	-- the default tnix kernel
/etc/cvtscript	-- directory containing distributed **cvt** scripts

SEE ALSO

cvt(5), **standalone**(8)

NOTES

The 8560 should be shut down and restarted after you reconfigure "/tnix", so that the new TNIX will be loaded into memory.

Because **cvt** deals with the low-level configuration parameters of the kernel, it is possible to generate a kernel that will not run properly. The use of configuration scripts is encouraged.

If you change the size of your root filesystem, the **cvt** swplo parameter **must** match the **mkfs** filesystem size parameter that you supply to the stand-alone version of **mkfs**.

DF(8)

NAME

df – disk free

SYNTAX

df [*filesystem*] ...

DESCRIPTION

Df prints out the number of free blocks available on the *filesystem(s).* If no file system is specified, the free space on all of the normally mounted file systems is printed.

DUMP(8)

dump - incremental file system dump

SYNTAX
dump [level[u] logical-device] [**-f** dumpdev] [**-s** block-size] [**-m**] [**-v**]

PARAMETERS

u	If the dump completes successfully, write the date of the beginning of the dump into the /etc/ddate file. This file records a separate date for each logical device and for each **dump** level.
level	A single, decimal digit (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9) that specifies the "dump level". All files modified since the last dump of a lesser level from a logical-device, as determined by the date stored in the file /etc/ddate, will be dumped. If no date is determined by the "level", the beginning of time is assumed; thus the option **0** causes the entire logical device to be dumped.
logical-device	Specifies the pathname of the device containing the filesystem you want to dump. The pathname must begin with /dev/. For example, if the root filesystem is located on the 8560's internal disk, then the logical device name of the root filesystem is /dev/rhd0.
-f dumpdev	Specifies the complete pathname of the dump device (the device upon which the dump will be written). The pathname must begin with /dev/. If you don't specify dumpdev, **dump** dumps to /dev/rfd0 (the flexible disk drive).
-s block-size	Specifies the size (in blocks) of the dump medium. If you don't specify size, **dump** assumes that you're dumping to 1995-block volumes.
-m	Causes **dump** to write and read back a test record on the target medium before attempting a dump. This option gives you the opportunity to replace a bad or improperly mounted volume without restarting the dump procedure.
-v	Causes **dump** to verify each volume can be read, after the volume is completely written. When you include **-v** in the command line, **dump** automatically includes the **-m** option as well.

Using **dump** without specifying any arguments is equivalent to typing:

```
# dump 9u /dev/rhd0 -f /dev/rfd0 -s 1995 <CR>
```

EXPLANATION

Dump copies all files on logical-device that have changed after a certain date to flexible disk(s) or magnetic tape(s).

Dump tells you the current date, the earliest date the dump starts at (the earliest possible date if this is a level 0 dump), the devices being dumped from and to, and when each of the four internal steps of the dump procedure begin. If the dump requires more than one volume (a single flexible disk or magnetic tape), **dump** will ask you to change volumes; after changing the volume, press the RETURN key. The first volume should be installed before you invoke **dump**.

PERFORMING A FULL BACKUP

This procedure tells how to create a full backup of your system, referred to as a "level-0 dump", using double-sided, double-density flexible disks (dump allows you to use single-density flexible disks, magnetic tapes, magnetic cartridges, etc. by specifying the device that **dump** writes to and/or the maximum number of blocks on each dump volume).

1. Log into the "root" account on the system console:

```
     login: root
            [you may be asked for a password]
```

2. Make sure that you are the only person logged into your system.
3. Determine how many formatted, double-density, double-sided flexible disks you will need in order to perform a full backup of your system. To do this, open the flexible disk drive's door, then enter the following **dump** command to determine how many disks you will need:

```
     # dump 0 /dev/rhd0

               [ several messages are displayed before dump
                 prints the following message ]

     dump: estimated 16259 blocks on 9 volume(s)

               [ according to this example, you should have
                 9 formatted disks available before
                 starting the backup procedure ]
                         .
                         .
                         .
               [ several error messages are displayed because the
                 flexible disk drive door is open ]
```

(To format a flexible disk, cover the disk's write-protect slot, insert the disk into the disk drive, then type the **format** command.)

4. To start the backup procedure, insert a formatted disk into the disk drive, then type the following command:

    ```
    # dump 0u /dev/rhd0
    ```

    The **dump** command will guide you through the backup process, asking you to insert new disks into the disk drive when necessary. If an additional disk is necessary, **dump** prints the following message:

    ```
    dump: change volumes, current inode = XXX
                  [ XXX is the starting inode number
                    of the next volume. ]
    ```

    Before inserting the new disk into the disk drive, label it with the following information:

    - date
    - dump level (0)
    - volume number (the first disk is volume 1, the second is volume 2, and so on)
    - starting inode number (the starting inode number for volume 1 is "0")

Once you have labeled the disk with the above information, insert it into the disk drive, close the drive's door, then press the RETURN key.

PERFORMING INCREMENTAL DUMPS

The 10 dump levels allow you to perform a complete backup of your filesystem with a level 0 dump, followed by fast, incremental dumps (levels 1--9) that store only those files that have changed since the last backup (incremental or complete).

When performing incremental backups, the rule of thumb is to perform the incremental backup at the same dump level until the incremental backup at that dump level takes longer than 15 minutes or more than one dump volume (one flexible diskette, for example). At that point, proceed to the next higher dump level. Thus, a level 1 dump is followed by either a level 1 or level 2 dump; a level 2 dump is always followed by a level 2 or level 3 dump, and so on up to level 9. When a level 9 dump takes longer than 15 minutes or more than one dump volume, you perform a complete backup (a level 0 dump), and start over again.

