CTIX™ OPERATING SYSTEM MANUAL

Version C
Volume 1

This manual was prepared on a Convergent Technologies S/320 Computer System and was printed on an Apple LaserWriter II Laser Printer.

Second Edition (November 1989) 09-02262-01

TABLE OF CONTENTS: VOLUME 1

SUMMARY OF CHANGES

This second edition of the *CTIX Operating System Manual, Version C*, documents the new commands and features of the CTIX operating system for S/Series systems.

Changes to the manual are summarized below.

Volumes 1 and 2

- Revised front matter
- New pages:

  createdev(1M), *masterupd*(1M), *passmgmt*(1M), *pwconv*(1M), *pwunconv*(1M), *rmnttry*(1M), *rumount*(1M), *serstat*(1M).

- Deleted pages:

  apcon(1), *apnum*(1), *catman*(1), *console*(1M), *crup*(1M), *dismount*(1), *keystate*(1M), *man*(1), *mkboot*(1M), *muser*(1M), *ofcli*(1M), *ofcopy*(1M), *ofcpin*(1M), *ofdf*(1M), *ofeditors*(1M), *oflog*(1M), *ofls*(1M), *ofmkdir*(1M), *ofrm*(1M), *pbuf*(1M), *perc*(1), *pmon*(1M), *tc*(1).

Volume 3

- Revised front matter
- New pages:

  nfssys(2).

  getspent(3X), *putspent*(3X).

- Deleted pages:

  excall(2), *exchanges*(2), *exfinal*(2), *exrequest*(2), *exrespond*(2), *exserverq*(2), *exwait*(2), *nfs_getfh*(2), *nfs_svc*(2).

  ofcreate(3X), *ofdir*(3X), *ofopen*(3X), *ofread*(3X), *ofrename*(3X), *ofstatus*(3X), *quadd*(3X), *quremove*(3X), *spawn*(3X), *swapshort*(3X) *wmgetid*(3X) *wmgop*(3X) *wmlayout*(3X) *wmsetid*(3X)

Volume 4

- Revised front matter
- New pages:

  loginlog(4), *queuedefs*(4), *shadow*(4).

  stape(7).
- Deleted pages:

  pilf(5).

  mt(7).

GUIDE TO TECHNICAL DOCUMENTATION

This manual is one of a set that documents the Convergent Family of information processing systems. The set can be grouped as follows:

Hardware

S/Series

MightyFrame VME Ethernet Controller Card Manual
S/80 Installation Manual
S/80 Diagnostics Manual
S/80 Technical Reference Manual
S/80 SCSI/LAN Board Technical Refernce Manual
S/120 Installation Manual
S/220 Installation Manual
S/221 and S/222 Installation Manual
S/280 Installation Guide
S/320 and S/220 Hardware Manual and S/120 Supplement
S/320 and S/640 Installation Manual
S/320 VME Half-Inch Tape Controller Card Manual
S/320 VME SMD Controller Card Manual
S/480 Installation Manual
S/480 Technical Reference Manual
S/640 Technical Reference Manual
S/MT Series Diagnostics Manual
S/Series SCSI Expansion Cabinet Installation Manual
S/MT Series SCSI/RS-232 I/O Expansion Technical Reference
S/MT Series Ethernet Combo Board Technical Reference
S/MT Series I/O Processor (IOP) Expansion Technical Reference
S/MT Series Remote I/O Processor Technical Reference
S/MT Series RS-232-C Expansion Technical Reference
S/MT Series RS-422 Expansion Technical Reference
MPCC VME Communications Controller Card Technical Reference
S/MT Series VME Communications Controller Card Technical Reference
S/MT Series VME Expansion Technical Reference

Graphics Terminal

Graphics Terminal Hardware Manual
Graphics Terminal Installation Manual

Programmable Terminal

Programmable Terminal Hardware Manual
Programmable Terminal Installation Manual
Programmable Terminal Programmer's Guide

Operating Systems

CTIX Operating System Manual, Version C

System Administration

S/Series CTIX Administrator's Guide
CTIX Administration Tools Manual

Program Development

CTAM Applications Programmer's Guide
Programmer's Guide: CTIX Supplement
AT&T UNIX System V Release 3.2 Programmer's Guide.
AT&T UNIX System V Release 3.2 Streams Primer.
AT&T UNIX System V Release 3.2 Streams Programmer's Guide.

Programming Languages

CTIX BASIC Interpreter and Compiler Manual
CTIX FORTRAN Manual
CTIX Enhanced FORTRAN and Pascal Debugger
CTIX Pascal Manual

Communications and Networks

CTIX BSC 2780/3780 RJE Terminal Emulator Manual
CTIX BSC 3270 Terminal Emulator Manual
CTIX Network Administrator's Guide
CTIX Network Programmer's Primer
CTIX SNA 3270 Terminal Emulator Manual
CTIX SNA LU6.2 APPC Server Manual
CTIX SNA Network Gateway Manual
CTIX SNA PU Type 2.1 Network Gateway Manual
CTIX SNA RJE Manual
CTIX X.25 Network Gateway Manual
CTIX X.25 Terminal/Host Adaptor Manual
AT&T UNIX System V Release 3.2 Network Programmer's Guide.

Data Management Facilities

CTIX ISAM Manual

The following section outlines the contents of these manuals.

HARDWARE

S/Series

The *MightyFrame VME Ethernet Controller Card Manual* describes the Ethernet card, which is installed in a card cage of MightyFrame computer systems. The manual provides detailed descriptions of the card (and Ethernet network), software interfaces, and the theory of operation.

The *S/80 Installation Manual* describes how to install and expand the S/80 computer module. In addition, it describes the procedure for attaching peripheral devices and installing hard disk drives.

The *S/80 Diagnostics Manual* describes the tests used to verify the proper operation of all S/MT Series systems. Individual tests for peripheral devices are covered.

The *S/80 Technical Reference Manual* introduces and provides reference material for the S/MT Series S/80 CPU board and memory expansion. This information serves system programmers who need to make additions and/or modifications to the operating system or diagnostics. The manual also serves hardware engineers and technicians who require a functional understanding of the boards and provides a performance description of the S/80 CPU board.

The *S/80 SCSI/LAN Board Technical Reference Manual* provides principles of operation, software interface information, and host software control information for programmers and hardware designers.

The *S/120 Installation Manual* describes procedures for locating, inspecting, and booting the S/120, and for attaching peripherals and installing expansion boards and hard disk drives. It also includes a summary of system status codes and instructions for building required cables.

The *S/220 Installation Manual* describes procedures for locating, inspecting, and booting an S/220 system, and for attaching peripherals and installing expansion boards and hard disk drives. The manual also includes a summary of system status codes and instructions for building required cables.

The *S/221 and S/222 Installation Manual* describes procedures for locating, inspecting, and booting S/221 and S/222 systems, and for attaching peripherals and installing expansion boards and hard disk drives. The manual also includes summaries of system status codes and instructions of building required cables.

The *S/280 Installation Guide* provides instructions for setting up, configuring, and installing options in the S/280, including preparations required for software installation.

The *S/320 and S/220 Hardware Manual* (Vols. 1 and 2) provides information on the functional description, software interface, and theory of operation for the S/320 and S/220 computer systems.

The *S/120 Supplement to the S/320 and S/220 Hardware Manual* provides information on the functional description, software interface, and theory of operation for the S/120.

The *S/320 and S/640 Installation Manual* provides principles of operation, register descriptions, and connector information for programmers and hardware designers. It includes instructions for setting up, configuring, and installing internal options in the S/320 and S/640.

The *S/320 VME Half-Inch Tape Controller Card Manual* describes the Half-Inch Tape Controller card, which is installed in the VME card cage of a S/320 or S/640 computer system. S/320 or S/640 computer system. The manual provides a detailed description of card installation, theory of operation, and software interfaces.

The *S/320 VME SMD Controller Card Manual* describes the SMD Controller card, which is installed in the VME card cage of an S/320 or S/640 computer system. which is installed in the VME card cage of an S/320 or S/640 computer system. The manual provides a detailed description of card installation, theory of operation, and software interfaces.

The *MPCC VME Communications Controller Card Technical Reference Manual* describes the X.21 version of the MPCC card.

The *S/MT Series VME Communications Controller Card Technical Reference Manual* provides hardware functional description and interfacing information for the MPCC X.25 card.

The *S/MT Series VME Expansion Technical Reference Manual* provides hardware functional description and interfacing information for the MPCC X.25 expansion card.

The *S/480 Installation Manual* describes procedures for locating, inspecting, and booting an S/480 system, and for attaching peripherals and installing expansion boards and hard disk drives. The manual also includes summaries of system status codes and instructions for building required cables.

The *S/480 Technical Reference Manual* provides principles of operation, register descriptions, and connector information for programmers and hardware designers.

The *S/640 Technical Reference Manual* provides principles of operation, software interface information, and host software control information for the SCSI-only S/640.

The *S/MT Series Diagnostics Manual* describes how to use diagnostics for models S/120, S/220, S/221, S/222, S/320, S/480, and S/640, and their options.

The *S/Series SCSI Expansion Cabinet Installation Manual* provides instructions for setting up the Mass Cab and connecting it to an S/Series host computer system. The manual, which includes procedures for installing and removing storage devices and connecting to an external SCSI device, serves as a companion to the appropriate S/Series host computer's installation manual.

The *S/MT Series SCSI/RS-232 I/O Expansion Technical Reference Manual* provides principles of operation, register descriptions, and connector information for programmers and hardware designers.

The *S/MT Series Ethernet Combo Board Technical Reference Manual* provides installation information, with a description of the software interface and theory of operation.

The *S/MT Series I/O Processor (IOP) Expansion Technical Reference Manual* contains a description of the software interface and theory of operation.

The *S/MT Series Remote I/O Processor Technical Reference Manual* contains installation information, a description of the software interface, and theory of operation.

The *S/MT Series RS-232-C Expansion Technical Reference Manual* describes both 10-port and 20-port versions of the board. Includes functional descriptions of interrupt handling, processor interface, and I/O operations.

The *S/MT Series RS-422 Expansion Technical Reference Manual* describes the hardware, including clock, bus and channel control. It also covers the onboard line printer interface.

The *MPCC VME Communications Controller Card Technical Reference Manual* describes the X.21 version of the MPCC card.

The *S/MT Series VME Communications Controller Card Technical Reference Manual* Hardware functional description and interfacing information for the MPCC X.25 card.

The *S/MT Series VME Expansion Technical Reference Manual* Hardware functional description and interfacing information for the MPCC X.25 expansion card.

Graphics Terminal

The *Graphics Terminal Hardware Manual* describes the architecture and theory of operation of the Graphics Terminal.

The *Graphics Terminal Installation Manual* describes procedures for installing, powering up, testing, and connecting the Graphics Terminal. The manual also describes how to connect peripherals and how to connect terminals in a cluster.

Programmable Terminal

The *Programmable Terminal Hardware Manual* describes the architecture and theory of operation of the Programmable Terminal.

The *Programmable Terminal Installation Manual* describes procedures for installing, powering up, testing, and connecting the Programmable Terminal. The manual also describes how to connect peripherals and how to connect terminals in a cluster.

The *Programmable Terminal Programmer's Guide* describes the terminal's boot ROM software and programmable functions, including text display, interchangeable hard fonts, and local editing.

OPERATING SYSTEMS

The *CTIX Operating System Manual, Version C*, describes Releases 6.x of the CTIX operating system, Release 6.x of CTIX is derived from the UNIX System V operating system. The manual describes CTIX commands, application programs, system calls, library subroutines, special files, file formats, games, miscellaneous facilities, and system maintenance procedures. This manual is the starting point for detailed information about CTIX features.

SYSTEM ADMINISTRATION

The *S/Series CTIX Administrator's Guide* defines the responsibilites of an S/Series system administrator and provides procedures for the administrator to follow. The manual explains the concepts an administrator must understand to maintain an S/Series computer system including user support, CTIX modes, CTIX file systems, peripheral devices, and troubleshooting procedures.

The *CTIX Administration Tools Manual* provides an introduction to the principles of CTIX system administration. This manual, and the appropriate administrator's reference manual, represent a complete set of administration instructions.

PROGRAM DEVELOPMENT

The *Programmer's Guide: CTIX Supplement* addresses the programmer's need for detailed explanations about the CTIX operating system. The guide discusses major tools, including basic interaction with the operating system, calculator languages, text editing, and C programming. The guide is used in conjunction with the *CTIX Operating System Manual* and the *UNIX System V Release 3.2 Programmer's Guide*

The *AT&T UNIX System V Release 3.2 Programmer's Guide* describes standard System V, Release 3.2 programming tools and operating system facilities.

The *AT&T UNIX System V Release 3.2 Streams Primer* contains basic information for programming with the System V, Release 3.2 facilities.

The *AT&T UNIX System V Release 3.2 Streams Programmer's Guide* provides detailed programming information about System V, Release 3.2 I/O.

AT&T UNIX System V Release 3.2 Streams Primer provides basic how-to information for programming using the System V, Release 3 Streams I/O facilities.

PROGRAMMING LANGUAGES

The *CTIX BASIC Interpreter and Compiler Manual* describes the BASIC language, the built-in editors with which source files can be created, and the use of both the interpreter and compiler to create and execute BASIC programs.

The *CTIX FORTRAN Manual* is a reference and user's guide for both standard and enhanced versions of CTIX FORTRAN.

The *CTIX Enhanced FORTRAN and Pascal Debugger Manual* describes the interactive execution and debugging of enhanced FORTRAN and Pascal programs, including reference material on debugger concepts and commands, and example sessions with the debugger.

The *CTIX Pascal Manual* is a reference and user's guide for both standard and enhanced versions of CTIX Pascal.

COMMUNICATIONS AND NETWORKS

The *CTIX BSC 2780/3780 RJE Terminal Emulator Manual* describes the daemon, configuration files, operational, maintenance, and line monitoring utilities, as well as the CTIX BSC device drivers, which are components of the Terminal Emulator. This manual is designed to assist end users and system administrators who must configure, operate, and maintain the CTIX BSC 2780/3780 RJE Terminal Emulator.

The *CTIX BSC 3270 Terminal Emulator Manual* provides a product description, a list of IBM Information Display System components emulated by the 3270, CTIX BSC device drivers and servers used, and operational, maintenance, and line monitoring utilities. This manual is designed to assist end users and system administrators who must configure, operate, and maintain the CTIX BSC 3270 Terminal Emulator.

The *CTIX Network Administrator's Guide* describes how to administer networking on an S/Series system: it explains the responsibilities of the network administrator and provides a roadmap to the network setup process.

The *CTIX Network Programmer's Primer* contains instructions for programmers on choosing a networking method. The manual includes details of the CTIX implementations of networking standards.

The *CTIX SNA 3270 Terminal Emulator Manual* outlines the features and functions of the SNA 3270 Terminal Emulator for the CTIX operating system. The manual provides detailed instructions for using, installing, configuring, and troubleshooting the emulator software.

The *CTIX SNA LU6.2 APPC Server Manual* describes the features, functions, advantages, and components of the server. The manual is designed to assist system administrators and transaction programmers who must install, configure, maintain, and troubleshoot the LU6.2 Server.

The *CTIX SNA Network Gateway Manual* provides technical information and operating procedures for the installation, configuration, operation, and maintenance of a CTIX SNA Network Gateway. The manual also defines IBM SNA general concepts and describes the Gateway components.

The *CTIX SNA PU Type 2.1 Network Gateway Manual* provides a complete description of the Gateway along with installation, configuration, operation, maintenance, and troubleshooting procedures designed to aid the system administrator.

The *CTIX SNA RJE Manual* describes the SNA RJE subsystem. Build on the SNA Network Gateway, SNA RJE allows multiple, concurrent logical unit sessions with remote IBM-compatible hosts. The manual describes user interface features, installation, and a procedural interface for user-defined RJE application systems.

The *CTIX X.25 Network Gateway Manual* describes the S/Series CTIX SNA network gateway, including basic SNA concepts and detailed instructions for using, installing, configuring, and troubleshooting the gateway software.

The *CTIX X.25 Terminal/Host Adaptor Manual* describes how to set up and configure the PAD and Host Adaptor for use with the CTIX X.25 network gateway.

The *AT&T UNIX System V Release 3.2 Network Programmer's Guide* contains detailed information on the System Transport Interface.

DATA MANAGEMENT FACILITIES

The *CTIX ISAM Manual* documents the CTIX ISAM software package. It provides a tutorial on writing ISAM applications, describes ISAM procedures, and presents information about utilities designed for ISAM maintenance.

HOW TO USE THIS MANUAL

This second edition of the *CTIX Operating System Manual, Version C*, describes the commands, system calls, libraries, data files, and device interfaces that make up the CTIX Operating System for S/Series Computer Systems. This manual should always be your starting point when you need to find the documentation for a CTIX feature with which you are unfamiliar.

The manual consists of a large number of short entries, sometimes called "the *man* pages," after the command that accesses the entries when they are kept online. Each entry briefly documents some feature of CTIX. Some features require longer documentation than an entry in this manual; such features have an entry that outlines the feature and cross-references the manual that documents the feature fully. Entries that do not refer to other manuals are self-contained and are the final word on the features they describe.

Organization of the manual. The entries are organized into seven sections in four volumes:

Volumes 1 and 2:

1. Commands and Application Programs.

Volume 3:

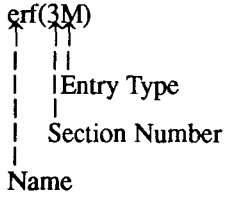
2. System Calls.
3. Subroutines and Libraries.

Volume 4:

4. File Formats.
5. Miscellaneous Facilities.
6. Games.
7. Special Files.

Within each section, entries are alphabetical by title, except for an *intro* entry at the beginning of each section.

Entry Title Conventions. An entry title looks like this example:



Name is the name of the entry. *Section Number* indicates the section that contains the entry. In this case, the entry is in Section 3, which is in Volume 2. *Entry Type* appears only on entries that belong to special categories; refer to the section's *intro* entry for an explanation. In this case, a reference to *intro*(3) would tell you that *erf*(3M) describes functions from the Math Library, which the C compiler does not load by default.

Finding the entry you need. To find out which entry you need, refer to the following guides:

- The Permuted Index. This indexes each significant word in each entry's description. It is useful when you have only a general notion what you're looking for. It is also useful when you know the name of the command or function you are interested in, but there is no entry by that name.
- The Table of Contents. This is a simple list of entries, by section, together with the entry descriptions. Volumes 1 and 2 have Tables of Contents for Section 1. Volume 3 has a Table of Contents for Sections 2 and 3. Volume 4 has a Table of Contents for Sections 4 through 7.
- The Table of Related Entries. For Volume 1 only. A table of entries organized so that related entries are grouped together.

Section organization. Each section begins with an *intro* entry, which provides important general information for that section.

Section 1, Commands and Application Programs, describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory **/bin** (for **bin**ary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the *shell*. Commands that were not transported from UNIX System V reside in **/usr/local/bin**; this directory is recommended for locally implemented programs. Some administrative commands reside in **/etc** and various other places. The **/etc** directory is searched automatically if you are logged in as root; otherwise use the full path name given under **SYNOPSIS** or change the **PATH** environment variable to include the command's directory.

Section 2, System Calls, describes the entries into the CTIX kernel, including the C language interfaces.

Section 3, Subroutines and Libraries, describes the available library functions or subroutines. Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4, File Formats, documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories **/usr/include** and **/usr/include/sys**.

Section 5, Miscellaneous Facilities, contains descriptions of character sets, macro packages, and other such information.

Section 6, Games, describes the games and educational programs that reside in the directory **/usr/games**.

Section 7, Special Files, discusses the characteristics of files that actually refer to input/output devices.

Entry organization. All entries are based on a common format, in which some parts are optional:

NAME The **NAME** part gives the name(s) of the entry and briefly states its purpose.

SYNOPSIS The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (Commands and Application Programs):

- **Bold** **Boldface** strings are literals, and are to be typed just as they appear.
- *Regular* *Regular face* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.
- [ ] Square brackets around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file," it always refers to a *file* name.
- ... Ellipses are used to show that the previous argument prototype can be repeated.
- − + = A final convention is used by the commands themselves. An argument beginning with a minus (−), plus (+), or equal sign (=) is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with −, +, or =.

DESCRIPTION The **DESCRIPTION** part discusses the subject at hand.

EXAMPLE(S) The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

FILES The **FILES** part gives the file names that are built into the program.

SEEALSO The **SEE ALSO** part gives pointers to related information.

DIAGNOSTICS The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

NOTES The **NOTES** part gives information that might be helpful under the particular circumstance described.

WARNINGS The **WARNINGS** part points out potential pitfalls.

BUGS The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents is provided at the front of each of the four volumes, along with a complete permuted index derived from the tables. On each *index* line, the title of the

entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

HOW TO GET STARTED

This discussion provides the basic information you need to get started on CTIX: how to log in and log out, how to communicate through your terminal, and how to run a program. (See the *Programmer's Guide: CTIX Supplement* for a more complete introduction to the system.)

Logging in. Most S/Series terminals are 9600 baud asynchronous terminals. An unused terminal prompts **login:**.

Most asynchronous terminals have a speed switch that should be set to the appropriate speed and a half-/full-duplex switch that should be set to full-duplex. When a connection (at the speed of the terminal) has been established, the system displays a **login:** prompt; you should enter your user name and press Return. If you have a password (and you should!), the system prompts for it, but does not print (echo) it on the terminal.

It is important that you use lowercase characters, if possible, to enter your login name; if you use uppercase, CTIX assumes that your terminal cannot generate lowercase letters and that you mean all subsequent uppercase input to be treated as lowercase.

Once you log in successfully, the shell displays a dollar sign (**$**) prompt. (The shell is described below, under "How to run a program.")

For more information, consult *login*(1), which discusses the login sequence in more detail, and *stty*(1), which tells you how to describe the characteristics of your terminal to the system. The *profile*(4) page describes how to have the shell automatically perform startup tasks when you log in. To log out, type an end-of-file indication to the shell (ASCII **EOT** character, Control-D on most terminals). The shell terminates and the **login:** message appears again.

How to communicate through your terminal. When you type, the system gathers and saves your characters. These characters are not given to a program until you press Return, as described above under "Logging in."

Terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is displaying information on the screen. Of course, if you type during output, the output display is interspersed with your input characters. However, whatever you type is saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away characters.

On an input line from a terminal, the character **@** kills all the characters typed before it. The Backspace key (Control-H if your terminal lacks a Backspace key) erases the last character typed. Successive uses of Backspace erases characters back to, but not beyond, the beginning of the line; to print the **@** and Backspace characters, precede each with a backslash (\). The default erase and kill characters can be changed; see *stty*(1).

The ASCII **DC3** (Control-S) character can be used to temporarily stop output. It is useful with terminals to prevent output from disappearing before it can be read. Output is resumed when a **DC1** (Control-Q) or a second **DC3** (or any other character, for that matter) is typed. The **DC1** and **DC3** characters are not passed to any other program when used in this manner.

The ASCII **DEL** character is not passed to programs, but instead generates an *interrupt signal*, just like the break, interrupt, or attention signal. This signal generally terminates a running program. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed*(1), for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The *quit* signal is generated by typing the ASCII **FS** character It not only causes a running program to terminate, but also generates a file with the core image of the terminated process. *Quit* is useful for debugging.

The system tries to detect whether you have a terminal with the new-line function, or whether it must be simulated with a carriage return and line-feed pair. In the latter case, all input carriage return characters are changed to line-feed characters (the standard line delimiter), and a carriage return and line-feed pair is echoed to the terminal. If you get into the wrong mode, the *stty*(1) command can rescue you.

Tab characters are used freely in programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty*(1) command sets or resets this mode. The system assumes that tabs are set every eight character positions. The *tabs*(1) command sets tab stops on your terminal, if that is possible.

How to run a program. Once you successfully log in, a program called the shell is listening to your terminal. The shell reads the lines you type, splits them into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see "The current directory" below) for a program with the given name; if the program does not exist there, the shell looks in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space and/or tab characters.

When a program terminates, the shell ordinarily regains control, prompting with the $ prompt to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh*(1).

The current directory. The CTIX file system is arranged in a hierarchy of directories. When the system administrator assigns you a user name, that person should also create a directory for you (ordinarily given your name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name typed is by default assumed to be in that directory. Because you are

the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions for other directories are granted or denied to you by their respective owners, or by the system administrator. To change the current directory use *cd*(1).

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with /, which is the name of the *root* directory of the whole file system. After the slash comes the name of each directory containing the next subdirectory (followed by a /), until finally the file name is reached: for example, **/usr/ae/filex** refers to file **filex** in directory **ae**, while **ae** is itself a subdirectory of **usr**; **usr** springs directly from the root directory. See *intro*(2) for a formal definition of *path name*.

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (*without* a prefixed /). Unless otherwise specified, a path name can be used anywhere a file name is required.

Important commands that modify the contents of files are *cp*(1), *mv*, and *rm*(1), which respectively copy, move (that is, rename), and remove files. To find out the status of files or directories, use *ls*(1). Use *mkdir*(1) for making directories and *rmdir*(1) for destroying them.

For more information about file systems, you might want to glance through Section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Writing a program. To enter the text of a source program into a file, use *ed*(1), *ex*(1), or *vi*(1). After the program text has been entered and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor is left in a file in the current directory named **a.out** (if that output is significant, use *mv*(1) to give it a less vulnerable name).

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the shell in response to the **$** prompt.

If any execution (run-time) errors occur, you should use *adb*(1) to examine the remains of your program.

Your programs can receive arguments from the command line just as system programs do; see *exec*(2).

Surprises. Certain commands provide *inter-user* communication. Even if you do not plan to use them, it is a good idea to learn something about them, because someone else may aim them at you. To communicate with another user currently logged in, use *write*(1); *mail*(1) leave a message whose presence is announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

When you log in, a message-of-the-day may greet you before the first **$**.

Changes from UNIX System V. This second edition of the *CTIX Operating System Manual, Version C*, documents Release 6.2 of CTIX for S/Series systems, which is derived from UNIX System V, Release 3.2.

The manual also includes descriptions of the CTIX Internetworking programs and tools.

These are the important changes in UNIX software in CTIX:

The language support provided by the *bs*, *efl*, *ratfor*, *sno*, and *f77* programs. In their place, Convergent Technologies can provide the following CTIX languages: GSA high level COBOL; GSA-certified FORTRAN 77; Pascal; BASIC.

A terminal name is of the form **tty***xxx* instead of **tty***xx*. RS-232 terminal numbers range from tty000 to tty255; RS-422 terminal numbers range from tty256 to tty511.

There are two changes in terminal defaults. The default speed for RS-232 terminals is 9600 baud instead of 300 baud. The default erase character for all terminals is BACKSPACE (control-h if your terminal lacks a BACKSPACE key) instead of #.

Ls columnizes its output by default if the standard output is a terminal, making *ls* easier to use on video terminals. This convention and the associated **-C** option are borrowed from the Berkeley Software Distribution.

Many Berkeley Software Distribution programs, libraries, and networking programs are included. See especially the indispensible *head*(1), *more*(1), *renice*(1), and *ul*(1). In addition to the *curses* [based on *terminfo*(4)], the Berkeley *ocurse* library (based on *termcap*(4)) is supported.

Compatibility of CTIX features. The following lists are provided for cross-machine comparisons of commands and files on Convergent systems. The programs listed in the categories below are not provided with the standard UNIX System V operating system.

Note that most CTIX commands for S/Series systems are compatible with other operating systems based on the UNIX System V operating system. In this manual, commands or certain features that apply only to one system are clearly announced in the text. Note that although most CTIX commands on S/Series systems are CTIX- and UNIX-compatible, they may not be identical on other computer systems; to ensure portability, compare the documentation for each system to be used.

S/Series:

See the following pages for CTIX features that are provided by Convergent exclusively for S/Series systems; they are not available on other CTIX or UNIX systems:

Section 1: *conlocate*(1M), *errdead*(1M), *errdemon*(1M), *errpt*(1M), *erstop*(1M), *extproc*(1M), *iv*(1M), *reboot*(1M), *riopcfg*(1M), *riopqry*(1M), *scsimap*(1M), *swap*(1M), *tio*(1).

Section 2: *uadmin*(2).

Section 3: *libdev*(3X).

Section 4: *errfile*(4), *gateways*(4), *system*(4), *tapedrives*(4).

Section 7: *en*(7), *err*(7), *ipt*(7), *qic*(7), *stape*(7), *tiop*(7), *vme*(7).

Convergent Systems Only:

The following pages apply only to Convergent systems:

Section 1: *bcopy*(1M), *cc1sw*(1), *createdev*(1M) *ctinstall*(1), *getservaddr*(1M), *lpset*(1M), *masterupd*(1M), *mkdbsym*(1M), *rtpenable*(1M), *serstat*(1M), *tapeset*(1M), *tsioctl*(1), *uconf*(1M), *update*(1M).

Section 2: *locking*(2), *notify*(2).

Section 4: *tapedrives*(4).

Section 5: *Devices*(5), *Dialers*(5), *naddr.d*(5).

Section 7: *scsi*(7).

(Note that this is not an exhaustive list of all CTIX features. For further information about the special features of CTIX on other Convergent systems, see the appropriate operating system manuals and Release Notices.)

PERMUTED INDEX

This index includes entries for all pages of Volumes 1 through 4. The entries themselves are based on the one-line descriptions or titles found in the NAME portion of each manual page; the significant words (keywords) of these descriptions are listed alphabetically down the center of the index.

The index is actually a keyword-in-context (KWIC) index that has three columns. To use the index, read the center column to look up specific commands by name or by subject topics. Note that the entry may begin in the left column or wrap around and continue into the left column. A period (.) marks the end of the entry, and a slash (/) indicates where the entry has been continued or truncated. The right column gives the manual page where the command or subject is described.

TABLE OF RELATED ENTRIES

Administration

Accounting and Profiling

acct . . . . . overview of accounting and miscellaneous accounting commands
acctcms . . . . . command summary from per-process accounting records
acctcom . . . . . search and print process accounting file(s)
acctcon . . . . . connect-time accounting
acctmerg . . . . . merge or add total accounting files
acctprc . . . . . process accounting
acctsh . . . . . shell procedures for accounting
fwtmp . . . . . manipulate connect accounting records
prof . . . . . display profile data
runacct . . . . . run daily accounting
sar . . . . . system activity report package

Backups

ckbupscd . . . . . check file system backup schedule
ff . . . . . list file names and statistics for a file system
finc . . . . . fast incremental backup
frec . . . . . recover files from a backup tape
tio . . . . . asynchronous tape i/o filter
volcopy . . . . . copy file systems with label checking

Controlling System State

brc . . . . . system initialization shell scripts
crash . . . . . examine system images
getty . . . . . set terminal type, modes, speed, and line discipline
init . . . . . process control initialization
killall . . . . . kill all active processes
login . . . . . sign on
rc0 . . . . . run commands performed to stop the operating system
rc2 . . . . . run commands performed for multi-user environment
reboot . . . . . reboot the system
shutdown . . . . . terminate all processing
telinit . . . . . direct the actions of init
wall . . . . . write to all users

Disk Management

bcopy . . . . . interactive block copy
clri . . . . . clear i-node
dcopy . . . . . copy file systems for optimal access time
devnm . . . . . device name
df . . . . . report number of free disk blocks
fsck . . . . . file system consistency check and interactive repair
fsdb . . . . . file system debugger
fsstat . . . . . report file system status
fstyp . . . . . determine file system identifier
fuser . . . . . identify processes using a file or file structure
iv . . . . . initialize and maintain volume
labelit . . . . . provide labels for file systems
link . . . . . exercise link and unlink system calls
mkfs . . . . . construct a file system

mklost+found . . . . . make a lost+found directory for fsck
mount . . . . . mount and dismount file system
mountall . . . . . mount, unmount multiple file systems
mvdir . . . . . move a directory
ncheck . . . . . generate names from i-numbers
scsimap . . . . . set mappings for SCSI devices
setmnt . . . . . establish mount table
swap . . . . . swap administrative interface
sync . . . . . update the super block
update . . . . . provide disk synchronization

General

adman . . . . . administer a CTIX system
config . . . . . configure a CTIX system
conlocate . . . . . locate a terminal to use as the virtual system console
cpset . . . . . install object files in binary directories
createdev . . . . . create device nodes
cron . . . . . clock daemon
errdead . . . . . extract error records and status information from dump
errdemon . . . . . error-logging demon
errpt . . . . . process a report of logged errors
errstop . . . . . terminate the error-logging daemon
helpadm . . . . . makes changes to the Help Facility database
install . . . . . install commands
masterupd . . . . . update master file
mkdbsym . . . . . load symbols in kernel debugger
mknod . . . . . build special file
passmgmt . . . . . password files management
path . . . . . locate executable file for command
pwck . . . . . password/group file checkers
pwconv . . . . . convert shadow password file
pwuconv . . . . . convert shadow password file
rsterm . . . . . manually start and stop terminal input and output
serstat . . . . . display serial port error statistics
setuname . . . . . set name of system
uadmin . . . . . administrative control
uconf . . . . . configure the operating system
whodo . . . . . who is doing what

Interprocess Communication

ipcrm . . . . . remove a message queue, semaphore set or shared memory id
ipcs . . . . . report inter-process communication facilities status

Assist

assist . . . . . assistance using CTIX system commands
astgen . . . . . generate/modify ASSIST menus and command forms

Basic File Commands

cat . . . . . concatenate and print files
chmod . . . . . change mode
chown . . . . . change owner or group
dircmp . . . . . directory comparison
cp . . . . . copy, link or move files
dd . . . . . convert and copy a file
file . . . . . determine file type
find . . . . . find files
ls . . . . . list contents of directory

pwd . . . . . working directory name
mkdir . . . . . make a directory
rm . . . . . remove files or directories
umask . . . . . set file-creation mode mask

Basic General Commands

calendar . . . . . reminder service
date . . . . . print and set the date
finger . . . . . user information lookup program
id . . . . . print user and group IDs and names
kill . . . . . terminate a process
logname . . . . . get login name
makekey . . . . . generate encyption key
newgrp . . . . . log in to a new group
news . . . . . print news items
passwd . . . . . change login password
ps . . . . . report process status
sum . . . . . print checksum and block count of a file
uname . . . . . print name of system
who . . . . . who is on the system

Communication Between Systems

ct . . . . . spawn getty to a remote terminal
cu . . . . . call another computer system
uuclean . . . . . uucp spool directory clean-up
uucp . . . . . copy data between computer systems
uustat . . . . . uucp status inquiry and job control
uuto . . . . . public computer system-to-computer system file copy
Uutry . . . . . try to contact a remote system with debugging on
uux . . . . . remote system command execution

Communication Between Users

mail . . . . . send mail to users or read mail
mailx . . . . . interactive message processing system
mesg . . . . . permit or deny messages
sendmail . . . . . send mail over the Internet
talk . . . . . talk to another user
talkd . . . . . remote user communication server
write . . . . . write to another user

Document Formatting and Checking

col . . . . . filter reverse line-feeds
cw . . . . . prepare constant-width text for troff
deroff . . . . . remove nroff/troff, tbl, and eqn constructs
du . . . . . summarize disk usage
eqn . . . . . format mathematical text for nroff or troff
greek . . . . . select terminal filter
hyphen . . . . . find hyphenated words
mm . . . . . print/check documents formatted with the MM macros
mmt . . . . . typeset documents, view graphs, and slides
nroff . . . . . format text
ptx . . . . . permuted index
spell . . . . . find spelling errors
tbl . . . . . format tables for nroff or troff
troff . . . . . typeset text

Internetworking Tools

arp . . . . . address resolution display and control
finger . . . . . user information lookup program
fingerd . . . . . remote user information server
ftp . . . . . file transfer program
ftpd . . . . . DARPA Internet File Transfer Protocol server
hostid . . . . . set or print identifier of current host system
hostname . . . . . set or print the Internet host name of the current system
ifconfig . . . . . configure network interface parameters
inetd . . . . . Internet "super/server"
mkhosts . . . . . make node name commands
named . . . . . Internet domain name server
netstat . . . . . show network status
ping . . . . . send ICMP ECHO_REQUEST packets to network hosts
portmap . . . . . DARPA port to RPC program number mapper
rcmd . . . . . remote shell command execution
rcp . . . . . remote file copy
rexecd . . . . . remote execution server
rlogin . . . . . remote login
rlogind . . . . . remote login server
route . . . . . remove files or directories
routed . . . . . network rounting daemon
rpcinfo . . . . . report RPC information
rshd . . . . . remote shell server
ruptime . . . . . display status of nodes on local network
rwho . . . . . who is logged in on local network
rwhod . . . . . node status server
setaddr . . . . . set DARPA Internet address from nodename
setenet . . . . . write Ethernet address on disk
slattach . . . . . attach serial lines as network interfaces
sldetach . . . . . detach serial lines as network interfaces
slink . . . . . streams linker, load socket configuration
slipd . . . . . switched Serial Line Internet Protocol control facility
telnet . . . . . user interface to TELNET protocol
telnetd . . . . . DARPA TELENET protocol server
tftp . . . . . user interface to the DARPA TFTP protocol
tftpd . . . . . DARPA Trivial File Transfer Protocol server

Network File System (NFS) Utilities

mountd . . . . . mount request server
nfsd . . . . . NFS daemons
nfsstat . . . . . Network File System statistics
nmountall . . . . . mount, unmount Network File System resources
showmount . . . . . show all remote mounts

Networking Support Utilities

nlsadmin . . . . . network listener service administration
strace . . . . . print STREAMS trace messages
strclean . . . . . STREAMS error logger cleanup program
strerr . . . . . STREAMS error logger daemon

Mathematics Tools

bc . . . . . arbitrary-precision arithmetic language
dc . . . . . desk calculator
factor . . . . . factor a number
spline . . . . . interpolate smooth curve

units . . . . . conversion program

Miscellaneous

glossary . . . . . definitions of common CTIX system terms and symbols
locate . . . . . identify a CTIX system command using keywords
makekey . . . . . generate encryption key
nl . . . . . line numbering filter
pack . . . . . compress and expand files
script . . . . . make typescript of terminal session
starter . . . . . information about the operating system for beginning users
su . . . . . become super-user or another user
usage . . . . . retrieve a command description and usage examples
wc . . . . . word count

Offline Storage

cpio . . . . . copy file archives in and out
tapeset . . . . . set drive parameters for tape controllers
tsioctl . . . . . facilitate usage of a tape drive
tar . . . . . tape file archiver

Printer Spooling

accept . . . . . allow/prevent LP requests
enable . . . . . enable/disable LP printers
lp . . . . . send/cancel requests to an LP line printer
lpadmin . . . . . configure the LP spooling system
lpr . . . . . line printer spooler
lpsched . . . . . start/stop the LP request scheduler and move requests
lpset . . . . . set parallel line printer options
lpstat . . . . . print LP status information

Program Development

adb . . . . . absolute debugger
ar . . . . . archive and library maintainer for portable archives
as . . . . . mc68010 assembler
bs . . . . . a compiler/interpreter for modest-sized programs
cb . . . . . C program beautifier
cc . . . . . C compiler
cc1sw . . . . . front-end to the cc command
cflow . . . . . generate C flow graph
conv . . . . . common object file converter
convert . . . . . convert archive files to common formats
cpp . . . . . the C language preprocessor
cprs . . . . . compress a common object file
ctags . . . . . create a tags file
ctrace . . . . . C program debugger
cxref . . . . . generate C program cross reference
dis . . . . . object code disassembler
dump . . . . . dump selected parts of an object file
efl . . . . . extended FORTRAN language
fsplit . . . . . split FORTRAN, ratfor, or efl files
gencc . . . . . create a front-end to the cc command
hd . . . . . hexadecimal and ascii file dump
ld . . . . . link editor for common object files
lint . . . . . a C program checker
list . . . . . produce C source listing from a common object file
lorder . . . . . find ordering relation for an object library

m4 . . . . . macro processor
mcs . . . . . manipulate the object file comment section
make . . . . . maintain, update, and regenerate groups of programs
mkshlib . . . . . create a shared library
nm . . . . . print name list of common object file
od . . . . . octal dump
ratfor . . . . . rational FORTRAN dialect
regcmp . . . . . regular expression compile
size . . . . . print section sizes of common object files
sno . . . . . SNOBOL interpreter
strings . . . . . extract the ASCII text strings in a file
strip . . . . . strip symbol and line number information
time . . . . . time a command
timex . . . . . time a command; report process data and system activity
touch . . . . . update access and modification times of a file
tsort . . . . . topological sort
xstr . . . . . extract and share strings in C programs