Let's assume that your company has a corporate archive policy that requires you to perform a complete backup of the data on your system at the beginning of each month for permanent, off-line storage. Let's also assume that you want a fast method of backing up your system once each day, so that you never have to worry about losing more than one day's worth of data. Figure 1 presents an algorithm for performing an incremental backup. Let's look at each step:

1. At the beginning of each month, perform a level 0 dump of your entire system. The level 0 dump copies all data on a single filesystem onto the backup media (flexible disk or magnetic tape).

2. On the next day, perform an incremental backup using dump level 1. This dump copies all files that have changed since the last level 0 dump onto the backup media.

3. On the next day, perform an incremental backup at the same dump level as the last incremental backup that you performed. This dump copies all files that have changed since the last incremental backup performed at a lesser dump level. Repeat this step each day until the incremental backup takes longer than 15 minutes or more than one dump volume to complete. Then advance to step 4.

4. If have been performing a level 9 dump, perform a level 0 dump, then return to step 2. If you have not been performing a level 9 dump, perform the next level dump, then return to step 3.

Continue the above process until the beginning of the next month, when when you start from step 1 again. The following chart illustrates this process:

```
                      +----------------------+
                      |Beginning of the Month|
                      +----------------------+
                                 |
                                 V
                           +----------+
          +--------------->|level := 0|
          |                +----------+
          |                      |
          |                      V
          |            +-------------------+
          |            |perform_dump(level)|
          |            +-------------------+
          |                      |
          |                      V
          |            +-------------------+
          |            | level := level + 1|<--------------+
          |            +-------------------+               |
          |                      |                         |
          |                      V                         |
          |            +-------------------+               |
          |            |perform_dump(level)|<-----------+  |
          |            +-------------------+            |  |
          |                      |                      |  |
          |                      V                      |  |
          |  ----------------------------------------+  |  |
          |  |                  is                   |  |  |
          |  |(time_to_perform_dump(level) > 15 Min.)|  |  |
          |  |                  or                   |--+  |
          |  |(number_of_volumes_dumped(level) > 1) ?| no  |
          |  +---------------------------------------+     |
          |                      | yes                     |
          |                      V                         |
          |               +-------------+                  |
          |          yes  |     is      | no               |
          +---------------|(level = 9) ?|------------------+
                          +-------------+
```

Figure 1. An algorithm for performing incremental dumps.

EXAMPLES

To perform a complete dump of the root filesystem onto flexible disks, updating /etc/ddate, type:

```
# dump 0u /dev/rhd0 <CR>
```

To perform a complete dump of the non-root filesystem /dev/rhd23 onto flexible disks, updating /etc/ddate, type:

```
# dump 0u /dev/rhd23 <CR>
```

To do a full dump of /dev/rhd1, updating /etc/ddate, onto 10,000-block tapes (using /dev/mt0), verifying each volume before and after it is written, type:

```
    # dump 0u /dev/rhd1 -f /dev/mt0 -s 10000 -v <CR>
```

To dump all files on /dev/rhd0 that have changed since the last level 0 dump, updating /etc/ddate, onto 20,000 block tapes (using /dev/mt0), verifying each volume before it is written, type:

```
    # dump 1u /dev/rhd0 -f /dev/mt0 -s 20000 -m <CR>
```

To dump all files on /dev/rhd0 that have changes since the last level 7 or earlier dump, onto flexible disks, updating /etc/ddate, type:

```
    #  dump 8u /dev/rhd0 <CR>
```

To do a full dump of the filesystem on /dev/rhd0, updating /etc/ddate, onto single-sided, single-density flexible disks, type:

```
    # dump 0u /dev/rhd0 -s 500 <CR>
```

FILES
/etc/ddate -- record **dump** dates of filesystem/level.

SEE ALSO
dump(5), **dumpdir**(8), **restor**(8).

NOTES
Read errors on logical-device are ignored. Write errors on the dump volume(s) are usually fatal.

Single-density flexible disks have a 500 block capacity. Double-density flexible disks have a 1995 block capacity.

A magnetic tape's block capacity depends on the tape length (in feet) and the **blocking factor** (the number of blocks-per-record). The following tables show the approximate maximum block capacities **dump** allows for different tape lengths (to compensate for varying tape lengths, use a smaller number):

Reel-to-Reel Magnetic Tape Drive			
	Tape Length (Feet)		
Blocking Factor	600	1200	2400
8	18016	35128	70944
7	17125	34087	68740
6	16469	32780	66106
5 (*a)	15930	30940	62740
4	14521	28903	58287
3	12985	25846	52122
2	10718	21333	43021
1	7034	14000	28339

(*a) **dump** uses a blocking factor of 5.

Cartridge Magnetic Tape Drive		
	Tape Length (Feet)	
Blocking Factor	300	450
8	5370	8060
7	5320	7980
6	5280	7920
5 (*b)	5180	7770
4	5080	7630
3	4880	7320
2	4550	6820
1	3770	5650

(*b) **dump** uses a blocking factor of 5.

DUMPDIR(8)

NAME

dumpdir—print the names of files on dump diskette(s)

SYNTAX

dumpdir

DESCRIPTION

Dumpdir reads flexible diskette(s) dumped with the *dump* command. **Dumpdir** lists the names and inode numbers of all the files and directories on the diskette(s).

If the dump extends over more than one diskette, it may ask you to change diskettes. Press RETURN after you have inserted the next diskette.

FILES

rst*

SEE ALSO

dump*(8)*, **restor***(8)*

FORMAT(8)

NAME

format—write fixed disk format in standalone mode

SYNTAX

format *drive*

DESCRIPTION

Format writes the appropriate formatting information to prepare the fixed disk *drive* (physical drive number in the range 0—3, inclusive) for subsequent data storage. See **format***(1)* for information on formatting flexible disks.

This version of the **format** command is executed by booting the "standalone utilities" diskette. Type

```
fbr filename to boot> format <CR>
```

Respond with the *drive* number to format when queried.

Format will search for bad disk blocks and attempt to make them invisible by telling the disk controller to use the spare block available on each track. If **format** reports that there is more than one bad block in a track, first use **mkfs***(8)* to buide a new filesystem, then run **syschk***(8)* to put the additional bad blocks in the bad block file.

SEE ALSO

format*(1)*, **mkfs***(8)*, **standalone***(8)*, **syschk***(8)*

GETTY(8)

NAME

getty – set typewriter mode

SYNTAX

/etc/getty [*char*]

DESCRIPTION

Getty is invoked by **init (8)** immediately after a typewriter comes on line. It reads the user's login name and calls **login (1)** with the name as argument. While reading the name **getty** attempts to adapt the system to the speed and type of terminal being used.

Init calls **getty** with a single character argument taken from the **ttys (5)** file entry for the terminal line. This argument determines a sequence of line speeds through which **getty** cycles.

The user's name is terminated by a new-line or carriage-return character. In the second case CRMOD mode is set (see **ioctl (2)**).

If the terminal's 'break' key is depressed, **getty** cycles to the next speed appropriate to the type of line and prints the greeting message again.

Finally, login is called with the user's name as argument.

The following arguments from the **ttys** file are understood.

0	Cycles through 9600-4800-2400-1200-600-300 baud. Useful as a default for a variety of on-line terminals.
1	Same as '0' but starting at 4800 baud.
2	Same as '0' but starting at 2400 baud.
3	Same as '0' but starting at 1200 baud.
4	Same as '0' but starting at 600 baud.
5	Same as '0' but starting at 300 baud.
6	Starts at 1200 baud, cycles to 300 and back. Useful with 212 datasets where most terminals run at 1200 speed.
7	Same as '6' but starts at 300.
A	Intended for on-line CRT terminals such as TEK CT8500 at 9600 baud.
B	Same as **A** except 4800 baud.
C	Same as **A** except 2400 baud.
D	Same as **A** except 1200 baud.
E	Same as **A** except 600 baud.
F	Same as **A** except 300 baud.
a-f	Same as **A-F** except HSI protocol will be forced on line even if jumped for RS-232.

SEE ALSO

init(8), login(1), ioctl(2), ttys(5)

INIT(8)

NAME

init, rc – process control initialization

SYNTAX

/etc/init
/etc/rc

DESCRIPTION

Init is invoked as the last step of the boot procedure. Generally its role is to create a process for each typewriter on which a user may log in.

First **init** invokes a shell with input taken from the file **/etc/rc** . This command file performs housekeeping like removing temporary files, mounting file systems, and starting daemons.

Then **init** reads the file **/etc/ttys** and forks several times to create a process for each typewriter specified in the file. Each of these processes opens the appropriate typewriter for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input, output and error files. Opening the typewriter may involve a delay, since the **open** is not completed until someone is dialed up and carrier established on the channel. Then **/etc/getty** is called with argument as specified by the second character of the **ttys** file line. **Getty** reads the user's name and invokes **login** (1) to log in the user and execute the shell.

Ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of **init** , which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp* , which records current users, and makes an entry in */usr/adm/wtmp* , which maintains a history of logins and logouts. Then the appropriate typewriter is reopened and **getty** is reinvoked.

Init catches the hangup signal SIGHUP and interprets it to mean that the system should be brought from multi user to single user. Use 'kill -1 1' to send the hangup signal.

Init also catches the interrupt signal SIGINT. This signal is interpreted to mean that **init** is to re-read /etc/ttys and close any files that have been disabled. It will also open any NEW files that have been just opened. This allows you to reset your terminal configuration on the fly. Use 'kill -2 1' to send the interrupt signal.

FILES

/dev/tty?, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), getty(8)

[This page intentionally left blank.]

INSTALL(8)

NAME

install—install software

SYNTAX

install

DESCRIPTION

Install installs new or updated software supplied on a TEKTRONIX software distribution diskette on your 8560. To install TEKTRONIX-supplied software on your 8560:

1. Log in to the root account on your system console.
2. Make sure that you are the only one logged in to your 8560.
3. Change your current directory to the "/" directory. Type:

   ```
   # cd /  <CR>
   ```

   Insert the Software Distribution Diskette into the 8560's flexible disk drive and close the drive door.
4. Install the software by typing:

   ```
   # install  <CR>
   ```

5. When the "#" prompt is displayed, the software is installed.
6. Remove the Software Distribution Diskette from the flexible disk drive.
7. Shut down your 8560 by typing:

   ```
   # shutdown  <CR>
   ```

8. Reboot your 8560 and resume normal operations.

LPD(8)

NAME

lp1d, lp2d – line printer daemons

SYNTAX

/usr/lib/lp?d

DESCRIPTION

Each **lp?d** is the daemon for a line printer, responsible for printing files spooled to that printer. Each **lp?d** uses the directory */usr/spool/lp?* to communicate with the corresponding spooler (**lp?r**). The file **lock** in that directory is used to prevent two daemons from becoming active simultaneously. After the program has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with **df** Each such file is submitted as a job by the spooler (**lp?r**). Each line of a job file begins with a key character to specify what to do with the remainder of the line.

L	is followed by the decimal uid and gid of the user who spooled this job. This line causes the banner page for this job to be printed.
B	specifies that the rest of the line is the name of a file to be printed.
F	is the same as **B** except a form feed is prepended to the file.
U	specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.
M	is followed by a user ID; after the job is sent, a message is mailed to the user via the **mail (1)** command to verify the sending of the job.