Remote File Sharing (RFS) Utilities

adv . . . . . advertise a directory for remote access
dname . . . . . print Remote File Sharing domain and network names
fumount . . . . . forced unmount of an advertised resource
fusage . . . . . disk access profiler
idload . . . . . Remote File Sharing user and group mapping
nsquery . . . . . Remote File Sharing name server query
rfadmin . . . . . Remote File Sharing domain administration
rfpasswd . . . . . change Remote File Sharing host password
rfstart . . . . . start Remote File Sharing
rfstop . . . . . stop the Remote File Sharing environment
rfuadmin . . . . . Remote File Sharing notification shell script
rfudaemon . . . . . Remote File Sharing daemon process
rmntstat . . . . . display mounted resource information
rmnttry . . . . . attempt to mount remote resources
rmount . . . . . retry remote resource mounts
rmountall . . . . . mount, unmount Remote File Sharing resources
rumount . . . . . cancel queued remote resource request
unadv . . . . . unadvertise a Remote File Sharing resource

The Shell

basename . . . . . deliver portions of path names
chroot . . . . . change root directory for a command
cd . . . . . change working directory
echo . . . . . echo arguments
env . . . . . set environment for command execution
expr . . . . . evaluate arguments as an expression
getopt . . . . . parse command options
getopts . . . . . parse command options
line . . . . . read one line
machid . . . . . processor type
nice . . . . . run a command at low priority
nohup . . . . . run a command immune to hangups and quits
rtpenable . . . . . real-time priorities enabled/disabled
sh . . . . . shell, the standard/restricted command programming language
sleep . . . . . suspend execution for an interval
tee . . . . . pipe fitting
test . . . . . condition evaluation command
true . . . . . provide truth values

wait . . . . . await completion of process
xargs . . . . . construct argument list(s) and execute command

Source Code Control System (SCCS)

admin . . . . . create and administer SCCS files
cdc . . . . . change the delta commentary of an SCCS delta
comb . . . . . combine SCCS deltas
delta . . . . . make a delta (change) to an SCCS file
get . . . . . get a version of an SCCS file
help . . . . . ask for help
prs . . . . . print an SCCS file
rmdel . . . . . remove a delta from an SCCS file
sact . . . . . print current SCCS file editing activity
sccsdiff . . . . . compare two versions of an SCCS file
unget . . . . . undo a previous get of an SCCS file
val . . . . . validate SCCS file
vc . . . . . version control
what . . . . . identify SCCS files

Terminal Support

300 . . . . . handle special functions of DASI 300 and 300s terminals
4014 . . . . . paginator for the TEKTRONIX 4014 terminal
450 . . . . . handle special functions of the DASI 450 terminal
asa . . . . . interpret ASA carriage control characters
clear . . . . . clear terminal screen
captoinfo . . . . . convert a termcap description into a terminfo description
hp . . . . . handle special functions of HP 2640 and 2621-series terminals
infocmp . . . . . compare or print out terminfo descriptions
stty . . . . . set the options for a terminal
tabs . . . . . set tabs on a terminal
tdl . . . . . RS-232 terminal download
tic . . . . . terminfo compiler
tset . . . . . set terminal, terminal inteface, and terminal environment
tty . . . . . get the terminal's name
wm . . . . . window management

Text Tools

Browsers, Editors, and Splitters

bfs . . . . . big file scanner
csplit . . . . . context split
ed . . . . . text editor
ex . . . . . text editor
more . . . . . text perusal
newform . . . . . change the format of a text file
pg . . . . . file perusal filter for soft-copy terminals
split . . . . . split a file into pieces
vi . . . . . screen-oriented (visual) display editor based on ex

Comparing Files

bdiff . . . . . big diff
cmp . . . . . compare two files
comm . . . . . select or reject lines common to two sorted files
diff . . . . . differential file comparator
diff3 . . . . . 3-way differential file comparison
diffmk . . . . . mark differences between files

sdiff . . . . . side-by-side difference program

Customizable Filters and Text Programming Languages

awk . . . . . pattern scanning and processing language
cut . . . . . cut out selected fields of each line of a file
crypt . . . . . encode/decode
egrep . . . . . search a file for a pattern using full regular expressions
fgrep . . . . . search a file for a character string
fold . . . . . fold long lines for finite width output device
grep . . . . . search a file for a pattern
join . . . . . relational database operator
lex . . . . . generate programs for simple lexical tasks
paste . . . . . merge same lines of several files or subsequent lines of one file
pr . . . . . print files
sed . . . . . stream editor
sort . . . . . sort and/or merge files
tail . . . . . deliver the last part of a file
tr . . . . . translate characters
uniq . . . . . report repeated lines in a file
yacc . . . . . yet another compiler-compiler

Graphics and Displays

banner . . . . . make posters
cal . . . . . print calendar
graph . . . . . draw a graph

NAME

intro - introduction to commands and application programs

DESCRIPTION

This section describes CTIX commands in alphabetical order. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.

(1C) Commands for communication with other systems.

(1G) Commands used primarily for graphics and computer-aided design.

(1M) Commands for system maintenance and administration.

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option* (*s*)] [*cmdarg* (*s*)]

where:

name	The name of an executable file.
option	- *noargletter* (*s*) or, - *argletter* <> *optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with - or, - by itself indicating the standard input.

Throughout the manual pages there are references to *TMPDIR*, *BINDIR*, *INCDIR*, *LIBDIR*, and *LLIBDIR*. These are not environment variables and cannot be set; instead, they represent directory names whose value is specified on each manual page as necessary. [Note there is also an environment variable called **TMPDIR**, which can be set. When **TMPDIR** is referred to as an environment variable, it is printed in boldface. Commands that acknowledge **TMPDIR** are noted as such on the appropriate manual pages; see also *tmpnam*(3S).]

NOTES

CTIX Internetworking man pages frequently cite appropriate RFCs (Requests for Comments). RFCs can be obtained from the DDN Network Information Center, SRI International, Menlo Park, CA 94025.

SEE ALSO

getopt(1), *getopts*(1), *exit*(2), *wait*(2), *getopt*(3C).
Section 6 for computer games.
How to Get Started, at the front of this volume.

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program [see *wait*(2) and *exit*(2)]. The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, or bad or inaccessible data. It is called variously "exit code," "exit status," or "return code," and is described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

NAME

300, 300s - handle special functions of DASI 300 and 300s terminals

SYNOPSIS

300 [**+12**] [**-n**] [**-dt,l,c**]

300s [**+12**] [**-n**] [**-dt,l,c**]

DESCRIPTION

The *300* command supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; *300s* performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%. The *300* command can be used to print equations neatly, in the sequence:

neqn file ... | nroff | 300

The behavior of *300* can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

+12 Permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.

-*n* Controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of *n* overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, *nroff* half-lines could be made to act as quarter-lines by using **-2**. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **-3** alone, having set the PITCH switch to 12-pitch.

-d*t*,*l*,*c* Controls delay factors. The default setting is **-d3,90,30**. DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values.

Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like **/etc/passwd**.

Because terminal behavior varies according to the specific characters printed and the load on a system, the user might have to experiment with these values to get correct output. The **-d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file **/etc/passwd** can be printed by using **-d3,30,5**. The value **-d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

The *300* command can be used with the *nroff* **-s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of pressing the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

nroff -T300 files ... is equivalent to **nroff files ... | 300**
nroff -T300-12 files ... is equivalent to **nroff files ... | 300 +12**

The use of *300* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *300* may produce better-aligned output.

The *neqn* names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(5).

SEE ALSO

450(1), eqn(1), graph(1G), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

BUGS

Some special characters cannot be printed correctly in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

300(1)

WARNINGS

If your terminal has a PLOT switch, make sure it is turned *on* before you use the *300* command.

NAME

4014 - paginator for the Tektronix 4014 terminal

SYNOPSIS

4014 [**-t**] [**-n**] [**-cN**] [**-pL**] [file]

DESCRIPTION

The output of *4014* is intended for a Tektronix 4014 terminal; *4014* 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, *4014* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command !*cmd* sends the *cmd* to the shell.

The command line options follow:

-t Do not wait between pages (useful for directing output into a file).

-n Start printing at the current cursor position and never erase the screen.

-c*N* Divide the screen into *N* columns and wait after the last column.

-p*L* Set page length to *L*; *L* accepts the scale factors **i** (inches) and **l** (lines); default is lines.

SEE ALSO

pr(1), tc(1), troff(1).

NAME

450 - handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

The *450* command supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the Diablo 1620 or Xerox 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as *300*(1). Use *450* to print equations neatly, in the sequence:

neqn file ... | nroff | 450

Use *450* with the *nroff* **-s** flag or **.rd** requests when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of pressing the Return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the use of *450* can be eliminated in favor of one of the following:

nroff -T450 files ...

or

nroff -T450-12 files ...

The use of *450* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *450* may produce better-aligned output.

The *neqn* names of, and resulting output for, the Greek and special characters supported by *450* are shown in *greek*(5).

SEE ALSO

300(1), eqn(1), graph(1G), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

BUGS

Some special characters cannot be printed correctly in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

WARNINGS

If your terminal has a PLOT switch, make sure it is turned *on* before you use the *450* command. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines per inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

NAME

Uutry - try to contact a remote system with debugging on

SYNOPSIS

/usr/lib/uucp/Uutry [**-x** debug_level] [**-r**] system_name

DESCRIPTION

The *Uutry* shell script is used to invoke *uucico*(1M) to call a remote site. Debugging is enabled; the **-x** option overrides the default debug level (5). The **-r** option causes *Uutry* to override any minimum retry time for the designated system. The debugging output is put in the file **/tmp/***system_name*. A **tail -f** of the output is executed. A terminal ''interrupt'' returns control to the terminal while *uucico*(1M) continues to run, putting its output in the file **/tmp/***system_name*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK.*
/usr/spool/uucppublic/*
/tmp/system_name

SEE ALSO

uucico(1M), uucp(1C), uux(1C).
S/Series CTIX Administrator's Guide.

NAME

accept, reject - allow or prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [**-r**[reason]] destinations

DESCRIPTION

The *accept* command allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

The *reject* command prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject*.

-r[*reason*] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next **-r** option. The *reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the **-r** option is not present or the **-r** option is given without a *reason*, a default *reason* is used.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).
S/Series CTIX Administrator's Guide.

NAME

acctdisk, acctdusg, accton, acctwtmp - overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**-u** file] [**-p** file]

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. The *acctsh*(1M) command describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into **/etc/utmp**, as described in *utmp*(4). The programs described in *acctcon*(1M) convert this file into session and charging records, which are then summarized by *acctmerg*(1M).

Process accounting is performed by the CTIX system kernel. Upon termination of a process, one record per process is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data may be examined using *acctcom*(1).

Process accounting and connect time accounting [or any accounting records in the format described in *acct*(4)] can be merged and summarized into total accounting records by *acctmerg* [see **tacct** format in *acct*(4)]. *prtacct* [see *acctsh*(1M)] is used to format any or all accounting records.

The *acctdisk* command reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

The *acctdusg* command reads its standard input (usually from **find / -print**) and computes disk resource consumption (including indirect blocks) by login. If **-u** is given, records consisting of those filenames for which *acctdusg* charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If **-p** is given, *file* is the name of the password file. This option is not needed if the password file is **/etc/passwd**. [See *diskusg*(1M) for more details.]

Alone, *accton* turns process accounting off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records [see *acct*(2) and *acct*(4)].

The *acctwtmp* command writes a *utmp*(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned [see *utmp*(4)]. The *reason* must be a string of 11 or fewer characters, numbers, $, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp uname >> /etc/wtmp
acctwtmp "file saved" >> /etc/wtmp
```

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
S/Series CTIX Administrator's Guide.

NAME

acctcms - command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [options] files

DESCRIPTION

The *acctcms* command reads one or more *files*, normally in the form described in *acct*(4). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The *options* follow:

-a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, and "hog factor", characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore-minutes.

-c Sort by total CPU time, rather than total kcore-minutes.

-j Combine all commands invoked only once under "***other".

-n Sort by number of command invocations.

-s Any filenames encountered hereafter are already in internal summary format.

-t Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old style *acctcms* internal summary format records.

The following options can be used only with the **-a** option:

-p Output a prime-time-only command summary.

-o Output a non-prime (offshift) time only command summary.

When **-p** and **-o** are used together, a combination prime and non-prime time report is produced. All the output summaries report total usage, except number of times executed, CPU minutes and real minutes, which are split into prime and non-prime.

A typical sequence for performing daily command accounting and for maintaining a running total follows:

```
acctcms file ... >today
cp total previoustotal
acctcms -s today previoustotal > total
acctcms -a -s today
```

SEE ALSO

acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
S/Series CTIX Administrator's Guide.

BUGS

Unpredictable output results if **-t** is used on new style internal summary format files, or if it is not used with old style internal summary format files.

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

acctcom [[options] [file]] ...

DESCRIPTION

The *acctcom* command reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(4) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: **1** for *fork* without *exec*), **STAT** (the system exit status), **HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD**, and **BLOCKS READ** (total blocks read and written).

A **#** is prepended to the command name if the command was executed with superuser privileges. If a process is not associated with a known terminal, a **?** is printed in the **TTYNAME** field.

If no *files* are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using **&** in the shell), **/usr/adm/pacct** is read; otherwise, the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, that is, in chronological order by process completion time. The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/usr/adm/pacct***?*. The *options* follow:

-a Show some average statistics about the processes selected. The statistics will be printed after the output records.

-b Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.

-f Print the *fork/exec* flag and system exit status columns in the output. The numeric output for this option will be in octal.

-h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This *hog factor* is computed as (total CPU time)/(elapsed time).

-i Print columns containing the I/O counts in the output.

-k Instead of memory size, show total kcore-minutes.

-m	Show mean core size (the default).
-r	Show CPU factor [user time/(system-time + user-time)].
-t	Show separate system and user CPU times.
-v	Exclude column headings from the output.
-l *line*	Show only processes belonging to terminal */dev/***line**.
-u *user*	Show only processes belonging to *user* that may be specified by a user ID; a login name that is then converted to a user ID; a **#**, which designates only those processes executed with superuser privileges; or **?**, which designates only those processes associated with unknown user IDs.
-g *group*	Show only processes belonging to *group*. The *group* can be designated by either the group ID or group name.
-s *time*	Select processes existing at or after *time*, given in the format *hr* [:*min* [:*sec*]].
-e *time*	Select processes existing at or before *time*.
-S *time*	Select processes starting at or after *time*.
-E *time*	Select processes ending at or before *time*. Using the same *time* for both **-S** and **-E** shows the processes that existed at *time*.
-n *pattern*	Show only commands matching *pattern* that can be a regular expression as in *ed*(1), except that + means one or more occurrences.
-q	Do not print any output records, just print the average statistics as with the **-a** option.
-o *ofile*	Copy selected process records in the input data format to *ofile*; supress standard output printing.
-H *factor*	Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **-h** above.
-O *sec*	Show only processes with CPU system time exceeding *sec* seconds.
-C *sec*	Show only processes with total CPU time, system plus user, exceeding *sec* seconds.
-I *chars*	Show only processes transferring more characters than the cutoff number given by *chars*.

Listing options together has the effect of a logical *and*.

FILES

/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO

ps(1),
acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), su(1), acct(2), acct(4), utmp(4).
S/Series CTIX Administrator's Guide.

BUGS

The *acctcom* command reports only on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time, *time* is interpreted as occurring on the previous day.

NAME

acctcon1, acctcon2 - connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

The *acctcon1* command converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/etc/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The *options* follow:

-p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

-t The *acctcon1* command maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **-t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

-l *file* *File* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hangup, termination of *login*(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).

-o *file* *File* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

The *acctcon2* command expects as input a sequence of login session records and converts them into total accounting records [see **tacct** format in *acct*(4)].

EXAMPLES

These commands are typically used as shown below. The file **ctmp** is created only for the use of *acctprc* (1M) commands:

```
acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 > ctmp
acctcon2 <ctmp | acctmerg > ctacct
```

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), login(1), runacct(1M), acct(2), acct(4), utmp(4).
S/Series CTIX Administrator's Guide.

BUGS

The line usage report is confused by date changes. Use *wtmpfix* [see *fwtmp*(1M)] to correct this situation.

NAME

acctmerg - merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [options] [file] ...

DESCRIPTION

The *acctmerg* command reads its standard input and up to nine additional files, all in the **tacct** format [see *acct*(4)] or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys. *Options* follow:

-a Produce output in ASCII version of **tacct**.

-i Input files are in ASCII version of **tacct**.

-p Print input with no processing.

-t Produce a single record that totals all input.

-u Summarize by user ID, rather than by user ID and name.

-v Produce output in verbose ASCII format, with more precise notation for floating point numbers.

EXAMPLES

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg -v <file1 >file2

        Edit file2 as desired ...

acctmerg -i <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
S/Series CTIX Administrator's Guide.

NAME

acctprc1, acctprc2 - process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [**ctmp**]

/usr/lib/acct/acctprc2

DESCRIPTION

The *acctprc1* command reads input in the form described by *acct*(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (ticks), non-prime CPU time (ticks), and mean memory size (in memory segment units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

The *acctprc2* command reads records in the form written by *acctprc1*, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:

acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct

FILES

/etc/passwd

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

BUGS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from *cron*(1M), for example. More precise conversion can be done by faking login sessions on the console by using the *acctwtmp* program in *acct*(1M).

CAVEAT

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor. For example, S/Series systems measure this in 4-kilobyte units.

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct - shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee login-name number

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk [-o] [files ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [**-l**] [**-c**] [mmdd]

/usr/lib/acct/prtacct file ["heading"]

/usr/lib/acct/runacct [mmdd] [mmdd state]

/usr/lib/acct/shutacct ["reason"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | off | switch

DESCRIPTION

The *chargefee* command can be invoked to charge a *number* of units to *login-name*. A record is written to **/usr/adm/fee**, to be merged with other accounting records during the night.

The *ckpacct* command, which should be initiated through the use of *cron*(1M), periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* is invoked with argument *switch*. If the number of free 512-byte disk blocks in the **/usr** file system falls below 500, *ckpacct* automatically disables collection of process accounting records through the use of the **off** argument to *turnacct*. When at least this number of blocks is restored, the accounting is activated again. This feature is sensitive to the frequency at which *ckpacct* is executed, usually by *cron*.

The *dodisk* command should be invoked by *cron* to perform the disk accounting functions. By default, it performs disk accounting on the special files in **/etc/fstab**. If the **-o** flag is used, it performs a slower version of disk accounting by login directory. The *files* parameter specifies the one or more filesystem names where disk accounting is performed; If used, disk accounting is performed on these filesystems only. If the **-o** flag is used, *files* should specifiy

mount points of mounted filesystems; If omitted, *files* should specify the special file names of mountable filesystems.

The *lastlogin* command is invoked by *runacct* to update **/usr/adm/acct/sum/loginlog**, which shows the last date on which each person logged in.

The *monacct* command should be invoked once each month or each accounting period. The *number* parameter indicates the month or period. If *number* is not given, the default is the current month (01-12). This default is useful if *monacct* is executed through *cron*(1M) on the first day of each month. The *monacct* command creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

The *nulladm* command creates *file* with mode 664 and ensures that owner and group are **adm**. It is called by various accounting shell procedures.

The *prctmp* command can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon1* [see *acctcon*(1M)].

The *prdaily* command is invoked by *runacct* to format a report of the previous day's accounting data. The report resides in **/usr/adm/acct/sum/rprt***mmdd* where *mmdd* is the month and day of the report. The current daily accounting reports can be printed by using *prdaily*. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The **-l** flag prints a report of exceptional usage by login id for the specified date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of *monacct*. The **-c** flag prints a report of exceptional resource usage by command, and may be used on current day's accounting data only.

The *prtacct* command can be used to format and print any total accounting (**tacct**) file.

The *runacct* command performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see *runacct*(1M).

The *shutacct* command should be invoked during a system shutdown (usually in **/etc/shutdown**) to turn process accounting off and append a "reason" record to **/etc/wtmp**.

The *startup* command is invoked to turn accounting on at system initialization through the presence of a zero-length file named **/etc/rcopts/ACCT**.

The *turnacct* command is an interface to *accton* [see *acct*(1M)], used to turn process accounting **on** or **off**. The **switch** argument disables accounting, moves

the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr* (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then enables accounting again. This procedure is called by *ckpacct* and can be automated by *cron* and used to keep **pacct** to a reasonable size. The *acct* command starts and stops process accounting through the use of *init* and *shutdown*.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct*	used if pacct gets large and during execution of daily accounting procedure
/etc/wtmp	login/logoff summary
/usr/lib/acct/ptelus.awk	contains the limits for exceptional usage by login ID
/usr/lib/acct/ptecms.awk	contains the limits for exceptional usage by command name
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/acct/sum	summary directory, should be saved

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
S/Series CTIX Administrator's Guide.

NAME

adb - absolute debugger

SYNOPSIS

adb [**-w**] [objfil [corfil]]

DESCRIPTION

The *adb* program is a general purpose debugging program. It can be used to examine files and to provide a controlled environment for the execution of CTIX programs.

The *objfil* parameter is normally an executable program file, preferably containing a symbol table; if not, the symbolic features of *adb* cannot be used, but the file can still be examined. The default for *objfil* is **a.out**. The *corfil* parameter 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, both *objfil* and *corfil* are created, if necessary, and opened for reading and writing so that files can be modified using *adb*. Note that *adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general, requests to *adb* are of the following form:

[*address*] [, *count*] [*command*] [;]

If *address* is present, *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command is 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 Hexadecimal by default or if preceded by 0x; octal if preceded by 0o or 0O; decimal if preceded by 0t or 0T.

integer.fraction
A 32-bit floating point number.

'cccc' The ASCII value of up to 4 characters. A \ may be used to escape a '.

< name The value of *name*, which is either a variable name or a 68010/68020 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 registers are **d0** through **d7**, **a0** through **a7**, **sp**, **pc**, **cc**, **sr**, and **usp**.

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*.

From C, only external variables are available as symbols. The symbol name is the same as the C variable name, except that an underscore (_) is prepended to any name that is the same as the name for a register.

(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*. The value of *dot* is incremented by the sum of the increments for each format letter (q.v.).

/f Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for **?**.

=f The value of *address* 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 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. The value *n* is the length of the string including its zero terminator.

Y 4 Print 4 bytes in date format [see *ctime*(3C)].

i n Print as machine instructions. The value *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 eight-space tab stop.

r 0 Print a space.

n 0 Print a new-line.

"..." 0 Print the enclosed string.

^ The value of *dot* is decremented by the current increment. Nothing is printed.

\+ The value of *dot* is incremented by 1. Nothing is printed.

\- The value of *dot* is decremented by 1. Nothing is printed.

new-line

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 * the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If the list is terminated by **?** or /, the file (*objfil* or *corfil*, respectively) is used for subsequent requests. (So that, for example, **/m?** causes / to refer to *objfil*.)

>*name* The value of *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* follow:

- <*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**. The value of *dot* is set to **pc**.
- **b** Print all breakpoints and their associated counts and commands.
- **c** C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **fp**). 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.
- **f** Print the 68881 floating-point registers.

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 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. The value *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess can 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* [see *signal*(2)]. If *address* is given, 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

The *adb* command provides a number of variables. Named variables are set initially by *adb*, but they 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, 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 (0407, 0410, or 0413).

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≤*address*<*e1* => *file address*=*address*+*f1*-*b1*

otherwise:

b2≤*address*<*e2* => *file address*=*address*+*f2*-*b2*,

otherwise, the requested *address* is not legal. In some cases (for example, for programs with separated I and D space) the two segments for a file can overlap. If a ? or / is followed by an asterisk (*), 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 you can use *adb* on large files, all appropriate values are kept as signed 32-bit integers.

FILES

/dev/kmem
/dev/swap
a.out
core

SEE ALSO

ptrace(2), a.out(4), core(4).

DIAGNOSTICS

"Adb" when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, and so on. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

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.

Shared libraries are not included in the map.

NAME

adman - administer a CTIX system

SYNOPSIS

adman

DESCRIPTION

The *adman* program is used to administer a CTIX system. It is a form- and menu-based program that can run on a wide variety of terminals. It performs administrative functions such as backup and restore, adding users to the system, printer management, disk management, network configuration, Remote File Sharing configuration, Network File System configuration, and so forth. Help text is available for all *adman* functions.

Because some administrative functions, such as adding or removing users, are available only to superuser, certain menu items are not displayed when *adman* is invoked by a non-privileged user.

The following is a list of administrative features of *adman*:

- User Administration
  - — List User Accounts
  - — Change Your Password
  - — Add a User
  - — Add a Group
  - — Change User Information
  - — Change a User's Password
  - — Delete a User
  - — Bar a User
  - — Re-admit a User
- System Status
  - — View Hardware Inventory
  - — Error Log Functions
    - — View Summary Report
    - — View Detailed Report
    - — Reset Error Log
  - — Console Log Functions

- — View Network Log
- — Reset System Console
- — Networking Log Functions
- — View Console Log
- — Reset
- — Enable/Disable System Activity Reporting
- — Enable/Disable Process Accounting

- Backup and Restore
  - — Backup Functions
    - — Complete Backup
    - — Incremental Backup
    - — Partial Backup
    - — User Backup
  - — Restore Functions
    - — View Tape Contents
    - — Complete Restore
    - — Partial Restore
- Disk and File System Administration
  - — View Disk Description
  - — Configure a Disk
  - — View/Change Partition Uses
  - — Monitor Disk Usage
  - — List Files of a Certain Age
  - — List Largest Files
  - — List Largest Users of Disk Space
- Terminal Administration
  - — Enable a Login on a Serial Port
  - — Disable a Login on a Serial Port
  - — Configure Cluster Lines and Devices

- View Port Status
- Printer Administration
  - View Print System Status
  - View Print Queue Status
  - Add a Printer
  - Change Printer Status
  - Remove a Printer
  - Set the Default Printer
  - Start/Stop the Print System
  - Make a Printer Model
- UUCP Administration
  - Show Systems Entries
  - Rename This System
  - Add A Remote System Entry
  - Change Entry Information
  - Delete a Remote System Entry
  - Add a Modem to a Port
  - Delete a Modem from a Port
- Network Administration
  - Machine Status
  - Network Users
  - Network Setup
    - Add an Equivalent User
    - Delete an Equivalent User
    - Add an Equivalent Machine
    - Delete an Equivalent Machine
    - Add a Network Service
    - Delete a Network Service
    - Add a New Host Entry

- — Change a Host Entry
- — Delete a Host Entry

— Network Interface Statistics

- — Active Connections
- — Network Interface
- — Routing Tables
- — Memory Usage
- — Protocol Statistics

• Remote File Sharing Administration

— Share Resources

- — List Local Machine's Advertised Resources
- — List Mounted Remote Resources
- — List Mountable Remote Resources
- — Advertise a Local Resource
- — Remove a Local Resource from Advertised List
- — Mount a Remote Resource
- — Unmount a Remote Resource

— Configure the Domain

- — List All Systems in Current Domain
- — Add an Entry for a Primary from a Different Domain
- — Administer Secondary Name Servers in Current Domain
  - — List Secondary Name Servers
  - — Add a New Secondary Entry
  - — Delete an Existing Secondary Entry
- — Add a System to the Domain
- — Delete a System from the Domain
- — Delete All Systems in the Domain

— Set Up Your Machine to Run RFS

— Start or Stop RFS

- — Map or Exclude Remote Users or Groups
- — Change Your RFS Password
- — Display Current Primary
- — Transfer Name Server Responsibilities

- Network File System Administration
  - — Share Resources
    - — List Local Machine's Exported Resources
    - — List Mounted Remote Resources
    - — Show Machines That Have Mounted Any Local Resource
    - — List Other Machines' Exported Resources
    - — Export a Local Resource
    - — Remove a Local Resource From Exported List
    - — Mount a Remote Resource
    - — Unmount a Remote Resource
  - — Set Up Your Machine for NFS
  - — Start NFS
  - — Stop NFS
  - — Network File System Status
  - SCSI Tape Administration
    - — Add a SCSI Tape Drive
    - — Delete a SCSI Tape Drive
    - — Change Entry Information
    - — View SCSI Tape Status

FILES

/usr/lib/terminfo/?/*
/usr/lib/adman/*
/usr/lib/ctam/fonts/*
/usr/lib/ctam/kbmaps/*

SEE ALSO

CTIX Administration Tools Manual.

NOTES

The "Network Administration" menu items are available only if TCP/IP is installed; the "Remote File Sharing Administration" menu items are available only if RFS is installed; the "Network File System Administration" menu items are available only if NFS is installed; the "SCSI Tape Administration" menu items are available only if you have a system with SCSI.

NAME

admin - create and administer SCCS files

SYNOPSIS

admin [**-n**] [**-i**[name]] [**-r**rel] [**-t**[name]] [**-f**flag[flag-val]] [**-d**flag[flag-val]] [**-a**login] [**-e**login] [**-m**[mrlist]] [**-y**[comment]] [**-h**] [**-z**] files

DESCRIPTION

The *admin* command is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which can appear in any order, consist of keyletter arguments, which begin with -, and named files (note that SCCS file names must begin with the characters s.). If a named file does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

-n This keyletter indicates that a new SCCS file is to be created.

-i[name] The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **-r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file can be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **-i** keyletter). Note that the **-i** keyletter implies the **-n** keyletter.

-r*rel* The *rel*ease into which the initial delta is inserted. This keyletter can be used only if the **-i** keyletter is also used. If

the **-r** keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

-t[name] The *name* of a file from which descriptive text for the SCCS file is to be taken. If the **-t** keyletter is used and *admin* is creating a new SCCS file (the **-n** and/or **-i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a **-t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **-t** keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

-f*flag* This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters can be supplied on a single *admin* command line. The allowable *flag*s and their values are:

b Allows use of the **-b** keyletter on a *get*(1) command to create branch deltas.

c*ceil* The highest release (ceiling), a number greater than 0 but less than or equal to 9999, which can be retrieved by a *get*(1) command for editing. The default value for an unspecified **c** flag is 9999.

f*floor* The lowest release (floor), a number greater than 0 but less than 9999, which can be retrieved by a *get*(1) command for editing. The default value for an unspecified **f** flag is 1.

d*SID* The default delta number (SID) to be used by a *get*(1) command.

i*[str]* Causes the *No id keywords (ge6)* message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords [see *get*(1)] are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must

exactly match the given string, however the string must contain a keyword, and no embedded newlines.

j Allows concurrent *get*(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* A *list* of releases to which deltas can no longer be made (**get -e** against one of these "locked" releases fails). The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range>::= | **a**

The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.

n Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (for example, in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas can later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

q*text* User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(1).

m*mod* *Mod*ule name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

t*type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*(1).

v*pgm* Causes *delta*(1) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of

an MR number validity checking program [see *delta*(1)]. (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null.)

-d*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The **-d** keyletter can be specified only when processing existing SCCS files. Several **-d** keyletters can be supplied on a single *admin* command. See the **-f** keyletter for allowable *flag* names.

l*list* A *list* of releases to be "unlocked." See the **-f** keyletter for a description of the **l** flag and the syntax of a *list*.

-a*login* A *login* name, or numerical CTIX system group ID, to be added to the list of users which can make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters can be used on a single *admin* command line. As many *login*s, or numerical group IDs, as desired can be on the list simultaneously. If the list of users is empty, then anyone can add deltas. If *login* or group ID is preceded by a ! they are to be denied permission to make deltas.

-e*login* A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **e** keyletters can be used on a single *admin* command line.

-m*[mrlist]* The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The **v** flag must be set and the MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). Diagnostics will occur if the **v** flag is not set or MR validation fails.

-y*[comment]* The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the **-y** keyletter results in a default comment line being inserted in the form:

date and time created *YY/MM/DD HH:MM:SS* by *login*

The **-y** keyletter is valid only if the **-i** and/or **-n** keyletters are specified (that is, a new SCCS file is being created).

-h Causes *admin* to check the structure of the SCCS file [see *sccsfile*(5)], and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a truly corrupted file can prevent future detection of the corruption.

The last component of all SCCS file names must be of the form **s.***file-name*. New SCCS files are given mode 444 [see *chmod*(1)]. Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file called **x.***file-name*, [see *get*(1)], created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode can be changed to 644 by the owner allowing use of *ed*(1). *Care must be taken!* The edited file should *always* be processed by an *admin* **-h** to check for corruption followed by an *admin* **-z** to generate a proper check-sum. Another *admin* **-h** is recommended to ensure the SCCS file is valid.

admin also makes use of a transient lock file (called **z.***file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get*(1) for further information.

FILES

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; can exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4).
UNIX System V Release 3.2 Programmer's Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

adv - advertise a directory for remote access

SYNOPSIS

adv [**-r**] [**-d** description] resource pathname [clients ...]

adv -m resource **-d** description | [clients ...]

adv -m resource [**-d** description] | clients ...

adv

DESCRIPTION

The *adv* command is the Remote File Sharing command used to make a resource from one computer available for use on other computers. The machine that advertises the resource is called the *server*, while computers that mount and use the resource are *clients*. [See *mount*(1M).] (A resource represents a directory, which could contain files, subdirectories, named pipes and devices.)

There are three ways *adv* is used:

1) To advertise the directory *pathname* under the name *resource* so it is available to Remote File Sharing *clients*.

2) Modify *client* and *description*. To modify fields for currently advertised resources.

3) To print a list of all locally-advertised resources.

The following options are available:

-r Restricts access to the resource to a read-only basis. The default is read-write access.

-d *description* Provides brief textual information about the advertised resource. *description* is a single argument surrounded by double quotes (") and has a maximum length of 32 characters.

resource This is the symbolic name used by the server and all authorized clients to identify the resource. It is limited to a maximum of 14 characters and must be different from every other resource name in the RFS domain. All characters must be printable ASCII characters but must not include periods (.), slashes (/), or white space.

pathname This is the local pathname of the advertised resource. It is limited to a maximum of 64 characters. This pathname cannot be the mount point of a remote resource and it can only be advertised under one resource name.

clients These are the names of all clients that are authorized to remotely mount the resource. The default is that all machines that can connect to the server are authorized to access the resource. Valid input is of the form *nodename*, *RFSdomain.nodename*, *RFSdomain.*, or an alias that represents a list of client names. A RFS domain name must be followed by a period (.) to distinguish it from a node name. The aliases are defined in **/etc/host.alias** and must conform to the alias capability in *mailx*(1).

-m *resource* This option modifies information for a resource that has already been advertised. The resource is identified by a *resource* name. Only the *clients* and *description* fields can be modified. (To change the *pathname*, *resource* name, or read/write permissions, you must unadvertise and re-advertise the resource.)

When used with no options, *adv* displays all local resources that have been advertised; this includes the resource name, the pathname, the description, the read-write status, and the list of authorized clients. The resource field has a fixed length of 14 characters; all others are of variable length. Fields are separated by two white spaces, double quotes (") surround the description, and blank lines separate each resource entry.

This command may be used without options by any user; otherwise it is restricted to the super-user.

Remote File Sharing must be running before *adv* can be used to advertise or modify a resource entry.

EXIT STATUS

If there is at least one syntactically valid entry in the *clients* field, a warning will be issued for each invalid entry and the command will return a successful exit status. A non-zero exit status will be returned if the command fails.

ERRORS

If (1) the network is not up and running, (2) *pathname* is not a directory, (3) *pathname* isn't on a file system mounted locally, or (4) there is at least one entry in the *clients* field but none are syntactically valid, an error message will be sent to standard error.

FILES

/etc/host.alias

SEE ALSO

mailx(1), mount(1M), rfstart(1M), unadv(1M).
S/Series CTIX Administrator's Guide.

NAME

ar - archive and library maintainer for portable archives

SYNOPSIS

ar key [posname] afile [name] ...

DESCRIPTION

The *ar* command maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose. The magic string and the file headers used by *ar* consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar*(4). The archive symbol table [described in *ar*(4)] is used by the link editor [*ld*(1)] to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *ar* when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *ar*(1) command is used to create or update the contents of such an archive, the symbol table is rebuilt. The s option described below will force the symbol table to be rebuilt.

Unlike command options, the command key is a required part of *ar*'s command line. The key (which may begin with a -) is formed with one of the following letters: **drqtpmx**. Arguments to the *key*, alternatively, are made with one of more of the following set: **vuaibcls**. *Posname* is an archive member name used as a reference point in positioning other files in the archive. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are as follows:

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 dates of modification 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. This option is

useful to avoid quadratic behavior when creating a large archive piece-by-piece. Unchecked, the file may grow exponentially up to the second degree.

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.

The meanings of the key arguments are as follows:

v Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, precede each file with a name.

c Suppress the message that is produced by default when *afile* is created.

l Place temporary files in the local (current working) directory rather than in the default temporary directory, *TMPDIR*.

s Force the regeneration of the archive symbol table even if *ar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

FILES

*$TMPDIR/** temporary files

$TMPDIR is usually /tmp but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*() in *tmpnam*(3S)].

SEE ALSO

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4).

NOTES

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

NAME

arp - address resolution display and control

SYNOPSIS

arp hostname
arp -a [namelist] [corefile]
arp -d hostname
arp -s hostname ether_addr [**temp**] [**pub**] [**trail**]
arp -f filename

DESCRIPTION

The *arp* program displays and modifies the Internet-to-Ethernet address translation table, which is normally maintained by the address resolution protocol [*arp*(7)].

When *hostname* is the only argument, *arp* displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation [see *hosts*(4) and *inet*(7)].

Options are interpreted as follows:

-a [*namelist*] [*corefile*]
Display all of the current ARP entries by reading the table from the file *corefile* (default **/dev/kmem**) based on the kernel file *namelist* (default **/etc/lddrv/unix.exec**).

-d Delete an entry for the host whose name is *hostname*. (This can be performed only by the super-user.)

-s *hostname ether_addr* [**temp**] [**pub**] [**trail**]
Create an ARP entry for the host whose name is *hostname* with the Ethernet address *ether_addr*. The Ethernet address is given as six colon-separated, two-digit hexadecimal numbers. The entry will be permanent unless the argument **temp** is specified on the command line. If **pub** is specified, the entry will be "published": that is, this system will act as an ARP server, responding to requests for *hostname* even though the host address is not an address of the local host. If **trail** is specified, trailer encapsulations are to be used with this host.

-f *filename*
Read the file *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form

hostname ether_addr [**temp**] [**pub**] [**trail**]

with argument meanings as given above.

SEE ALSO

inet(3), arp(7), ifconfig(1M).
CTIX Network Administrator's Guide.

NAME

as - common assembler

SYNOPSIS

as [options] filename

DESCRIPTION

The *as* command assembles the named file. The following flags may be specified in any order:

-o *objfile* Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.

-n Turn off long/short address optimization. By default, address optimization takes place.

-m Run the *m4* macro processor on the input to the assembler.

-R Remove (unlink) the input file after assembly is completed.

-dl Do not produce line number information in the object file.

-T Truncate symbols to eight characters.

-V Write the version number of the assembler being run on the standard error output.

-Y *[md],dir* Find the **m4** preprocessor (**m**) and/or the file of predefined macros (**d**) in directory *dir* instead of in the customary place.

FILES

*TMPDIR/** temporary files

TMPDIR is usually **/tmp** but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*( ) in *tmpnam*(3S)].

SEE ALSO

cc(1), ld(1), m4(1), nm(1), strip(1), tmpnam(3S), a.out(4).
Programmer's Guide: CTIX Supplement.

WARNING

If the **-m** (*m4* macro processor invocation) option is used, keywords for *m4* [see *m4*(1)] cannot be used as symbols (variables, functions, labels) in the input file since *m4* cannot determine which are assembler symbols and which are real *m4* macros.

BUGS

The **.align** assembler directive may not work in the **.text** section when optimization is performed.

CAVEATS

Arithmetic expressions may only have one forward referenced symbol per expression.

NOTES

Wherever possible, the assembler should be accessed through a compilation system interface program [such as *cc*(1)].

NAME

asa - interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

The *asa* command interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names are supplied. The first character of each line is assumed to be a control character, with the following meanings:

' '	(blank) Single new line before printing
0	Double new line before printing
1	New page before printing
+	Overprint previous line.

Lines beginning with other than the above characters are treated as if they began with blank. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic appears on standard error. This program forces the first line of each input file to start on a new page.

To view the output of FORTRAN programs that use ASA carriage control characters, use the following *asa* command:

```
a.out | asa | lp
```

The output, properly formatted and paginated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by using the following command:

```
asa file
```

NAME

assist - assistance using CTIX system commands

SYNOPSIS

assist [name]
assist [**-s**]
assist [**-c** name]

DESCRIPTION

The *assist* command invokes the ASSIST menu interface software for the CTIX system. The ASSIST menus categorize CTIX system commands according to function in a hierarchy. The menus lead to full-screen forms (called command forms) that aid you in the execution of a syntactically correct CTIX system command line. The menus also lead to interactive simulations of CTIX system commands or concepts (called walkthrus).

If you type *assist* without options, you enter at the top of the menu interface hierarchy. New users may want to use the -s option to select an introductory tutorial explaining how to use the ASSIST software.

Options to *assist* follow:

name Invoke an ASSIST-supported CTIX system command form or walkthru for *name*.

-c *name* Invoke the version of *name* in your current directory.

-s Reinvoke the ASSIST setup module and check or modify your terminal variable; or access the introductory information about ASSIST.

When you invoke *assist*, you perform operations within the program by using *assist* commands. To see a list of the *assist* commands, press **Control A** (control-**a**) or **F8** (function-key 8) when you are in *assist*. A list of the commands is dispalyed on standard output. The entire set of commands is described in the "Glossary of ASSIST Commands" in the *AT&T ASSIST Software User's Guide*.

EXAMPLE

This example illustrates how to go directly to a particular command form. In this case, *mkdir* is the desired command form.