Any error encountered will cause the daemon to wait and start over. This means that an improperly constructed **df** file may cause the same job to be submitted repeatedly.

The appropriate **Lp?d** is automatically initiated by the line printer command, **lp1r** or **lp2r**.

Each daemon writes to the corresponding /dev/lp? file. To provide a flexible line printer interface, the /dev/lp? files are actually links to either a /dev/aux ("line printer" port) or a /dev/tty ("terminal" port). Perform the following procedure as the superuser to move the line printer link from an aux device to a tty:

Choose what /dev/tty?? is going to be the new line printer. For example, choose /dev/tty03 as the new /dev/lp1.

change the file **/etc/ttys** so that there will be no login run for the terminal (tty03). (see **ttys(5)**)

Change the lp link to the new tty from the old aux. For example:

```
rm /dev/lp1
ln /dev/tty03 /dev/lp1
```

Move your line printer plug from the "line printer 1" port to the "HSI I/O -3" port.

If you choose to move the line printer back to the aux port, reverse the above process.

To restart a daemon (in the case of hardware or software malfunction), it is necessary to first kill the old daemon (if still alive), and remove the lock file before initiating the new daemon. This is done automatically when the system is brought up, by **/etc/rc** , in case there were any jobs left in the spooling directory when the system last went down.

FILES

/usr/spool/lp? spool area for a line printer daemon
/etc/passwd to get information required to print the banner page.
/dev/lp1 /dev/lp2 line printer devices

SEE ALSO

lpr(1)
banproto(5)
slp(1)
stty(1)

MAKEKEY(8)

NAME

makekey – generate encryption key

SYNTAX

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e. to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key.*

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs that perform encryption (e.g. *ed* and *crypt (1)).* Usually its input and output will be pipes.

SEE ALSO

crypt(1), ed(1)

MKBOOT(8)

NAME

mkboot—copy a boot block to the fixed disk in standalone mode

SYNTAX

mkboot

DESCRIPTION

Mkboot installs a boot block on the fixed disk. To invoke **mkboot**, insert the standalone utilities diskette into the flexible disk drive, boot the 8560, then type:

```
fbr filename to boot> mkboot <CR>
```

Once invoked, **mkboot** searches the **fbr***(1)* format flexible disk for a file named **pmuboot**. If such a file exists, its first block is copied to the first block (block 0) of the fixed disk.

DIAGNOSTICS

'cannot find pmuboot' if the boot file is not on the standalone utilities disk; other diagnostics similar to those generated by **fbr***(1)*.

SEE ALSO

fbr*(1)*, **init***(8)* **standalone***(8)*

MKFS(8)

mkfs - construct a file system

SYNTAX

mkfs logical-device blocks

EXAMPLES

```
fbr filename to boot> mkfs
mkfs arguments: 65000
```

Creates a root filesystem on /dev/hd0, the 8560's internal disk. Execute this command while operating the 8560 in standalone mode.

```
# mkfs /dev/hd1 69576
```

Creates a filesystem on the /dev/hd1 8503 Disk Expansion Unit. Execute this command while operating the 8560 in multi-user mode.

PARAMETERS

logical-device
: The software driver that interfaces TNIX to the physical disk drive. For example, /dev/hd0 is the name of the logical device that is associated with the root ("/") filesystem.

blocks
: The number of blocks on the logical device. The following table shows you the number of blocks per logical device:

Disk Drive Capacity (MBytes)	Number of Disk Drives	Number of Blocks (root filesystem)	Number of Blocks (non-root filesystem)
13.6	1	24360	not supported
35.6	1	65000	69576
35.6	2	134576	139576
35.6	3	204152	208728
35.6	4	273728	not supported

EXPLANATION

Mkfs constructs an empty file system consisting of the given decimal number of blocks, on the given logical-device, and announces the number of inodes it created for the filesystem (based on the number of blocks).

Mkfs is available as either a standalone command or a normal command. The standalone version is used for creating the root filesystem (the one beginning on fixed disk drive 0). The normal command version is used for creating any other filesystem. See **standalone**(8) for instructions on running standalone programs.

The standalone version prompts for blks. It assumes that the filesystem begins on fixed disk drive zero and occupies as many fixed disks as are required.

SEE ALSO

cvt(8), **dir**(5), **filsys**(5), **mkboot**(8), **restor**(8) **standalone**(8), **syschk**(8)

NOTES

Mkfs will destroy any data on the specified logical device.

Mkfs should not be performed on a mounted filesystem.

If you change the size of the root filesystem (the one beginning on fixed disk 0), you must reconfigure your system using the **cvt**(8) program.

MKGROUP(8)

NAME

mkgroup – add new group to system

SYNTAX

mkgroup *groupname username* [*username*] ...
mkgroup -r *groupname*

DESCRIPTION

This command will add the new *groupname* to the system, and make the *username(s)* members of that group. The -r option will remove the named *groupname* from the system.

DIAGNOSTICS

Groupname already exists, *username* does not exist

SEE ALSO

mkuser(8), group(5), passwd(5)

NOTES

Mkgroup needs to open the passwd and group files exclusively.

MKUSER(8)

NAME

mkuser – install new user on system or modify existing user

SYNTAX

mkuser [-r] *username* [*groupname*] ...

DESCRIPTION

If the given *username* does not already exist, this command will install *username* on the system as a member of the *groupname(s)* given. If *username* already exists, each existing parameter will be displayed and may be optionally replaced.

The -r option will remove the user from the named *groupnames* with no change to the password file . If no *groupname* is given the user will be removed from the password file, and from any groups in the group file.