```
assist mkdir
```

FILES

$HOME/.assistrc information needed by *assist* (for example, about the terminal you are using)

/usr/lib/assist	default directory containing assist command forms, walkthrus, and executable programs

NOTES

The first time you invoke *assist* it ignores any options and asks for information about the terminal. Once it has saved this information in a file named *.assistrc* in your home directory, it displays a list of basic *assist* commands and offers an introduction to ASSIST.

SEE ALSO

astgen(1).
AT&T ASSIST Software User's Guide.

NAME

astgen - generate/modify ASSIST menus and command forms

SYNOPSIS

astgen name[**.fs**]

DESCRIPTION

The *astgen* command starts is an interactive program to generate information files (ASCII text data files) that define a menu or command form used by the *assist*(1) program.

Both the *astgen* and *assist*(1) programs recognize and process information files whose names are suffixed with three characters: **.fs**. If no **.fs** file exists for the specified name, *astgen* assumes that a new menu or command form is to be created. If *name* is given without **.fs**, *astgen* automatically creates the file *name*.**fs**.

SEE ALSO

assist(1).
AT&T ASSIST Development Tools Guide.
AT&T ASSIST User's Guide.

NAME

at, batch - execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]

at *-r* job...

at *-l* [*job* ...]

batch

DESCRIPTION

at and *batch* read commands from standard input to be executed at a later time. *at* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. *at* may be used with the following options:

-r Removes jobs previously scheduled with *at*.

-l Reports all jobs scheduled for the invoking user.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use *at* if their name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** is checked to determine if the user should be denied access to *at*. If neither file exists, only root is allowed to submit a job. If **at.deny** exists and is empty, global usage is permitted. If **at.allow** exists and is empty, no usage is permitted. If **at.allow** exists, **at.deny** is ignored. The allow/deny files consist of one user name per line. These files can only be modified by the superuser.

The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour*:*minute*. A suffix **am** or **pm** may be appended; otherwise a 24-hour clock time is understood. The suffix **zulu** may be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", **today** and **tomorrow** are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **tomorrow** is assumed if it is less.

If the given month is less than the current month (and no year is given), next year is assumed.

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.)

Thus legitimate commands include:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

at and *batch* write the job number and schedule time to standard error.

batch submits a batch job. It is almost equivalent to "at now", but not quite. For one, it goes into a different queue. For another, "at now" will respond with the error message **too late**.

at **-r** removes jobs previously scheduled by *at* or *batch*. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at* **-l**. You can only remove your own jobs unless you are the super-user.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *sh*(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
sort filename >outfile
```
<Control-D> (hold down 'Control' and press 'D')

This sequence, which demonstrates how to redirect standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
sort filename 2>&1 >outfile | mail loginid
!
```

To have a job reschedule itself, invoke *at* from within the shell procedure, by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

cron(1), kill(1), mail(1), nice(1), ps(1), sh(1).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [**-Fc**] [prog] [parameters] [files]

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*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

Parameters, in the form x=... y=... etc., may be passed to *awk*.

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; see below). 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, new-lines, 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 *>expr* 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*(3S)].

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*(3S) 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* [see *grep*(1)]. 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 new-line); 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 }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
        { print }
```

command line:

```
awk -f program n=5 input
```

SEE ALSO

grep(1), lex(1), malloc(3X), nawk(1), sed(1).
UNIX System V Release 3.2 Programmer's Guide.

BUGS

Input white space is not preserved on output if fields are involved.

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 the null string ("") to it.

NAME

banner - make posters

SYNOPSIS

banner strings

DESCRIPTION

banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

echo(1).

NAME

basename, dirname - deliver portions of path names

SYNOPSIS

basename string [suffix]

dirname string

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 (` `) within shell procedures.

Dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, invoked with the argument **/usr/src/cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

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

The following example will set the shell variable NAME to **/usr/src/cmd**:

```
NAME=`dirname /usr/src/cmd/cat.c`
```

SEE ALSO

sh(1).

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

bc [**-c**] [**-l**] [file ...]

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The *bc*(1) utility is actually a preprocessor for *dc*(1), which it invokes automatically unless the **-c** option is present. In this case the *dc* input is sent to the standard output instead. The options are as follows:

-c Compile only. The output is send to the standard output.

-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 new-lines 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.

EXAMPLE

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

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

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

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

FILES

/usr/lib/lib.b	mathematical library
/usr/bin/dc	desk calculator proper

SEE ALSO

dc(1).
Programmer's Guide: CTIX Supplement.

BUGS

The *bc* command does not yet recognize the logical operators, && and | |.
For statement must have all three expressions (E's).
Quit is interpreted when read, not when executed.

NAME

bcheck - print the list of blocks associated with an i-node or i-nodes

SYNOPSIS

/usr/local/bin/bcheck [-i number] special

DESCRIPTION

The *bcheck* command prints a list of all 1024-byte blocks associated with each i-node for a filesystem on *special*, where *special* is the device name. If the **-i** *number* option is given, the printout is restricted to the i-node *number*.

EXAMPLES

```
bcheck /dev/rdsk/c0d0s1

bcheck -i 2 /dev/rdsk/c0d0s3
```

SEE ALSO

ncheck(1M).

NAME

bcopy - interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

bcopy copies from and to files starting at arbitrary block (512-byte) boundaries.

The following questions are asked:

to: (you name the file or device to be copied to).
offset: (you provide the starting "to" block number).
from: (you name the file or device to be copied from).
offset: (you provide the starting "from" block number).
count: (you reply with the number of blocks to be copied).

After **count** is exhausted, the **from** question is repeated (giving you a chance to concatenate blocks at the **to+offset+count** location). If you answer **from** with a carriage return, everything starts over.

Two consecutive carriage returns terminate *bcopy*.

SEE ALSO

cpio(1), dd(1).

NAME

bdiff - big diff

SYNOPSIS

bdiff file1 file2 [n] [-s]

DESCRIPTION

The *bdiff* command is used in a manner analogous to *diff*(1) to find which lines in two files must be changed to bring the files into agreement. Its purpose is to allow processing of files which are too large for *diff*.

The parameters to *bdiff* are:

file1 (file2)
: The name of a file to be used. If *file1 (file2)* is -, the standard input is read.

n
: The number of line segments. The value of *n* is 3500 by default. If the optional third argument is given and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail.

-s
: Specifies that no diagnostics are to be printed by *bdiff* (silent option). Note, however, that this does not suppress possible diagnostic messages from *diff*(1), which *bdiff* calls.

bdiff ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

SEE ALSO

diff(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

bfs - big file scanner

SYNOPSIS

bfs [-] name

DESCRIPTION

The *bfs* command is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). *bfs* is usually more efficient than *ed*(1) for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional - suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed, as in *ed*(1). Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed*(1) are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **p**, **q**, **w**, =, **!** and null commands operate as described under *ed*(1). Commands such as **---**, **+++-**, **+++=**, **-12**, and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no *remembered* file name. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt** and **xc** commands, below). The following additional commands are available:

xf*file* Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the **xf**. The **xf** commands may be nested to a depth of 10.

xn List the marks currently in use (marks are set by the **k** command).

xo [*file*] Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666 (readable and writable by everyone), unless your *umask* setting [see *umask*(1)]

dictates otherwise. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label* This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the : and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(.,.)**xb**/*regular expression*/*label*

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, **.** is set to the line matched and a jump is made to *label*. This command is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.

The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt *number* Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

xv[*digit*] [*spaces*] [*value*]

The variable name is the specified *digit* following the **xv**. The commands **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. The command **xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6**, the following statements all print the first 100 lines:

```
1,%5p
1,%5
%6
```

```
g/%5/p
```

globally searches for the characters **100** and print each line containing a match. To escape the special meaning of **%**, a \ must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.

Another feature of the **xv** command is that the first line of output from a CTIX command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a \.

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*

xbn *label* These two commands will test the last saved *return code* from the execution of a CTIX command (**!***command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string **size**.

```
xv55
:l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
xv45
:l
/size/
```

```
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz !
```

xc [*switch*] If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it is not. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

SEE ALSO

csplit(1), ed(1), umask(1).

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

NAME

brc, bcheckrc, drvload, powerfail - system initialization procedures

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/drvload

/etc/powerfail

DESCRIPTION

These shell procedures are executed via entries in **/etc/inittab** by *init*(1M) whenever the system is booted (or rebooted). Except for *powerfail*, they are run when the system is changed out of *SINGLE USER* mode. *powerfail* is executed whenever a system power failure is detected.

First, the *bcheckrc* procedure performs all the necessary consistency checks to prepare the system to change into multi-user mode. It actually contains two procedures: an interactive procedure that runs *fsck*(1M) and sets the time; and a noninteractive procedure that only checks the file system. *bcheckrc* looks for the presence of a file named **/etc/rcopts/BCRCCON**: if there is such a file, the interactive procedure is selected. If the noninteractive procedure is selected and it fails because of file system problems or because it was interrupted from the controlling terminal, *bcheckrc* switches the system to state 6, which is normally CTIX Administrator Mode. *bcheckrc* also sets the date to the date currently in the time-of-day clock.

Then, the *brc* procedure clears the mounted file system table, **/etc/mnttab** and puts the entry for the root file system into the mount table.

The *drvload* procedure causes any desired device drivers and additional swap areas to be loaded into the system. The namelist of the running system (**/etc/lddrv/unix.exec**) is built up, starting with **/unix** and adding each of the loaded drivers. This procedure uses *hinv*(1M) to determine what hardware exists and then loads the appropriate drivers. In addition, a number of files in **/etc/rcopts** control the loading of drivers that are not associated with hardware.

The *powerfail* procedure is invoked when the system detects a power failure condition. It calls *shutdown*(1M) to bring down the system gracefully.

After the three boot procedures have executed, *init* checks for the *initdefault* value in **/etc/inittab**. This tells *init* in which run level to place the system. Since *initdefault* is initially set to **2**, the system is placed in the multi-user state by use of the */etc/rc2* procedure.

Note that *bcheckrc* should always be executed before *brc*. Also, these shell procedures can be used for several run-level states.

FILES

/unix
/etc/log/confile
/etc/rcopts/BCRCCON

SEE ALSO

conlocate(1M), date(1), fsck(1M), hinv(1M), init(1M), rc0(1M), rc2(1M), shutdown(1M), who(1), inittab(4), mnttab(4).
S/Series CTIX Administrator's Guide.

NAME

bs - a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [file [args]]

DESCRIPTION

bs is a remote descendant of Basic and Snobol4 with a little C language thrown in. *bs* is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the *trace* and *dump* statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see *compile* and *execute* below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

bs programs are made up of input lines. If the last character on a line is a \, the line is continued. *bs* accepts lines of the following form:

```
statement
label statement
```

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A *bs* statement is either an expression or a keyword followed by zero or more expressions. Some keywords (*clear*, *compile*, *!*, *execute*, *include*, *ibase*, *obase*, and *run*) are always executed as they are compiled.

Statement Syntax

expression

The expression is executed for its side effects (value, assignment, or function call). The details of expressions follow the description of statement types below.

break

Break exits from the inner-most *for*/*while* loop.

clear

Clears the symbol table and compiled statements. *Clear* is executed immediately.

compile
[expression]
Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A *clear* is associated with this latter case. *Compile* is executed immediately.

continue
Continue transfers to the loop-continuation of the current *for*/*while* loop.

dump
[name]
The name and current value of every non-local variable is printed. Optionally, only the named variable is reported. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.

exit
[expression]
Return to system level. The expression is returned as process status.

execute
Change to immediate execution mode (an interrupt has a similar effect). This statement does not cause stored statements to execute (see *run* below).

for
name = expression expression statement
for
name = expression expression
...
next

for
expression , expression , expression statement
for
expression , 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. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

fun
f([a, ...]) [v, ...]
...

nuf

Fun defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn

A way to signal the failure of a user-written function. See the interrogation operator (?) below. If interrogation is not present, *freturn* merely returns zero. When interrogation *is* active, *freturn* transfers to that expression (possibly by-passing intermediate function returns).

goto
name

Control is passed to the internally stored statement with the matching label.

ibase
N

Ibase sets the input base (radix) to *N*. The only supported values for *N* are **8**, **10** (the default), and **16**. Hexadecimal values 10-15 are entered as **a-f**. A leading digit is required (that is, **f0a** must be entered as **0f0a**). *Ibase* (and *obase*, below) are executed immediately.

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. The strings **0** and "" (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The elision of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if* ... *elif* ... [*else* ...] sequence.

include expression

The expression must evaluate to a file name. The file must contain *bs* source statements. Such statements become part of the program being compiled. *Include* statements may not be nested.

obase
N
Obase sets the output base to *N* (see *ibase* above).

onintr
label
onintr
The *onintr* command provides program control of interrupts. In the first form, control will pass to the label given, just as if a *goto* had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause *bs* to terminate.

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 random number generator is reset. Control is passed to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop
Execution of internal statements is stopped. *bs* reverts to immediate mode.

trace
[expression]
The *trace* statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace* expression value.

while
expression statement
while
expression
...
next
While is similar to *for* except that only the conditional expression for loop-continuation is given.

! shell command
An immediate escape to the Shell.

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

Expression Syntax

name

A name is used to specify a variable. Names are composed of a letter (upper or lower case) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open*( ) below).

name

([expression [, expression] ...])

Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value.

name

[expression [, expression] ...]

This syntax is used to reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference 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.

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.

string

Character strings are delimited by " characters. The \ escape character allows the double quote (\"), new-line (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. Otherwise, \ stands for itself.

(expression)

Parentheses are used to alter the normal order of evaluation.

(expression,

expression [, expression ...]) [expression]

The bracketed expression is used as a subscript to select a comma-separated

expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

(False, True)[a == b]

has the value **True** if the comparison is true.

? expression
The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the PROGRAMMING TIPS section below), the result of the *eval* built-in function, and for checking the return from user-written functions (see *freturn*). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

- expression
The result is the negation of the expression.

++ name
Increments the value of the variable (or array reference). The result is the new value.

-- name
Decrements the value of the variable. The result is the new value.

! expression
The logical negation of the expression. Watch out for the shell escape command.

expression
operator expression
Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence)

=

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

_

_ (underscore) is the concatenation operator.

& |

& (logical and) has result zero if either of its arguments are zero. It has result one if both of 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 is non-zero. Both operators treat a null string as a 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*. A string comparison is made if both operands are strings.

+ -

Add and subtract.

*** / %**

Multiply, divide, and remainder.

^

Exponentiation.

Built-in Functions

Dealing with arguments

arg(i)

is the value of the *i*-th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument [*arg*(0) returns **bs**].

narg( )

returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

abs(x)

is the absolute value of *x*.

atan(x)

is the arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.

ceil(x)

returns the smallest integer not less than *x*.

cos(x)

is the cosine of *x* (radians).

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

floor(x)
returns the largest integer not greater than *x*.

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

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

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

sqrt(x)
is the square root of *x*.

String operations

size(s)
the size (length in bytes) of *s* is returned.

format(f, a)
returns the formatted value of *a*. *F* is assumed to be a format specification in the style of *printf*(3S). Only the **%...f**, **%...e**, and **%...s** types are safe.

index(x, y)
returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

trans(s, f, t)
Translates characters of the source *s* from matching characters in *f* to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

substr(s, start, width)
returns the sub-string of *s* defined by the *start*ing position and *width*.

match(string, pattern)
mstring(n)
The *pattern* is similar to the regular expression syntax of the *ed*(1) command. The characters ., [,], ^ (inside brackets), * and $ are special. The *mstring* function returns the *n*-th (1 <= *n* <= 10) substring of the subject that occurred between pairs of the pattern symbols \(and \) for the most recent call to *match*. To succeed, patterns must match the beginning

of the string (as if all patterns began with ^). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a-z]\)") == 6
mstring(1) == "b"
```

File handling

open(name, file, function)
close(name)

The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may be 1) a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively; 2) a string representing a file name; or 3) a string beginning with an **!** representing a command to be executed (via *sh -c*). The *function* argument must be either **r** (read), **w** (write), **W** (write without new-line), or **a** (append). After a *close*, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

access(s, m)

executes *access*(2).

ftype(s)

returns a single character file type indication: **f** for regular file, **p** for FIFO (that is, named pipe), **d** for directory, **b** for block special, or **c** for character special.

Tables

table(name, size)

A table in *bs* is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a *bs* variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. *bs* prints an error message and stops on table overflow.

item(name, i)
key()

The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the "subscript" of the previous *item* call.

The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table; for example:

```
table("t", 100)
 ...
# If word contains "party", the following expression
# adds one to the count of that word:
++t[word]
 ...
# To print out the the key/value pairs:
for i = 0, ?(s = item(t, i)), ++i  if key()  put = key()_":"_s
```

iskey(name, word)

The *iskey* function tests whether the key **word** exists in the table **name** and returns one for true, zero for false.

Odds and ends

eval(s)

The string argument is evaluated as a *bs* expression. The function is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++"_ name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control *bs* error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a *goto* to the label *L* (if it exists):

```
label="L"
if !(?eval("goto "_ label)) puterr = "no label"
```

plot(request, args)

The *plot* function produces output on devices recognized by *tplot*(1G). The *requests* are as follows:

Call	*Function*
plot(0, term)	causes further *plot* output to be piped into *tplot*(1G) with an argument of -T*term*.

plot(4) "erases" the plotter.

plot(2, string) labels the current point with *string*.

plot(3, x1, y1, x2, y2)
draws the line between (*x1*,*y1*) and (*x2*,*y2*).

plot(4, x, y, r) draws a circle with center (*x*,*y*) and radius *r*.

plot(5, x1, y1, x2, y2, x3, y3)
draws an arc (counterclockwise) with center (*x1*,*y1*) and endpoints (*x2*,*y2*) and (*x3*,*y3*).

plot(6) is not implemented.

plot(7, x, y) makes the current point (*x*,*y*).

plot(8, x, y) draws a line from the current point to (*x*,*y*).

plot(9, x, y) draws a point at (*x*,*y*).

plot(10, string) sets the line mode to *string*.

plot(11, x1, y1, x2, y2)
makes (*x1*,*y1*) the lower left corner of the plotting area and (*x2*,*y2*) the upper right corner of the plotting area.

plot(12, x1, y1, x2, y2)
causes subsequent x (y) coordinates to be multiplied by *x1* (*y1*) and then added to *x2* (*y2*) before they are plotted. The initial scaling is **plot(12, 1.0, 1.0, 0.0, 0.0)**.

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot*(1G). See *plot*(4) for more details.

last( )
in immediate mode, *last* returns the most recently computed value.

PROGRAMMING TIPS

Using *bs* as a calculator:

```
$ bs
#   Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

#   Compound interest (6% for 5 years on $1,000).
int = .06 / 4
```

```
bal = 1000
for i = 1 5*4  bal = bal + bal*int
bal - 1000
3346.855007
...
exit
```

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while  ?(str = read)
            ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
#   Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

#   Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls)  ...
...
```

```
# be sure to close (wait for) these:
close("ls")
close("pr")
```

SEE ALSO

ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), plot(4).

NAME

cal - print calendar

SYNOPSIS

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. If neither is specified, a calendar for the present 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 the United States.

EXAMPLES

An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type: **cal 9 1752**

BUGS

The year is always considered to start in January even though this is historically naive.

Beware that "cal 83" refers to the early Christian era, not the 20th century.

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

The *calendar* command consults the file **calendar** in the current directory and displays lines that contain today's or tomorrow's date anywhere in the line.

The command expects American date format, not European, and does not accept tabs within a date. Most reasonable month-day dates, such as "Aug. 24," "august 24," and "8/24," are recognized (but not "24 August" or "24/8"). 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 or her login directory and sends them any positive results by *mail*(1). Normally this is performed daily by facilities in the CTIX operating system.

FILES

/usr/lib/calprog	to figure out today's and tomorrow's dates
/etc/passwd	
/tmp/cal*	

SEE ALSO

mail(1).

BUGS

Your calendar must be public information for you to get reminder service.

The *calendar* command's extended idea of "tomorrow" does not account for holidays.

NAME

captoinfo - convert a termcap description into a terminfo description

SYNOPSIS

captoinfo [**-v** ...] [**-V**] [**-1**] [**-w** width] file ...

DESCRIPTION

The *captoinfo* command looks in *file* for *termcap* descriptions. For each one found, an equivalent *terminfo* (4) description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the *termcap tc*= field) will be reduced to the minimum superset before being output.

If no *file* is given, then the environment variable **TERMCAP** is used for the filename or entry. If **TERMCAP** is a full pathname to a file, only the terminal whose name is specified in the environment variable **TERM** is extracted from that file. If the environment variable **TERMCAP** is not set, then the file */etc/termcap* is read.

-v print out tracing information on standard error as the program runs. Specifying additional **-v** options causes more detailed information to be printed.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

-w change the output to *width* characters.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

CAVEATS

Certain *termcap* defaults are assumed to be true. For example, the bell character (*terminfo bel*) is assumed to be ^G. The linefeed capability (*termcap nl*) is assumed to be the same for both *cursor_down* and *scroll_forward* (*terminfo cud1* and *ind*, respectively.) Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for *termcap* fields such as *cursor_position* (*termcap cm*, *terminfo cup*) will sometimes produce a string which, though technically correct, may not be optimal. In particular, the rarely used *termcap* operation **%n** will produce strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a *termcap* entry, a hold-over from an earlier version of the CTIX system, has been removed.

DIAGNOSTICS

tgetent failed with return code n (reason).
The termcap entry is not valid. In particular, check for an invalid 'tc=' entry.

unknown type given for the termcap code cc.
The termcap description had an entry for *cc* whose type was not boolean, numeric or string.

wrong type given for the boolean (numeric, string) termcap code cc.
The boolean *termcap* entry *cc* was entered as a numeric or string capability.

the boolean (numeric, string) termcap code cc is not a valid name.
An unknown *termcap* code was specified.

tgetent failed on TERM=term.
The terminal type specified could not be found in the *termcap* file.

TERM=term: cap cc (info ii) is NULL: REMOVED
The *termcap* code was specified as a null string. The correct way to cancel an entry is with an '@', as in ':bs@:'. Giving a null string could cause incorrect assumptions to be made by the software which uses *termcap* or *terminfo*.

a function key for cc was specified, but it already has the value vv.
When parsing the **ko** capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

the unknown termcap name cc was specified in the ko termcap capability.
A key was specified in the **ko** capability which could not be handled.

the vi character v (info ii) has the value xx, but ma gives n.
The **ma** capability specified a function key with a value different from that specified in another setting of the same key.

the unknown vi key v was specified in the ma termcap capability.
A *vi*(1) key unknown to *captoinfo* was specified in the **ma** capability.

Warning: termcap sg (nn) and termcap ug (nn) had different values.
terminfo assumes that the **sg** (now **xmc**) and **ug** values were the same.

Warning: the string produced for ii may be inefficient.
The parameterized string being created should be rewritten by hand.

Null termname given.
The terminal type was null. This is given if the environment variable **TERM** is not set or is null.

cannot open file for reading.
The specified file could not be opened.

SEE ALSO

infocmp(1M), tic(1), curses(3X), terminfo(4).
UNIX System V Release 3.2 Programmer's Guide.

NOTES

captoinfo should be used to convert *termcap* entries to *terminfo*(4) entries because the *termcap* database (from earlier versions of CTIX) may not be supplied in future releases.

NAME

cat - concatenate and print files

SYNOPSIS

cat [**-u**] [**-s**] [**-v** [**-t**] [**-e**]] file ...

DESCRIPTION

The *cat* command reads each *file* in sequence and writes it on the standard output. Thus, the following command prints **file** to the terminal:

```
cat file
```

The next command concatenates **file1** and **file2**, and writes the results in **file3**:

```
cat file1 file2 > file3
```

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

The following options apply to *cat*:

-u The output is not buffered. (The default is buffered output.)

-s Causes *cat* to be silent about non-existent files.

-v Causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. ASCII control characters (octal 000 - 037) are printed as ^*n*, where *n* is the corresponding ASCII character in the range octal 100 - 137 (@, A, B, C, . . ., X, Y, Z, [, \,], ^, and _); the DEL character (octal 0177) is printed ^?. Other non-printable characters are printed as M-*x*, where *x* is the ASCII character specified by the low-order seven bits.

When used with the **-v** option, the following options are available:

-t Causes tabs to be printed as ^I's.

-e Causes a $ character to be printed at the end of each line (prior to the new-line).

The **-t** and **-e** options are ignored if the **-v** option is not specified.

WARNING

Redirecting the output of **cat** onto one of the files being read causes the loss of the data originally in the file being read. For example, the following command causes the original data in **file1** to be lost:

```
cat file1 file2 >file1
```

SEE ALSO

cp(1), pg(1), pr(1).

NAME

cb - C program beautifier

SYNOPSIS

cb [**-s**] [**-j**] [**-l** leng] [file ...]

DESCRIPTION

The *cb* comand reads C programs either from its arguments or from the standard input, and writes them on the standard output with spacing and indentation that display the structure of the code. Under default options, *cb* preserves all user new-lines.

cb accepts the following options.

-s Canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*.

-j Causes split lines to be put back together.

-l *leng* Causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).

The C Programming Language. Prentice-Hall, 1978.

BUGS

Punctuation that is hidden in preprocessor statements will cause indentation errors.

NAME

cc - C compiler

SYNOPSIS

cc [options] files

DESCRIPTION

The *cc* command is the interface to the C Compilation System. The compilation tools consist of a preprocessor, compiler, optimizer, assembler and link editor. The *cc* command processes the supplied options and then executes the various tools with the proper arguments. The *cc* command accepts several types of files as arguments:

Files whose names end with **.c** are taken to be C source programs and may be preprocessed, compiled, optimized, assembled and link edited. The compilation process may be stopped after the completion of any pass if the appropriate options are supplied. If the compilation process runs through the assembler then an object program is produced and is left in the file whose name is that of the source with **.o** substituted for **.c**. However, the **.o** file is normally deleted if a single C program is compiled and then immediately link edited. In the same way, files whose names end in **.s** are taken to be assembly source programs, and may be assembled and link edited; and files whose names end in **.i** are taken to be preprocessed C source programs and may be compiled, optimized, assembled and link edited. Files whose names do not end in **.c**, **.s** or **.i** are handed to the link editor.

Since the *cc* command usually creates files in the current directory during the compilation process, it is necessary to run the *cc* command in a directory in which a file can be created. The following options are interpreted by *cc*.

-#
-##
-### These options cause *cc* to display each command that it would generate if it were to execute, but to fully execute only in the case of **-#**. Thus, **-#** specifies execution in verbose mode; **-##** specifies verbose mode (what *cc* would do if it were to execute), check permissions on all necessary files, but do not compile; and **-###** specifies verbose mode (what *cc* would do if it were to execute), but do nothing.

-c Suppress the link editing phase of the compilation, and do not remove any produced object files.

-g Cause the compiler to generate additional information needed for the use of *sdb* (1).

-o *outfile*
Produce an output object file by the name *outfile*. The name of the default file is **a.out**. This is a link editor option.

-p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing takes place, profiled versions of libc.a and libm.a (with -lm option) are linked and *monitor*(3C) is automatically called. A **mon.out** file will then be produced at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

-w Tell the linker (*ld*) not to print warnings about symbols that partially matched. This option is meaningful only when the -T option is also specified.

-B*string*
-t[p02al]
These options will be removed in the next release. Use the **-Y** option.

-E Run only *cpp*(1) on the named C programs, and send the result to the standard output.

-H Print out on *stderr* the pathname of each file included during the current compilation.

-O Do compilation phase optimization. This option will not have any affect on .s files.

-P Run only *cpp*(1) on the named C programs and leave the result in corresponding files suffixed .i. This option is passed to *cpp*(1).

-S Compile and do not assemble the named C programs, and leave the assembler-language output in corresponding files suffixed .s.

-T Truncate variable names to eight characters. Tell the loader to match eight character names (same as -G in the loader).

-Wc,arg1[,arg2...]
Hand off the argument[s] *argi* to pass *c* where *c* is one of [**p02al**] indicating the preprocessor, compiler, optimizer, assembler, or link editor, respectively. For example: **-Wa,-m** passes **-m** to the assembler.

-Y [p02alSILUc],*dirname* | *processor*
Specify a new pathname, *dirname*, for the locations of the tools and directories designated in the first argument; or select a processor type, *processor*, for which to generate code. **[p02alSILUc]** represents:

p preprocessor
0 compiler
2 optimizer
a assembler
l link editor
S directory containing the start-up routines
I default include directory searched by *cpp*(1)
L first default library directory searched by *ld*(1)
U second default library directory searched by *ld*(1)
c select the processor type, specified by the second argument, for which to generate code: **68020**, **68010**, **68881**. For example, **-Y c,68020** selects the 68020 processor with software floating point instructions. Note that **68881** implies **68020**.

If the location of a tool is being specified, then the new pathname for the tool will be *dirname/tool*. If more than one **-Y** option is applied to any one tool or directory, then the last occurrence holds.

The *cc* command also recognizes **-C**, **-D**, **-H**, **-I** and **-U** and passes these options and their arguments directly to the preprocessor without using the **-W** option. Similarly, the *cc* command recognizes **-l**, **-m**, **-o**, **-r**, **-s**, **-t**, **-u**, **-w**, **-x**, **-z**, **-F**, **-G**, **-L**, **-M**, **-N**, **-V** and **-Z** and passes these options and their arguments directly to the loader. See the manual pages for *cpp*(1) and *ld*(1) for descriptions.

Other arguments are taken to be C compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C compatible routines and are passed directly to the link editor. These programs, together with the results of any compilations specified, are link edited (in the order given) to produce an executable program with name **a.out** unless the **-o** option of the link editor is used.

If the *cc* command is put in a file *prefix*cc the prefix will be parsed off the command and used to call the tools, that is, *prefix*tool. For example, OLDcc will call OLDcpp, OLDcomp, OLDoptim, OLDas, and OLDld and will link OLDcrt1.o. Therefore, one MUST be careful when moving the *cc* command around. The prefix will apply to the preprocessor, compiler, optimizer, assembler, link editor, and the start-up routines.

The C compiler uses one of three code generators for the 68010, 68020, and 68020/68881. There are several ways to select a particular code generator, but the selection is normally done using one of two basic mechanisms.

The first is to specify the processor on the *cc* command line, for example, by using the **-Y** option. (An equivalent mechanism is provided by the *gencc*(1) command, and also by the *cc1sw*(1), *cc2sw*, or *cc2fp* command.) The **-Y** option

has additional arguments that allow you to specify pathnames of default libraries, include files, and tools as described earlier.

The second mechanism is to use the CENVIRON shell variable. Note that the first mechanism, specifying the processor and/or search path of libraries and include files, overrides the CENVIRON and any other shell variable settings.

The CENVIRON variable has the following syntax:

CPU=xxxxx,FPU=yyyyy

where CPU indicates the central processor to generate for and FPU indicates the style of floating-point math to use. *xxxxx* may be 68010 or 68020, and *yyyyy* may be 68881 or SOFTWARE. The FPU parameter may be omitted; the default is SOFTWARE. The CENVIRON variable should always be set to the appropriate values in the **.profile** or **.cshrc** files or in the makefile. [See *hinv*(1M).]

The C compiler interprets two shell variables which, along with the CENVIRON variable, allow cross-compilation for any CTIX machine:

LIBROOT — This variable is a path which is prepended to normal library names when searching for a library. See also *ld*(1).

INCROOT — This variable is a path which is prepended to the **/usr/include** and **/usr/include/sys** directories during include file searches. See also *cpp*(1).

The C language standard was extended to allow arbitrary length variable names. The option pair "**-Wp,-T -W0,-XT**" will cause *cc* to truncate arbitrary length variable names to 8 characters.

FILES

file.c	C source file
file.i	preprocessed C source file
file.o	object file
file.s	assembly language file
a.out	link edited output
LIBDIR/*crt1.o	start-up routine
LIBDIR/crtn.o	start-up routine
TMPDIR/*	temporary files
LIBDIR/cpp	preprocessor, *cpp*(1)
LIBDIR/ccom	68010 compiler
LIBDIR/ccom20	68020 compiler
LIBDIR/ccom20.81	68020/68881 compiler
LIBDIR/optim	optimizer
BINDIR/as	assembler, *as*(1)

BINDIR/ld	link editor, *ld*(1)
LIBDIR/libc.a	standard C library
LIBDIR/libc_s.a	standard C shared library

LIBDIR is usually /lib
BINDIR is usually /bin
TMPDIR is usually /tmp but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*( ) in *tmpnam*(3S)].

SEE ALSO

as(1), ld(1), cc1sw(1), cpp(1), gencc(1), lint(1), prof(1), sdb(1), tmpnam(3S).
Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, 1978.

CAVEATS

cc will complain if it encounters inconsistencies between the processor selected and default libraries or include files.

DIAGNOSTICS

The diagnostics produced by the C compiler are sometimes cryptic. Occasional messages may be produced by the assembler or link editor.

NOTES

By default, the return value from a compiled C program is completely random. The only two guaranteed ways to return a specific value is to explicitly call *exit*(2) or to leave the function **main**( ) with a "*return expression;*" construct.

NAME

cc1sw, cc2sw, cc2fp - front-end to the cc command

SYNOPSIS

cc1sw [options] files
cc2sw [options] files
cc2fp [options] files

DESCRIPTION

cc1sw, *cc2sw*, and *cc2fp* provide a front-end to *cc* for use in cross-compilation. *cc1sw* generates code for a 68010 processor with software floating point, *cc2sw* generates code for a 68020 processor with software floating point, and *cc2fp* generates code for a 68020 processor with hardware floating point. The commands call *cc* with the following **-Y** options:

cc1sw **-Y c,68010**
-Y S,/cross/1sw/lib
-Y L,/cross/1sw/lib
-Y U,/cross/1sw/usr/lib

cc2sw **-Y c,68020**
-Y S,/cross/2sw/lib
-Y L,/cross/2sw/lib
-Y U,/cross/2sw/usr/lib

cc2fp **-Y c,68881**
-Y S,/cross/2fp/lib
-Y L,/cross/2fp/lib
-Y U,/cross/2fp/usr/lib

Options are those options available for *cc*.

The default include directories searched by *cc* (called by *cc1sw*, *cc2sw*, or *cc2fp*) are **/usr/include** and **/usr/include/sys**. The default include directories can be overridden by using the **-Y I,***dirname* option or setting the INCROOT environment variable [see *cc*(1)].

FILES

file.c	C source file
file.i	preprocessed C source file
file.o	object file
file.s	assembly language file
a.out	link edited output
LIBDIR/*crt1.o	start-up routine
LIBDIR/crtn.o	start-up routine

TMPDIR/*	temporary files
LIBDIR/cpp	preprocessor, *cpp*(1)
LIBDIR/ccom	68010 compiler
LIBDIR/ccom20	68020 compiler
LIBDIR/ccom20.81	68020/68881 compiler
LIBDIR/optim	optimizer
BINDIR/as	assembler, *as*(1)
BINDIR/ld	link editor, *ld*(1)
LIBDIR/	standard C library
LIBDIR/libc_s.a	standard C shared library

LIBDIR is usually /lib
BINDIR is usually /bin
TMPDIR is usually /tmp but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*( ) in *tmpnam*(3S)].

SEE ALSO

cc(1), gencc(1M).

NAME

cd - change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **$HOME** is used as the new working directory. If *directory* specifies a complete path starting with /, ., .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable. *cd* 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; therefore, it is recognized and is internal to the shell.

SEE ALSO

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

NAME

cdc - change the delta commentary of an SCCS delta

SYNOPSIS

cdc -rSID [**-m**[mrlist]] [-y[comment]] files

DESCRIPTION

The *cdc* command changes the *delta commentary*, for the SID (SCCS **ID**entification string) specified by the **-r** keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the *delta*(1) command (**-m** and **-y** keyletters).

If a directory is named, *cdc* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see *WARNINGS*) and each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to *cdc*, which may appear in any order, consist of *keyletter* arguments and file names.

All the described *keyletter* arguments apply independently to each named file:

-r*SID* Used to specify the *S*CCS *ID*entification (SID) string of a delta for which the delta commentary is to be changed.

-m*[mrlist]* If the SCCS file has the v flag set [see *admin*(1)] then a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the **-r** keyletter *may* be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of *delta*(1). In order to delete an MR, precede the MR number with the character ! (see *EXAMPLES*). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If **-m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **-y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the **v** flag has a value [see *admin*(1)], it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, *cdc* terminates and the delta commentary remains unchanged.

-y*[comment]* Arbitrary text used to replace the *comment*(s) already existing for the delta specified by the **-r** keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

Simply stated, the keyletter arguments are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file

adds bl78-12345 and bl79-00001 to the MR list, removes bl77-54321 from the MR list, and adds the comment **trouble** to delta 1.6 of s.file.

cdc -r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble

does the same thing.

WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (- on the command line), then the **-m** and **-y** keyletters must also be used.