In the first case, this includes creating the directory */usr/username* as the default directory, adding *username* to the password file, and adding *username* to the *groupname* group file(s). *Username* will have a null password and will be set up to execute the default shell when he logs in.

If *username* did exist, the default group membership and the comment field will be displayed and optionally changed. These parameters may be changed by typing a new value for them. A carriage return will leave them unchanged.

DIAGNOSTICS

Groupname does not exist, invalid parameter

SEE ALSO

mkgroup(1), passwd(5), group(5)

NOTES

Mkuser needs to open the passwd and group files exclusively.

FILES

/etc/passwd
/etc/group
/etc/grplock
/tmp/pwdXXXXXX

[This page intentionally left blank.]

MOUNT(8)

NAME

mount, umount—mount and dismount file system

SYNTAX

/etc/mount [*device filesystem* [**-r**]]
/etc/umount *filesystem*

DESCRIPTION

Mount announces to the system that a removable filesystem, *device*, is present on the 8560. Data can be written to and read from *filesystem* after it is mounted. The file *filesystem* must exist before using **mount**. *Filesystem* must also be a directory (unless the root of the mounted file system is not a directory). *Filesystem* becomes the name of the newly mounted root. If **mount** is invoked without an argument, it prints the table of currently mounted filesystems available to the **mount** and **umount** commands.

Umount announces to the system that the removable file system previously mounted on device *filesystem* is to be removed.

OPTIONS

–r The file system is to be mounted read-only.

EXAMPLE

The following example mounts the device "/dev/hd23" as the filesystem called "/usr1":

```
# mount /dev/hd23 /usr1 <CR>
```

All files on device "/dev/hd23" are located in a subdirectory of "/usr1".

FILES

/etc/mtab—table of mountable devices

SEE ALSO

mount*(2)*, **mtab***(5)*

NOTES

Physically write-protected filesystems must be mounted read-only; otherwise, errors will occur when access times are updated, whether or not any explicit write is attempted.

Mounting a device that does not contain a valid filesystem will crash the system.

Flexible disks cannot be mounted as filesystems.

RESTOR(8)

restor - restore data from backups

SYNTAX

restor [**-f** logical-device] **-rxt** [filename ...]

or

restor [**-f mt(0,**addr**)**]

or

restor [**-f mt(1,**addr**)**]

EXAMPLES

```
# restor -f /dev/rfd0 -t
```

Print the date and dump level of the dump volume in the flexible disk drive.

```
# umount /usr1; restor -f /dev/rfd0 -r /usr1
```

Unmounts the /usr1 filesystem, then copies the contents of the dump volume(s) into the /usr1 filesystem.

```
fbr filename to boot> restor
restor arguments: -f mt(0,1)
```

Restores the root filesystem from a Dylon Corporation 2001/9001 7/9 track magnetic tape unit located at GPIB primary address 1. Execute this command while the system is in stand-alone mode.

PARAMETERS

-f device
: Specifies that device is the device containing the dump volume. The device name must begin with /dev/. If you don't specify device, **restor** assumes that /dev/rfd0 contains the dump volume.

-r
: Writes the contents of the dump volume(s) into the filesystem specified by filename. This operation recreates a filesystem from either full or incremental backups. You should unmount filename before performing the restore operation. See **mount**(8) for information about mounting and unmounting filesystems.

  You cannot use the **-r** option to restore the root filesystem -- you must use the standalone version of **restor** (described later) to restore the root filesystem from backups.

-x
: Extracts each filename from the dump volume, and places filename's contents into a file in the current directory. The resulting file is named with filename's inode number.

-t
: Print the date the volume was written and the date the filesystem was dumped.

EXPLANATION

Restor is used to read flexible disks or magnetic tapes dumped with the **dump**(8) command. (These flexible disks or magnetic tapes are known generically as dump volumes.) The option that you select specifies what action is to be taken.

RESTORING A COMPLETE FILESYSTEM

To restore a complete filesystem from backups, perform the following steps:

1. Restore the most recent complete (dump level 0) backup.

2. Restore the lowest-level incremental backup that is more recent than the level 0 backup. For example, if level 1, 2, and 3 backups have been performed since the latest level 0 backup, restore the most recent level 1 backup.

3. Restore the next-higher-level backup that is more recent than the previous incremental backup restored. Repeat this step until all incremental backups that meet this condition have been restored.

The following algorithm shows how to perform a restore operation:

```
level := 0;
restore(level);
last_level := level;

for level := 1 to 9 do
   begin
      if (date_of(level) > date_of(last_level)) then
         begin
            restore(level);
            last_level := level;
         end
   end
```

If you are restoring a non-root filesystem, the restore operation can be performed while TNIX is running. Otherwise, you must use the standalone **restor** program to restore the root filesystem.

RESTORING THE ROOT FILESYSTEM

There is a standalone version of **restor** available for restoring a complete or incremental dump to the root filesystem (the one that includes fixed disk drive 0). The standalone version is invoked by booting the "standalone utilities" disk and replying **restor** when prompted for the filename to boot.

When **restor** prompts you for arguments, it expects a carriage return if you are using flexible disks, or a response in the form **-f** **mt**(type,address) if you are using magnetic tape. Type must be either a "0" (indicating a Dylon 2001/9001 7/9 track Magnetic Tape System) or a "1" (indicating a Dylon 4000A Digital Cartridge Recording System); address is the tape drive's primary address. For example, if you have Dylon 9-track tape drive at GPIB primary address 3, respond to the prompt with "**mt(0,3)<CR>**".

Restor will issue an appropriate warning message before starting the restore operation, to allow the first volume to be installed. If more than one volume is involved in the dump, **restor** will tell you when to mount the next volume.

EXAMPLES

This example is a procedure for restoring files from either a full or incremental backup.

Assume that you need to restore all files belonging to user bjorng. The files may be on any or all of three dump volumes (in this case, flexible disks). The dump volumes are labeled by inode numbers: volume I contains 2--1006, volume II contains 1007--4628, and volume III contains 4629--7268.

Your first task is to find all files belonging to bjorng on the dump set, and to sort the filenames in order of increasing inode numbers. To make things easy, you can put the sorted list into the file list. One command line performs the task (be sure to put the first dump volume into the drive):

```
    $ dumpdir -f /dev/rfd0 | grep /usr/bjorng | sort +2 > list<CR>
```

The file list now contains the following text:

```
            dumpdir:        /usr/bjorng/file.cabinet        237
            dumpdir:        /usr/bjorng/file.single         516
            dumpdir:        /usr/bjorng/file.rasp           2813
            dumpdir:        /usr/bjorng/file.nail           2815
            dumpdir:        /usr/bjorng/file.round          6972
```

According to list, the dump set contains five files that belong to bjorng. For each entry in list, the file's inode number follows the file's name.

Your next task is to divide the filenames in list among three new files, vol.1, vol.2, and vol.3. These new files correspond to the three dump volumes, and contain the names of bjorng's files that reside on each volume. That is, you want vol.1 to contain

```
    /usr/bjorng/file.cabinet
    /usr/bjorng/file.single
```

and vol.2 to contain:

```
    /usr/bjorng/file.rasp
    /usr/bjorng/file.nail
```

and vol.3 to contain:

```
    /usr/bjorng/file.round
```

Use the following command lines to edit list and create vol.1, vol.2, and vol.3:

```
    $ ed list<CR>
    g/n.*\/usr/s//\/usr/<CR>
```

```
g/      .*$/s///g<CR>
w<CR>
1,2w!vol.1<CR>
3,4w!vol.2<CR>
5w!vol.3<CR>
q<CR>
$
```

Now, you can use vol.1, vol.2, and vol.3 with **restor** to write bjorng's files into the current directory. Install the first dump volume into the flexible disk drive, and type:

```
$ restor -xf /dev/rfd0 `cat vol.1`<CR>
```

Install the second dump volume, and type:

```
$ restor -xf /dev/rfd0 `cat vol.2`<CR>
```

Install the third dump volume, and type:

```
$ restor -xf /dev/rfd0 `cat vol.3`<CR>
```

The current directory now contains bjorng's five files. You can move these files to whatever other directory you wish.

FILES

rst*	temporary files used by **restor**

SEE ALSO

dump(8), **mkfs**(8), **dumpdir**(8), **mount**(8)

ERROR MESSAGES

There are various diagnostics involved with reading the dump volume and writing the hard disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the process extends over more than one volume, **restor** will ask you to change volumes. Press RETURN when the next volume has been mounted.

NOTES

Restoring a complete dump to a filesystem (via a standalone **restor** for the root filesystem or a "**restor** r ..." to any other filesystem) will destroy any information previously on that filesystem.

SHUTDOWN(8)

NAME

shutdown – bring the system down gracefully

SYNTAX

shutdown

DESCRIPTION

Shutdown will kill any outstanding user and daemon processes if there is not more than one user logged in. After killing all processes but the scheduler and initialization processes, it does a *sync(2)* , and writes to the console terminal that the system may be turned off.

If there is more than one user logged in, **shutdown** will report this, list who is logged in, and say to try again later. **Shutdown** will also complain if the user running **shutdown** is not the super user.

STANDALONE(8)

NAME

standalone—how to run standalone utilities

DESCRIPTION

There are a number of **standalone** programs, which perform functions that cannot or should not be performed while **TNIX** is running. These programs are located on the TNIX Standalone Utilities Diskette. This diskette uses the **fbr***(1)* disk format. To invoke one of these programs:

1. If **TNIX** is running, shut the system down —log in to the root account and type:
   ```
   # shutdown <CR>
   ```
2. Put the Standalone Utilities Diskette in the flexible disk drive and toggle the RESTART switch.
3. The system will display:
   ```
   fbr filename to boot>
   ```
   Type the name of the particular **standalone** program that you want to run (for example, **syschk**). If the program is not on the diskette, the system will again display:
   ```
   fbr filename to boot>
   ```
4. The **standalone** program will ask for any additional information it requires (for example, syschk displays:
   ```
   syschk flags:
   ```
5. When the program has finished executing, the system will again display:
   ```
   fbr filename to boot>
   ```
6. To return to normal operations, take the standalone utilities diskette out of the disk drive and reboot the system.

SEE ALSO

cvt*(8)*, **format***(8)*, **mkboot***(8)*, **mkfs***(8)*, **restor***(8)*, **syschk***(8)*

[This page intentionally left blank.]

SYNC(8)

NAME

sync – update the super block

SYNTAX

sync

DESCRIPTION

Sync executes the **sync** system primitive. If the system is to be stopped, **sync** must be called to insure file system integrity. See **sync (2)** for details.

SEE ALSO

sync(2), update(8)

SYSCHK(8)

NAME

syschk—check (and optionally repair) file system integrity

SYNTAX

syschk [**-bmw**] [**-t** *tempdir*] [*filesystem* ...]

DESCRIPTION

There are two **syschk** programs, a standalone version and a command version. The command version is invoked as shown here; see **standalone***(8)* for information on how to run standalone programs.

For each *filesystem* that you specify, **syschk** performs a number of checks on the integrity of that filesystem. If you do not specify a *filesystem* the root filesystem is checked. The standalone version of **syschk** checks the root filesystem (the filesystem beginning on fixed disk 0). The command version filesystem defaults are listed in the file **/etc/checklist.** (If **/etc/checklist** cannot be read by **syschk, /dev/rhd0** will be used as the default.)