FILES

x-file [see *delta*(1)]
z-file [see *delta*(1)]

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
UNIX System V Release 3.2 Programmer's Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cflow - generate C flowgraph

SYNOPSIS

cflow [**-r**] [**-ix**] [**-i_**] [**-dnum**] files

DESCRIPTION

The *cflow* command analyzes a collection of C, yacc, lex, assembler, and object files and attempts to build a graph charting the external references. Files suffixed with **.y**, **.l**, and **.c** are *yacc*ed, *lex*ed, and C-preprocessed as appropriate. The results of the preprocessed files, and files suffixed with **.i**, are then run through the first pass of *lint*(1). Files suffixed with **.s** are assembled. Assembled files, and files suffixed with **.o**, have information extracted from their symbol tables. The results are collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference number, followed by a suitable number of tabs indicating the level, then the name of the global symbol followed by a colon and its definition. Normally only function names that do not begin with an underscore are listed (see the **-i** options below). For information extracted from C source, the definition consists of an abstract type declaration (for example, **char ***), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (for example, *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only **< >** is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}

f()
{
        i = h();
}
```

the command

```
cflow -ix file.c
```

produces the output:

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the output of *cflow* can be piped to *pr*(1), using the **-e** option, to compress the tab expansion to something less than every eight spaces.

In addition to the **-D**, **-I**, and **-U** options [which are interpreted just as they are by *cc*(1) and *cpp*(1)], the following options are interpreted by *cflow*:

-r Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

-ix Include external and static data symbols. The default is to include only functions in the flowgraph.

-i_ Include names that begin with an underscore. The default is to exclude these functions (and data if *-ix* is used).

-dnum The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be ignored.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (for example, the C-preprocessor).

SEE ALSO

as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

chkshlib - compare shared libraries tool

SYNOPSIS

chkshlib [**-b**] [**-i**] [**-n**] [**-v**] file1 [file2 file3 ...]

DESCRIPTION

The *chkshlib* tool checks for compatibility between files. Input files can be combinations of host shared libraries, non-stripped target shared libraries, and non-stripped executable files. A file is compatible with another file if every library symbol in it that should be matched is matched in the second (that is, the symbol exists and has the same address in both files). The pathname for the target shared library in both files must be identical (unless the **-i** option is set.)

It is possible for *file1* to be compatible with *file2* without the reverse also being true.

If one incompatibility is found it is reported to stdout and processing stops (unless the **-v** option is set.)

The options to *chkshlib* follow:

-v Cause verbose reporting of all incompatibilities to stdout.

-b If there are symbols found in *file1* that are not in the bounds of *file2* report warning messages to stderr.

-i Turn off the restriction that the pathnames for the target shared library need to be identical for two files to be compatible.

-n Indicate that there are exactly two input files, which are target shared libraries, where the first references symbols in the second (''includes'' the second).

The output of *chkshlib* depends upon the input. If the first input file is an executable file and the other input files, if any, are target shared libraries, the output states whether or not the executable file can execute using each target shared library. If there are no target shared libraries supplied, *chkshlib* performs the compatibility check against the target shared libraries specified in the **.lib** section of the executable file.

If the first input file is an executable file and the other input file(s) is a host shared library, the output states whether or not the executable file could have been produced using each host.

If one input file is a host shared library and the other input file, if any, is a target shared library, the output states whether or not the host shared library could produce executable files that run with the target shared library. If no target

shared library is supplied, then *chkshlib* performs the compatibility check against the target specified in the **.lib** section of the library definition file found in the host.

If both input files are target shared libraries or both input files are host shared libraries, the output states whether or not the first file could replace the second and vice versa.

If both input files are target libraries and the **-n** option is set, the output states if the first file references symbols in the second file (''includes'' the second).

Compatibility of all other combinations of host shared libraries, target shared libraries, and executable files has no useful meaning and these other combinations of files are not accepted as valid input to *chkshlib*.

SEE ALSO

mkshlib(1).
''Shared Libraries'' chapter in the *UNIX System V Release 3.2 Programmer's Guide*.

DIAGNOSTICS

Exit status is 0 if no incompatibilities are found, 1 if an incompatibility is found, and 2 if a processing error occurs.

CAVEAT

The *chkshlib* command requires that you use the **-i** option whenever you use the **-n** option.

Standard binaries distributed with the UNIX system are stripped and *chkshlib* cannot be used with them.

NAME

chmod - change mode

SYNOPSIS

chmod mode file ...

chmod mode directory ...

DESCRIPTION

The permissions of the named *files* or *directories* are changed according to **mode**, which can be symbolic or absolute. Absolute changes to permissions are stated by using octal numbers, as follows:

chmod *nnn file(s)*

where *n* is a number from 0 to 7. Symbolic changes are stated by using mnemonic characters, as follows:

chmod *a operator b file(s)*

where *a* is one or more characters corresponding to **user**, **group**, or **other**; where *operator* is +, −, and =, signifying assignment of permissions; and where *b* is one or more characters corresponding to type of permission.

An absolute mode is given as an octal number constructed from the OR of the following modes:

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1** enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit is turned on [see *chmod*(2)]
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

Symbolic changes are stated by using letters that correspond to access classes and to the individual permissions. Permissions to a file can vary depending on user identification number (UID) or group identification number (GID). Permissions are described in three sequences, each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (meaning that user, group, and others all have reading, writing, and execution permission to a given file) demonstrates two categories for granting permissions: the access class and the permissions.

The following command syntax shows how to change the mode of a file's (or directory's) permissions by using *chmod*'s symbolic method:

[*who*] *operator* [*permission(s)*], ...

A command line using the symbolic method appears as follows:

chmod g+rw *file*

The above command makes *file* readable and writable by the group.

The *who* part of the symbolic method syntax can be stated as one or more of the following letters:

u	user's permissions
g	group's permissions
o	others' permissions

The letter **a** (all) is equivalent to **ugo**, and is the default if *who* is omitted.

Operator can be a plus sign (+) to add *permission* to the file's mode, a minus sign (-) to take away *permission*, or an equal sign (=) to assign *permission* absolutely. (Unlike other symbolic operations, = has an absolute effect in that it resets all other bits.) Omitting *permission* is useful only with = to take away all permissions.

Permission is any compatible combination of the following letters:

r	reading permission
w	writing permission
x	execution permission
s	user or group set-ID is turned on
t	sticky bit is turned on
l	mandatory locking will occur during access

Multiple symbolic modes separated by commas are valid, although no spaces can intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter **s** is meaningful only with **u** or **g**, and **t** works only with **u**.

Mandatory file and record locking (**l**) refers to a file's ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution

at the same time. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. The following examples are, therefore, illegal and elicit error messages:

chmod g+x,+l *file*
chmod g+s,+l *file*

Only the owner of a file or directory (or the super-user) can change a file's mode. Only the super-user can set the sticky bit on a nondirectory file; otherwise, if a regular user uses *chmod* +t, *chmod* masks the sticky bit but does not return an error. To turn on a file's set-group-ID, your own group ID must correspond to the file's, and group execution must be set.

EXAMPLES

The following two examples deny execution permission to all; the second, absolute (octal), example permits only reading permissions:

```
chmod a-x file
chmod 444 file
```

The following two examples make a file readable and writable by the group and others:

```
chmod go+rw file
chmod 666 file
```

The following example causes a file to be locked during access:

```
chmod +l file
```

The last two examples enable all to read, write, and execute the file; they also turn on the set-group-ID.

```
chmod =rwx,g+s file
chmod 2777 file
```

SEE ALSO

ls(1), chmod(2).

NOTES

In a Remote File Sharing environment, you do not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the *S/Series CTIX Administrator's Guide*.

NAME

chown, chgrp - change owner or group

SYNOPSIS

chown owner file ...

chown owner directory ...

chgrp group file ...

chgrp group directory ...

DESCRIPTION

chown changes the owner of the *files* or *directories* 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 the *files* or *directories* to *group*. The group may be either a decimal group ID or a group name found in the group file.

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

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

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), chown(1), group(4), passwd(4).

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe; see the *S/Series CTIX Administrator's Guide* for more information.

NAME

chroot - change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

chroot causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the path names is changed for the command and any of its child processes to *newroot*. Furthermore, upon execution, the initial working directory is *newroot*.

Notice, however, that if you redirect the output of the command to a file:

```
chroot newroot command >x
```

will create the file x relative to the original root of the command, not the new one.

The new root path name is always relative to the current root: even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

This command can be run only by the super-user.

SEE ALSO

cd(1), chroot(2).

BUGS

One should exercise extreme caution when referencing device files in the new root file system.

NAME

chrtbl - generate character classification and conversion tables

SYNOPSIS

chrtbl [file]

DESCRIPTION

The *chrtbl* command creates a character classification table and an upper/lower-case conversion table. The tables are contained in a byte-sized array encoded such that a table lookup can be used to determine the character classification of a character or to convert a character [see *ctype*(3C)]. The size of the array is 257*2 bytes: 257 bytes are required for the 8-bit code set character classification table and 257 bytes for the upper- to lower-case and lower- to upper-case conversion table.

chrtbl reads the user-defined character classification and conversion information from *file* and creates two output files in the current directory. One output file, **ctype.c** (a C-language source file), contains the 257*2-byte array generated from processing the information from *file*. You should review the content of **ctype.c** to verify that the array is set up as you had planned. (In addiition, an application program could use **ctype.c**.) The first 257 bytes of the array in **ctype.c** are used for character classification. The characters used for initializing these bytes of the array represent character classifications that are defined in **/usr/include/ctype.h**; for example, **_L** means a character is lower case and **_S|_B** means the character is both a spacing character and a blank. The last 257 bytes of the array are used for character conversion. These bytes of the array are initialized so that characters for which you do not provide conversion information will be converted to themselves. When you do provide conversion information, the first value of the pair is stored where the second one would be stored normally, and vice versa; for example, if you provide **<0x41 0x61>**, then **0x61** is stored where **0x41** would be stored normally, and **0x61** is stored where **0x41** would be stored normally.

The second output file (a data file) contains the same information, but is structured for efficient use by the character classification and conversion routines [see *ctype*(3C)]. The name of this output file is the value of the character classification **chrclass** read in from *file*. This output file must be installed in the **/lib/chrclass** directory under this name by someone who is super-user or a member of group **bin**. This file must be readable by user, group, and other; no other permissions should be set. To use the character classification and conversion tables on this file, set the environmental variable **CHRCLASS** [see *environ*(5)] to the name of this file and export the variable; for example, if the name of this file (and character class) is **xyz**, you should issue the commands: **CHRCLASS=xyz ; export CHRCLASS .**

If no input file is given, or if the argument - is encountered, *chrtbl* reads from the standard input file.

The syntax of *file* allows the user to define the name of the data file created by *chrtbl*, the assignment of characters to character classifications and the relationship between upper- and lower-case letters. The following character classifications are recognized by *chrtbl*:

chrclass name of the data file to be created by *chrtbl*.

isupper character codes to be classified as upper-case letters.

islower character codes to be classified as lower-case letters.

isdigit character codes to be classified as numeric.

isspace character codes to be classified as a spacing (delimiter) character.

ispunct character codes to be classified as a punctuation character.

iscntrl character codes to be classified as a control character.

isblank character code for the space character.

isxdigit character codes to be classified as hexadecimal digits.

ul relationship between upper- and lower-case characters.

Any lines with the number sign (#) in the first column are treated as comments and are ignored. Blank lines are also ignored.

A character can be represented as a hexadecimal or octal constant (for example, the letter **a** can be represented as 0x61 in hexadecimal or 0141 in octal). Hexadecimal and octal constants may be separated by one or more space and tab characters.

The dash character (-) may be used to indicate a range of consecutive numbers. Zero or more space characters may be used for separating the dash character from the numbers.

The backslash character (\) is used for line continuation. Only a carriage return is permitted after the backslash character.

The relationship between upper- and lower-case letters (**ul**) is expressed as ordered pairs of octal or hexadecimal constants: *<upper-case_character lower-case_character>*. These two constants may be separated by one or more space characters. Zero or more space characters may be used for separating the angle brackets (< >) from the numbers.

EXAMPLE

The following is an example of an input file used to create the ASCII code set definition table on a file named ascii.

```
chrclass ascii
isupper  0x41 - 0x5a
islower  0x61 - 0x7a
isdigit  0x30 - 0x39
isspace  0x20 0x9 - 0xd
ispunct  0x21 - 0x2f          0x3a - 0x40          \
        0x5b - 0x60  0x7b - 0x7e
iscntrl  0x0 - 0x1f  0x7f
isblank  0x20
isxdigit 0x30 - 0x39          0x61 - 0x66          \
        0x41 - 0x46
ul       <0x41 0x61> <0x42 0x62> <0x43 0x63> \
        <0x44 0x64> <0x45 0x65> <0x46 0x66> \
        <0x47 0x67> <0x48 0x68> <0x49 0x69> \
        <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c> \
        <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f> \
        <0x50 0x70> <0x51 0x71> <0x52 0x72> \
        <0x53 0x73> <0x54 0x74> <0x55 0x75> \
        <0x56 0x76> <0x57 0x77> <0x58 0x78> \
        <0x59 0x79> <0x5a 0x7a>
```

FILES

/lib/chrclass/*	data file containing character classification and conversion tables created by *chrtbl*
/usr/include/ctype.h	header file containing information used by character classification and conversion routines

SEE ALSO

ctype(3C), environ(5).

DIAGNOSTICS

The error messages produced by *chrtbl* are intended to be self-explanatory. They indicate errors in the command line or syntactic errors encountered within the input file.

NAME

ckbupscd - check file system backup schedule

SYNOPSIS

/etc/ckbupscd [**-m**]

DESCRIPTION

ckbupscd consults the file **/etc/bupsched** and prints the file system lists from lines with date and time specifications matching the current time. If the **-m** flag is present an introductory message in the output is suppressed so that only the file system lists are printed. Entries in the **/etc/bupsched** file are printed under the control of *cron*.

The file **/etc/bupsched** should contain lines of 4 or more fields, separated by spaces or tabs. The first 3 fields (the schedule fields) specify a range of dates and times. The rest of the fields constitute a list of names of file systems to be printed if *ckbupscd* is run at some time within the range given by the schedule fields. The general format is:

time[,time] day[,day] month[,month] fsyslist

where:

time Specifies an hour of the day (*0* through *23*), matching any time within that hour, or an exact time of day (*0:00* through *23:59*).

day Specifies a day of the week (*sun* through *sat*) or day of the month (*1* through *31*).

month Specifies the month in which the time and day fields are valid. Legal values are the month numbers (*1* through *12*).

fsyslist The rest of the line is taken to be a file system list to print.

Multiple time, day, and month specifications may be separated by commas, in which case they are evaluated left to right.

An asterisk (*) always matches the current value for that field.

A line beginning with a sharp sign (#) is interpretted as a comment and ignored.

The longest line allowed (including continuations) is 1024 characters.

EXAMPLES

The following are examples of lines which could appear in the **/etc/bupsched** file.

```
06:00-09:00 fri 1,2,3,4,5,6,7,8,9,10,11 /applic
```

Prints the file system name */applic* if *ckbupscd* is run between 6:00am and 9:00am any Friday during any month except December.

```
00:00-06:00,16:00-23:59 1,2,3,4,5,6,7  1,8  /
```

Prints a reminder to backup the root (/) file system if *ckbupscd* is run between the times of 4:00pm and 6:00am during the first week of August or January.

FILES

/etc/bupsched	specification file containing times and file system to back up

SEE ALSO

cron(1M), echo(1), sh(1).
S/Series CTIX Administrator's Guide.

BUGS

ckbupscd will report file systems due for backup if invoked any time in the window. It does not know that backups may have just been taken.

NAME

clear - clear terminal screen

SYNOPSIS

clear

DESCRIPTION

clear prints the string that clears your terminal's screen. The program obtains this string from the *terminfo*(4) database, using the **TERM** environment variable to determine the type of terminal.

FILES

/usr/lib/terminfo/?/* terminal capability database

SEE ALSO

sh(1), terminfo(4).

NAME

clri - clear i-node

SYNOPSIS

/etc/clri special i-number ...

DESCRIPTION

clri writes nulls on the 64 bytes at offset *i-number* from the start of the i-node list. This effectively eliminates the i-node at that address. *Special* is the device name on which a file system has been defined. After *clri* is executed, any blocks in the affected file will show up as "not accounted for" when *fsck*(1M) is run against the file-system. The i-node may be allocated to a new file.

Read and write permission is required on the specified *special* device.

This command is used to remove a file which appears in no directory; that is, to get rid of a file which cannot be removed with the *rm* command.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M), fs(4).

WARNINGS

If the file is open for writing, *clri* will not work. The file system containing the file should NOT be mounted.

If *clri* is used on the i-node number of a file that does appear in a directory, it is imperative to remove the entry in the directory at once, since the i-node may be allocated to a new file. The old directory entry, if not removed, continues to point to the same file. This sounds like a link, but does not work like one. Removing the old entry destroys the new file.

NAME

cmp - compare two files

SYNOPSIS

cmp [**-l**] [**-s**] [**-an**] [**-o**] 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.

-a*n* Start the comparison at byte offset *n*, where *n* is an octal number. (Note that the byte offset will be the same for both files.)

-o Ignore time and date stamp differences when comparing the contents of binary files.

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

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

NAME

col - filter reverse line-feeds

SYNOPSIS

col [**-b**] [**-f**] [**-x**] [**-p**]

DESCRIPTION

The *col* command reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-7), 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.

If the **-b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

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.

Unless the **-x** option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters **SO** (\017) and **SI** (\016) are assumed by *col* to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output **SI** and **SO** characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, **SI**, **SO**, **VT** (\013), and **ESC** followed by **7**, **8**, or **9**. The **VT** character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, *col* will ignore any escape sequences unknown to it that are found in its input; the **-p** option may be used to cause *col* to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

SEE ALSO

nroff(1), tbl(1).

NOTES

The input format accepted by *col* matches the output produced by *nroff* with either the **-T37** or **-Tlp** options. Use **-T37** (and the **-f** option of *col*) if the ultimate disposition of the output of *col* will be a device that can interpret half-line motions, and **-Tlp** otherwise.

BUGS

Cannot back up more than 128 lines.
Allows at most 800 characters, including backspaces, on a line.
Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comb - combine SCCS deltas

SYNOPSIS

comb [**-o**] [**-s**] [**ipsid**] [**-clist**] files

DESCRIPTION

The *comb* command generates a shell procedure [see *sh*(1)] which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

-o For each **get -e** generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the **-o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

-s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original - combined) / original

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

-p*SID* The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

-c*list* A *list* [see *get*(1) for the syntax of a *list*] of deltas to be preserved. All other deltas are discarded.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB	The name of the reconstructed SCCS file.
comb?????	Temporary.

SEE ALSO

admin(1), delta(1), get(1), help(1), help(1), prs(1), sh(1), sccsfile(4).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

comm - select or reject lines common to two sorted files

SYNOPSIS

comm [- [**123**]] file1 file2

DESCRIPTION

comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence [see *sort*(1)], and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name - 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** prints nothing.

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1).

NAME

config - configure a CTIX system

SYNOPSIS

/etc/config [**-l** file] [**-c** file] [**-m** file] [**-t**] [**-b** num] [**-d** num] [**-s** num] [**-f** num] dfile

DESCRIPTION

The *config* program takes a description of a CTIX system, generates a configuration table file, and generates a hardware interface file. The configuration table file is a C program defining the configuration tables for the various devices on the system. The hardware interface file provides information about the interface between the hardware and device handlers.

The options to *config*(1M) are as follows:

-l Specifies the name of the hardware interface file; **low.s** is the default.

-c Specifies the name of the configuration table file; **conf.c** is the default.

-m Specifies the name of the file that contains information about supported devices; **/etc/master** is the default name. This file is supplied with the CTIX system and should *not* be modified unless the user *fully* understands its construction.

-t Requests a short table of major device numbers for character- and block-type devices. This can facilitate the creation of special files.

-b Specifies the minimum number of entries in the *bdevsw* array. The default value is **20**.

-d Specifies the minimum number of entries in the *cdevsw* array. The default value is **128**.

-s Specifies the minimum number of entries in the *fmodsw* array. The default value is **16**.

-f Specifies the minimum number of entries in the *fstypsw* array. The default value is **8**. The **-b**, **-d**, **-s**, and **-f** options are provided to ensure that a sufficient number of empty slots are available for loadable modules, such as drivers, stream modules, software modules, and file system types.

The user must supply *dfile*, which must contain device information for the user's system. The *dfile* is divided into two parts: the first part contains physical device specifications; the second part contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment. A sample *dfile* file is provided in the **/usr/sys/cf** directory.

First Part of dfile

Each line in the first part of the *dfile* contains one field, *devname*, which is the name of the device, software module, stream module or file system type (as it appears in the **/etc/master** device table).

The disk driver section is the first group of the first part; this first group must contain only disk driver *devname* entries.

Note that for disk controllers, the position of *devname* in *dfile* determines the **/dev/dsk** controller number assigned to the *devname* driver. For tape controllers, the position of *devname* has no significance. The tape controller number mapping is performed by the CTIX installation tools, using the **mknod**(1M) command.

The following example shows how controller numbers are assigned from an S/640 or S/480 *dfile*:

dfile Entry	Controller Number
diskonbd	**/dev/dsk/c0d? (ST506)**
Vsmd3200	**/dev/dsk/c1d? (1st SMD)**
scsidisk	**/dev/dsk/c2d? (SCSI)**
Vsmd3200	**/dev/dsk/c3d? (2nd SMD)**
:	
scsi	**(required line for SCSI)**
stape	**(SCSI QIT and HIT)**

In the example, above the Quarter-Inch Tape (QIT) drive is a SCSI device, assigned to **/dev/rmt/c0d?**. The VME-based Half-Inch Tape controller is assigned **/dev/rmt/c1d?** by the installation tools. The driver (**/etc/lddrv/ipt.o**) is loaded dynamically at boot time, so **ipt** is not entered in the *dfile*.

For the S/120, S/22*x*, and S/320, disk controller numbers are assigned as they are on the S/640 and S/480, but tape controller numbers are assigned differently, as shown below:

dfile Entry	Controller Number
:	
qic	**/dev/rmt/c0d0 (QIC-2)**
scsi	**(required line for SCSI)**
stape	**(SCSI QIT and HIT)**

As shown above, the first Quarter-Inch Tape drive is QIC-2 controller-based; it is assigned **/dev/rmt/c0d0**. Again, the VME-based Half-Inch Tape drive

controller is assigned **/dev/rmt/c1d***?* by the installation tools. The SCSI Quarter-Inch Tape and Half-Inch tape drives are assigned **/dev/rmt/c2d***?*.

For the S/80 and S/280, the controller number assignment is different, because the onboard disk controller is the SCSI controller and there is no VME expansion:

dfile Entry	Controller Number
scsidisk	**/dev/dsk/c0d? (SCSI)**
:	
scsi	**(required line for SCSI)**
stape	**(SCSI QIT and HIT)**

As shown above, the first Quarter-Inch Tape drive is assigned **/dev/rmt/c0d0**. Remaining tape drives are assigned **/dev/rmt/c0d***?* (where *?* is 1 through 7).

Second Part of dfile

The second part of the *dfile* contains four types of lines, listed and described below. Note that *all* specifications of this part *are required*, although the order is arbitrary.

1. *Root/pipe device specification*

   Two lines of three fields each:

   root devname minor
   pipe devname minor

   where *minor* is the minor device number (in decimal) of the slice on the fixed disk.

2. *Swap device specification*

   One line that contains five fields, as follows:

   swap devname minor swplo nswap

   where *swplo* is the lowest disk block (decimal) in the swap area and *nswap* is the maximum number of 1K-byte disk blocks (decimal) in the swap area. The kernel sizes the actual swap area size and configures itself for up to this maximum.

3. *Dynamic device number assignment*

   The *devnames* for **root**, **swap**, and **pipe** can be specified as **any**. The major device numbers are undetermined until boot time.

The key word parameter **dynamic** can be used with the **any** *devname* to force the major device number of the **boot** device to 0. For example, if the **devnames** in the *dfile* are **diskonbd** and **scsidisk**, and the **boot** device is the **scsidisk**, the kernel swaps the device numbers so that **scsidisk** can be accessed through major device number 0 instead of 1, as specified in the *dfile*. All access to SCSI disks is through **/dev/dsk/c0d***?***s***?*, and all access to ST506 disks is through **/dev/dsk/c1d***?***s***?*. If **any** is not used, **dynamic** has no effect.

4. *Parameter specification*

Any number of lines of two fields each, chosen from the following list. *Number* is decimal. Note that the following parameter list is *not* complete; parameters not on the list either must not be changed or have no effect.