The command version will check any filesystem, or repair any filesystem that is not currently mounted. The standalone version is provided to repair the root filesystem, which is always mounted whenever TNIX is running.

The command version of **syschk** exits with a status of zero (0) if no problems occurred, one*(1)* if some file system corruption was detected, or other values if some other problem occurred (such as invocation of **syschk** with an illegal option). The standalone version completes after displaying the message:

```
syschk  complete. halt or reboot.
```

OPTIONS

-b	Scan for and remove bad blocks from the device. Typing the **-b** option is the same as typing both the **-b** and **-m** options together. **Syschk's** default is to not scan for bad blocks. As many bad blocks as possible (up to one per track) are removed by formatting the bad track(s). **Syschk's** method of formatting tracks preserves any good data that was on the track prior to formatting. Any bad blocks remaining after formatting are allocated to the bad block inode (i-number 1) and deallocated from their original structure.
-m	Allow modifications (repairs) to be performed on the file system. **Syschk's** default is to not attempt to repair any faults it finds in the file system. The filesystem being repaired must be **umounted** before you use the **-m** or **-b** options.
-w	Print warnings. Syschk's default is to not warn of unusual file system conditions that are not necessarily errors.
-t tempdir	Use *tempdir* as a directory for temp files, rather than the default, **/tmp.** This option is ignored by the standalone version.

MESSAGES

The following pages provide a list of **syschk**'s responses to file system corruptions. When a message ends in a question mark, you can choose not to perform the given repair by typing an "n" or <CR>. Any message regarding a particular inode (file) is preceded by the name of the file (as well as can be determined), the size and mode of the file, the names or id's of the file's owner, and the times associated with the file (given in GMT by the standalone **syschk**, local time by the command version). Any message regarding a corrupt link is preceded by the same information (about the directory involved) and the name of the link.

'filename' IS NOT A BLOCK OR CHARACTER DEVICE check regardless?

The filesystem that **syschk** was told to check is not a device. Probably the wrong name was typed when **syschk** was invoked. Syschk is offering to check the file anyway.

TOO MANY BAD BLOCKS ignoring bad block 'block'

There are more bad blocks on the disk than **syschk** can record. The given bad block is not being recorded as bad. This message indicates a serious problem with the disk. Perhaps the disk is not ready for I/O or the disk controller is at fault.

'count' NEW BAD BLOCKS add to bad block inode?

There are blocks on the disk that have become bad since the last addition to the bad block inode. Syschk is offering to allocate the newly bad blocks to the bad block inode, preventing them from being used in the future.

TOO MANY I-NODES ('count')

The superblock (block 0 on the disk device) indicates that there are more inodes than can be represented in an i-number (more than 65535). This is a fatal error.

TOO FEW I-NODES ('count')

The superblock (block 0 on the disk device) indicates that there are fewer than two inodes on the file system. This is a fatal error.

I-NODE AREA OVERFLOWS FILE SYSTEM

The superblock (block 0 on the disk device) indicates that the entire file system is consumed or overflowed by the inode area. This is a fatal error.

BAD SUPERBLOCK UPDATE TIME 'time' update?

The superblock's (block 0 on the disk device) indication of the last time it was changed doesn't make sense. Syschk is offering to replace this value with the current time. This check is omitted by the standalone version.

ROOT I-NODE IS UNALLOCATED

The inode dedicated to the root directory is not allocated. This is a fatal error.

ROOT I-NODE IS NOT A DIRECTORY

The inode dedicated to the root directory is allocated, but is something other than a directory. This is a fatal error.

BAD BLOCK I-NODE IS NOT A REGULAR FILE put one there?

The inode dedicated to storing bad blocks should be an ordinary file. It is not. Syschk is offering to rewrite that inode with one representing an ordinary file.

CANNOT ACCESS BAD BLOCK I-NODE

The bad block file cannot be opened (one possibility is that the inode itself is in a bad block and cannot be read). The previous record of what blocks were bad is ignored.

BAD MODE destroy?

The given file is allocated, but it is not one of the following types: directory, regular file, character special file, block special file. Syschk is offering to deallocate the file, destroying its contents.

BAD SIZE destroy?

The size of the given file does not make sense (is negative or greater than what can be represented). Syschk is offering to deallocate the file, destroying its contents.

DIRECTORY MISALIGNED shrink?

The size of the given directory is not a multiple of the size of a directory entry. Syschk is offering to discard away the partial entry.

REFERS TO A BAD BLOCK zero pointer?

The given file contains a bad block. Syschk is offering to change the file so that it refers to a block of 0's instead of the bad block.
(warning) has 'count' fewer blocks than are required

The given file has "holes" in it - blocks that have never been written to. No action is taken. Because a file can legitimately have such holes, a warning is produced only if the *–w* switch was specified when **syschk** was invoked.

NO '.' ENTRY add one?

The given directory has no entry to refer to itself. Syschk is offering to build one. NOTE: Some programs assume that the "." and ".." entries are the first two entries in each directory. A directory repaired in this way may no longer fit that assumption, and thus may not be treated properly by such programs.

HAS NO '..' ENTRY make one?

The given directory has no entry to refer to its parent directory. Syschk is offering to build one. (See the NOTE under "NO '.' ENTRY".)

I-NUMBER 'number' OUT OF RANGE remove entry?

The given directory entry refers to a nonexistent inode. Syschk is offering to remove the offending link.

ASSOCIATED I-NODE IS UNALLOCATED remove entry?

The given directory entry refers to a deleted file. Syschk is offering to remove the offending link.

'.' ENTRY IS INCORRECT fix?

The "." directory entry does not refer to this directory. Syschk is offering to replace the offending link with a correct one.

'..' ENTRY IS INCORRECT fix?

The ".." directory entry does not refer to the parent of this directory. Syschk is offering to replace the offending link with a correct one.

IS AN ORPHAN connect to /lost+found?

The given file, although allocated, has no directory entries referring to it. Syschk is offering to create a link to this file in the lost and found directory (the directory "lost+found" on the filesystem being repaired). The link created will have the form "Innn" where nnn is the i-number of the orphan file.

CANNOT EXTEND DIRECTORY TO MAKE 'filename'

The given filename cannot be put in the directory. Either the directory cannot be read reliably; or there are no free blocks available for allocation. A subsequent message offers an alternative.

NO /lost+found DIRECTORY

A link cannot be made in the lost and found directory because there is no such directory on the filesystem. Manual intervention is required if you want to recover orphan files. Here is the procedure:

1. Using **syschk**, fix all problems on the filesystem except any dealing with orphan files.
2. Running TNIX normally, create the directory "lost+found" on the appropriate filesystem. For example, the "lost+found" directory on the filesystem mounted as /usr1 would be called "/usr1/lost+found".
3. Run **syschk** again, this time attempting to recover orphan files.

CONNECTION FAILED destroy orphan?

Either the orphan file cannot be linked to, or you typed "n" in response to the "connect to lost+found" message. Syschk is offering to delete the file.

INCORRECT LINK COUNT record='count', actual='count' fix?

The inode record of the number of directory entries referring to it is not correct. Syschk is offering to correct the inode's record.

HAS 'count' BLOCKS OUT OF BOUNDS, 'count' DUPLICATED BLOCKS consider removal?

The given file contains blocks that are also members of other structures. Syschk is offering to remove the file (in terms of total number of block allocation problems on the disk). If you type "y", **syschk** will eventually display the message:

```
If that inode is removed, total out-of bounds will be  count
total duplicates will be  count   remove it?
```

or:

```
If that inode is removed, no block allocation problems will
remain here.  remove it?
```

In either case, typing "y" will delete the file. A file should only be deleted if the deletion would reduce the number of block allocation problems on the disk. The initial "consider removal" message prints the current totals; the later "remove it" message prints the resulting totals.

BAD COUNT IN FREELIST BLOCK rebuild free list?

The free list is corrupt. More precisely, the "number of blocks pointed to by this block" field in a free list block is negative or greater than the maximum allowed. Syschk is offering to replace the free list with a good one.

FREE BLOCK POINTER OUT OF RANGE rebuild free list?

The free list is corrupt. More precisely, one of the block pointers in the free list refers to something other than a data block. Syschk is offering to replace the free list with a good one.

FREELIST CONTAINS A BAD BLOCK rebuild free list?

One of the blocks in the free list cannot be reliably read. Syschk is offering to replace the free list with a good one.

FREELIST DUPLICATES A BLOCK rebuild free list?

The free list is corrupt. More precisely, one block is a member of the free list more than once, or is a member of the free list and some other structure (e.g., a file). Syschk is offering to replace the free list with a good one.

BLOCKS ARE MISSING rebuild free list?

After all filesystem structures have been examined, some blocks remain unaccounted for. Syschk is offering to replace the free list with a good one.

free list is ok. rebuild free list?

Syschk is offering to replace the good free list with a sorted one. Although this action is not necessary to repair a filesystem, sorting the free list may improve performance slightly.

changes were made to the filesystem. check again?

This message indicates that **syschk** has corrected some filesystem problems and that the filesystem should be checked again. A "n" response to this question will avoid the check.

FILES

lost+found	The directory into which orphan files (allocated files that have no links to them) are linked. A lost+found directory should be created under the root directory of each filesystem.
/etc/passwd	The password file.
/etc/group	Used to convert user and group IDs to names.
/etc/checklist	The file of default file systems to check.
/tmp/sysc????	Temp file used by the command version of **syschk**. The standalone version uses for its temp file the last 1001 blocks of the disk on which the root filesystem (the one beginning on fixed disk 0) ends. runs in either of two environments:

NOTES

Syschk runs in one of two environments:

- as a standalone program running to check or repair the root file system, **syschk** uses the swap space (which must reside on the last disk drive occupied by the root filesystem) as temporary storage.
- as a user command running to check the root file system or repair some other file system, **syschk** will create a file under /tmp to store its intermediate data.

Because of the heuristic nature of file system repair, **syschk** can only make reasonable guesses concerning the cause of a set of file system corruptions. The files that **syschk** offers to remove should first be examined to avoid unnecessary file deletion.

Any errors described in this document as "fatal" are severe enough that further checking of the filesystem is pointless - **syschk** cannot repair such a filesystem. The filesystem must be completely restored from the most recent backup, using **restor***(8)*.

Since the i-list is a fixed array on the disk, bad blocks in the i-list cannot be deallocated from it.

Intermittent read errors during checking will give confusing results.

A single corruption may result in several errors, all of which will disappear when the corruption is repaired. For example, a block allocation problem involving a directory may cause many link count errors that will disappear when the block allocation problem is corrected.

SEE ALSO

standalone*(8)*

UPDATE(8)

NAME

update – periodically update the super block

SYNTAX

/etc/update

DESCRIPTION

Update is a program that executes the **sync (2)** primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO

sync(2), sync(1), init(8)

NOTES

With **update** running, if the CPU is halted just as the **sync** is executed, a file system can be damaged.

WALL(8)

NAME

wall – write to all users

SYNTAX

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

'Cannot send to ...' when the open on a user's tty file fails.