```
buffers      number   /* number of 1024-byte file system
                         caching buffers */
buffers_4k number     /* number of 4096-byte file system
                         caching buffers */
dmmxsz       number   /* max number of pages per loadable
                         driver */
inodes       number   /* max open inodes in system */
s5inodes     number   /* max open s5inodes in system */
files        number   /* max open files in system */
flkrec       number   /* max locks active in system */
mounts       number   /* max file systems mounted */
regions      number   /* total number of regions in system */
procs        number   /* max processes in system */
maxproc      number   /* max processes per user ID */
maxfsiz      number   /* ulimit default in 512-byte blocks */
maxumem      number   /* max number of pages per process */
cbufsize     number   /* console circular buffer size in bytes */
msgmax       number   /* max chars in a message */
msgmni       number   /* max active message queues */
msgmnb       number   /* max total chars in message queues */
msgtql       number   /* max messages in system */
msgssz       number
msgseq       number   /* msgssz * msgseq = number bytes of
                         system buffering */
nlldrv       number   /* max number of loadable drivers */
semmni       number   /* max active semaphores */
semmns       number   /* max semaphores in system */
semmsl       number   /* max semaphores per ID */
```

```
semopm      number   /* max operations per semop call */
semume      number   /* max undo structures per process */
semmnu      number   /* max undo structures in system */
dirlosz     number   /* direct I/O default size */
shmmax      number   /* max bytes in a shared segment */
shmmin      number   /* min bytes in a shared segment */
shmmni      number   /* max active shared segments */
shmseg      number   /* max attached segments per process */
shmbrk      number   /* gap in pages between data and
                        shared memory */
nqueue      number   /* max stream queues */
nstream     number   /* max streams */
nblk4096    number   /* number of 4096 byte stream bufs */
nblk2048    number   /* number of 2048 byte stream bufs */
nblk1024    number   /* number of 1024 byte stream bufs */
nblk512     number   /* number of 512 byte stream bufs */
nblk256     number   /* number of 256 byte stream bufs */
nblk128     number   /* number of 128 byte stream bufs */
nblk64      number   /* number of 64 byte stream bufs */
nblk16      number   /* number of 16 byte stream bufs */
nblk4       number   /* number of 4 byte stream bufs */
shlbmax     number   /* max # of shared libs per process */
nofiles     number   /* max # of open files per process */
ntlmod      number   /* max # of TLI connections */
ntlrdwr     number   /* max # of TLI read/write connections */
nsp         number   /* max # of stream pipes */
```

Certain parameters, if set to 0, allow the kernel to autoconfigure. For example, **procs**, **regions**, **clists**, **i-nodes**, **s5inodes**, **files**, and **buffers** are autoconfigurable. The value of **procs** is based on the number of users; The values for **regions**, **i_nodes**, **s5inodes**, and **files** are based on the value of **procs**. The value of **clists** is based on the number of serial and cluster ports. The value of **buffers** is based on the amount of physical memory. Any or all of the autoconfigured values can be overridden. The value of **maxumem** can also be set to 0, in which case it floats between 1M byte and one quarter of the total swap space.

EXAMPLE

This example assumes an S/640 system with the following devices:

- Onboard ST506 disks (root)
- First Interphase SMD disk controller

- SCSI disks
- Second Interphase SMD disk controller
- RS-232-C (any number of ports)
- SCSI tape drives
- one parallel line printer
- root device is a disk (drive 0, section 1)
- pipe device is a disk (drive 0, section 1)
- swap device is a disk (drive 0, section 2), with a swplo of 1 and an nswap of 8000
- number of buffers is 100
- number of processes is 100
- maximum number of processes per user ID is 25
- number of mounts is 6
- number of inodes is 100
- number of files is 120
- number of character buffers is 64
- messages are to be included
- semaphores are to be included

The S/640 system configuration would be specified as follows in the *dfile*:

```
diskonbd
Vsmd3200
scsidisk
Vsmd3200
serial
scsi
stape
console
pip
root    any             01
pipe    any             01
swap    any             02      0       8000
* Comments are inserted in this manner
buffers 100
procs   100
maxproc 25
mounts  6
inodes  100
files   120
mesg    1
```

```
sema    1
clists  64
```

FILES

/etc/master	default input master device table
/usr/sys/cf/dfile	default system configuration
/usr/sys/cf/low.s	default output hardware interface
/usr/sys/cf/conf.c	default output configuration table

SEE ALSO

ldeeprom(1M), uconf(1M), master(4).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

Diagnostics are routed to the standard output and are self-explanatory.

BUGS

The **-t** option does not know about devices that have aliases.

NAME

conlocate - locate a terminal to use as the virtual system console

SYNOPSIS

/etc/conlocate [**-r**] [**-in**] [**-t**]

DESCRIPTION

The *conlocate* command searches for a terminal to use as the system console, **/dev/syscon**. *Conlocate* scans **/etc/inittab** for terminals that get a *getty*(1M) in state 6, and then spawns children to monitor the terminals for attempted logins. Each child performs the I/O control and login verification of the *getty-login* sequence, but only **root** can actually log in. The first terminal to have **root** log in gets its tty linked to **/dev/syscon**. The *conlocate* command then writes to its own standard output the new virtual system console's communication options, which are set from the values in **/etc/gettydefs**, by using the **stty -g** command.

The following are options to *conlocate*:

-r If **/dev/syscon** exists and can be opened, exit without scanning for a new system console terminal.

-i*n* Scan run level *n* instead of run level 6.

-t Begin by monitoring for logins on the existing **/dev/syscon**. If **root** logs in at that terminal within 20 seconds, abandon the search for another console.

FILES

/dev/syscon	virtual system console
/etc/inittab	definitions of operating states
/etc/gettydefs	communication options

SEE ALSO

init (1M), stty(1), gettydefs(4), inittab(4), termio(7).

WARNING

Beware of collision with other processes that might be trying to open the same terminals, especially *gettys* spawned by *init*.

NAME

conv - common object file converter

SYNOPSIS

conv [**-a**] [**-o**] [**-p**] **-t** target [- | files]

DESCRIPTION

The *conv* command converts object files in the common object file format from their current byte ordering to the byte ordering of the *target* machine. The converted file is written to *file.v*. The *conv* command can be used on either the source (sending) or target (receiving) machine.

Command line options follow:

- indicates that the names of *files* should be read from the standard input.

-a If the input file is an archive, produce the output file in the UNIX System V Release 2.0 portable archive format.

-o If the input file is an archive, produce the output file in the old (pre- UNIX System V) archive format.

-p If the input file is an archive, produce the output file in the UNIX System V Release 1.0 random access archive format.

-t target Convert the object file to the byte ordering of the machine (*target*) to which the object file is being shipped. This can be another host or a target machine. Legal values for *target* follow: **pdp**, **vax**, **ibm**, **x86**, **b16**, **n3b**, **mc68**, and **m32**.

The *conv* command is meant to ease the problems created by a multi-host cross-compilation development environment. The *conv* command is best used within a procedure for shipping object files from one machine to another.

The *conv* command recognizes and produces archive files in three formats: the pre- UNIX System V format, the UNIX System V Release 1.0 random access format, and the UNIX System V Release 2.0 portable ASCII format. By default, *conv* creates the output archive file in the same format as the input file. To produce an output file in a different format than the input file, use the **-a**, **-o**, or **-p** option. If the output archive format is the same as the input format, the archive symbol table is converted, otherwise the symbol table is stripped from the archive. The *ar*(1) command with its **-t** and **-s** options must be used on the target machine to recreate the archive symbol table.

EXAMPLE

To ship object files from a VAX to a S/MT Computer, execute the following commands:

```
conv -t mc68 *.out

uucp *.out.v myS320!~/rje/
```

DIAGNOSTICS

The diagnostics are self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics on an input file cause the program to continue to the next input file.

CAVEATS

The *conv* command does not convert archives from one format to another if both the source and target machines have the same byte ordering; use the CTIX system tool *convert*(1) instead.

SEE ALSO

ar(1), convert(1), ar(4), a.out(4).

NAME

convert - convert archive files to common formats

SYNOPSIS

convert infile outfile

DESCRIPTION

The *convert* command transforms input *infile* to output *outfile*. *Infile* must be a UNIX System V Release 1.0 archive file and *outfile* will be the equivalent UNIX System V Release 2.0 archive file. All other types of input to the *convert* command will be passed unmodified from the input file to the output file (along with appropriate warning messages).

Infile must be different from *outfile*.

FILES

TMPDIR/conv* temporary files

TMPDIR is usually /tmp but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*( ) in *tmpnam*(3S)].

SEE ALSO

ar(1), tmpnam(3S), a.out(4), ar(4)

NAME

cp, ln, mv - copy, link, or move files

SYNOPSIS

cp file1 [file2 ...] target

ln [**-f**] file1 [file2 ...] target

mv [**-f**] file1 [file2 ...] target

DESCRIPTION

These commands copy (link, move) *file1* to *target*. Under no circumstance can *file1* and *target* be the same [take care when using *sh*(1) metacharacters]. If *target* is a directory, one or more files are copied (linked, moved) to that directory. If *target* is a file, its contents are destroyed.

If *mv* or *ln* determines that the mode of *target* forbids writing, it prints the mode [see *chmod*(2)], asks for a response, and reads the standard input for one line; if the line begins with **y**, the *mv* or *ln* occurs, if permissible; if not, the command exits. For *mv*, when source parent directories or the target directory is writable and has the sticky bit set, any of the following conditions must be true:

the user must own the file
the user must own the directory
the file must be writable to the user
the user must be the super-user

When the **-f** option is used or if the standard input is not a terminal, no questions are asked and the *mv* or *ln* is done.

Only *mv* allows *file1* to be a directory, in which case the directory rename occurs only if the two directories have the same parent; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target becomes a new file.*

When using *cp*, if *target* is not a file, a new file is created with the same mode as *file1*, except that the sticky bit is not set unless you are super-user; the owner and group of *target* are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, nor group. The last modification time of *target* (and last access time, if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

SEE ALSO

chmod(1), cpio(1), rm(1).

WARNINGS

The *ln* command does not link across file systems. This restriction is necessary because file systems can be added and removed.

BUGS

If *file1* and *target* lie on different file systems, *mv* must copy the file and delete the original. In this case, any linking relationship with other files is lost.

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o[**acBQvV**] [**-C** *bufsize*] [[**-O** *file*] [**-M** *message*]]

cpio -i [**BQcdmrtuvVfsSb6k**] [**-C** *bufsize*] [[**-I** *file*] [**-M** *message*]] [*pattern* ...]

cpio -p [**adlmuvV**] directory

DESCRIPTION

The *cpio -o* (copy out) command reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary by default.

cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio -o**. The block size used with **cpio -i** must be the same as the block size used with the **cpio -o** when the archive was made. Only files with names that match *patterns* are selected. *Patterns* are regular expressions given in the name-generating notation of *sh*(1). In *patterns*, meta-characters **?**, *****, and [...] match the slash (/) character, and backslash (\) is an escape character. A **!** meta-character means *not*. (For example, the **!abc*** pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (that is, select all files). Each *pattern* must be enclosed in double quotation marks; otherwise, the name of a file in the current directory is used.

The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The files use the permissions of the previous **cpio -o**. The current user is the owner and group of the files unless the user is super-user, in which case the files use the the owner and group of the previous **cpio -o**.

Note that if **cpio -i** tries to create a file that already exists, and the existing file is the same age or newer, **cpio** displays a warning message and does not replace the file. (The **-u** option can be used to unconditionally overwrite the existing file.)

cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

a Reset *access* times of input files after they have been copied. Access times are not reset for linked files when **cpio -pla** is specified.

B Input/output is to be *blocked* 5,120 bytes to the record. The default buffer size is 512 bytes when this and the **C** or **Q** options are not used. (**-B** does not apply to the *pass* option; **-B** is meaningful only with data directed to or from a character special device, for example, **/dev/rmt0** or raw floppy disks).

Q Input/output is to be blocked 65,536 bytes to the record. Works like **-B** option, with which it is mutually exclusive. The **-Q** option optimizes quarter-inch tape access.

d *Directories* are to be created as needed.

c Write header information in ASCII *character* form for portability. Always use this option when origin and destination machines are different types.

C *bufsize* Input/output is to be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when the **Q**, and **B** options are not used. (**-C** does not apply to the *pass* option; **-C** is meaningful only with data directed to or from a character special device, for example, **/dev/rmt0** or raw floppy disks).

I *file* Read the contents of *file* as input. If *file* is a character special device, when the first medium is full replace the medium and type a carriage return to continue to the next medium. Use only with the **-i** option.

O *file* Direct the output of cpio to *file*. If *file* is a character special device, when the first medium is full replace the medium and type a carriage return to continue to the next medium. Use only with the **-o** option.

r Interactively *rename* files. If the user types a null line, the file is skipped. If the user types a "." the original pathname is copied. (Not available with **cpio -p**.)

t Print a *table of contents* of the input. No files are created.

u Copy *unconditionally* (normally, an older file does not replace a newer file with the same name).

v *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls -l** command [see *ls*(1)].

V *Special* Verbose: print a dot for each file seen. Useful to reassure the user that **cpio** is working without printing out all file names.

k Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For *cpio* archives that contain other *cpio* archives, if an error is encountered *cpio* can terminate prematurely. *cpio* finds the next good header, which can be one for a smaller archive, and terminates when the smaller archive's trailer is encountered.) Used only with the **-i** option.

l Whenever possible, *link* files rather than copying them. Usable only with the **-p** option.

m Retain previous file *modification* time. This option is ineffective on directories that are being copied.

M *message* Define a message to use when switching media. When you use the **-O** or **-I** options and specify a character special device, you can use this option to define the message that is printed when you reach the end of the medium. One **%d** can be placed in the message to print the sequence number of the next medium needed to continue.

f Copy in all *files* except those in *patterns*. (See the paragraph on **cpio -i** for a description of *patterns*.)

s *Swap* bytes within each half word. Use only with the **-i** option.

S *Swap* halfwords within each word. Use only with the **-i** option.

b Reverses the order of the *bytes* within each word. Use only with the **-i** option.

6 Process an old (that is, UNIX System *Sixth* Edition format) file. Only useful with -i (copy in).

Note that **cpio** assumes four-byte words.

If **cpio** reaches end of medium (end of a diskette for example), when writing to (**-o**) or reading from (**-i**) a character special device, **cpio** prints the following message:

Reached end of medium on output.
To exit - press <E> followed by <RETURN>.
To continue - insert volume #nn and press the <RETURN> key.

To continue, replace the medium and press Return.

EXAMPLES

The following examples show three uses of **cpio**.

When standard input is directed through a pipe to **cpio -o**, it groups the files so they can be directed (>) to a single file (../newfile). Instead of "ls," you could use find, echo, cat, etc. to pipe a list of names to cpio. You could direct the output to a device instead of a file.

ls | cpio -oc >.*/newfile*

cpio -i uses the output file of **cpio -o** (directed through a pipe with cat in the example), takes out those files that match the patterns (memo/a1, memo/b*), creates directories below the current directory as needed (-d option), and places the files in the appropriate directories. If no patterns were given, all files from **newfile** would be placed in the directory.

cat newfile | cpio -icd *"memo/a1" "memo/b*"*

cpio -p takes the file names piped to it and copies or links (-l option) those files to another directory on your machine (*newdir* in the example). The -d options says to create directories as needed. The -m option says retain the modification time. (It is important to use the -depth option of find to generate path names for cpio. This eliminates problems **cpio** could have trying to create files under read-only directories.)

find . -depth -print | cpio -pdlmv *newdir*

SEE ALSO

ar(1), find(1), ls(1), tar(1), cpio(4).

NOTES

1. Path names are restricted to 256 characters.
2. Only the super-user can copy special files.
3. Blocks are reported in 512-byte quantities.
4. If a file has 000 permissions, contains more than 0 characters of data, and the user is not **root**, the file is not saved or restored.

NAME

cpp - the C language preprocessor

SYNOPSIS

LIBDIR/cpp [option ...] [ifile [ofile]]

DESCRIPTION

The C language preprocessor, *cpp*, is invoked as the first pass of any C compilation by the *cc*(1) command. Thus *cpp*'s output is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than through the *cc*(1) command is not suggested, since the functionality of *cpp* may someday be moved elsewhere. See *m4*(1) for a general macro processor.

cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

-P Preprocess the input without producing the line control information used by the next pass of the C compiler.

-C By default, *cpp* strips C-style comments. If the **-C** option is specified, all comments (except those found on *cpp* directive lines) are passed along.

-U*name* Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. Following is the current list of these possibly reserved symbols.

operating system:	unix, gcos, ibm, os, tss
hardware:	interdata, mc68k, mc68000, mc68010, mc68020, pdp11, u370, u3b, u3b5, u3b2, u3b20d, vax
UNIX system variant:	RES, RT
lint(1):	lint

-D*name*

-D*name*=*def*

Define *name* with value *def* as if by a **#define**. If no =*def* is given, *name* is defined with value 1. The **-D** option has lower precedence than the **-U** option. That is, if the same name is used in both a **-U** option and a **-D** option, the name will be undefined regardless of the order of the options.

-T The **-T** option forces *cpp* to use only the first eight characters to distinguish preprocessor symbols and is included for backward compatibility.

-I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in " " will be searched for first in the directory of the file with the **#include** line, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the file with the **#include** line is not searched. By default, *cpp* searches for the name enclosed in < > in **/usr/include**; however, if the shell variable INCROOT is set, *cpp* prepends the value of INCROOT to the standard list. This is particularly useful for cross-compilation.

-Y*dir* Use directory *dir* in place of the standard list of directories when searching for **#include** files. Use of the **-Y** option overrides the value for INCROOT if it is set.

-H Print, one per line on standard error, the path names of included files.

Two special names are understood by *cpp*. The name **__LINE__** is defined as the current line number (as a decimal integer) as known by *cpp*, and **__FILE__** is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directive lines start with # in column 1. Any number of blanks and tabs is allowed between the # and the directive. The directives are:

#define *name token-string*

Replace subsequent instances of *name* with *token-string*.

#define *name*(*arg*, ..., *arg*) *token-string*

Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma-separated sets of tokens, and a) followed by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#undef *name*

Cause the definition of *name* (if any) to be forgotten from now on. No additional tokens are permitted on the directive line after *name*.

#ident *"string"*

Put *string* into the .comment section of an object file.

#include *"filename"*

#include *<filename>*

Include at this point the contents of *filename* (which will then be run through *cpp*). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See the **-I** and **-Y** options above for more detail. No additional tokens are permitted on the directive line after the final " or >.

#line *integer-constant "filename"*

Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file from which it comes. If *"filename"* is not given, the current file name is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

#endif

Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. No additional tokens are permitted on the directive line.

#ifdef *name*

The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**. No additional tokens are permitted on the directive line after *name*.

#ifndef *name*

The lines following will appear in the output if and only if *name* has not been the subject of a previous **#define**. No additional tokens are permitted on the directive line after *name*.

#if *constant-expression*

Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary –, **!**, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined**

(*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

To test whether either of two symbols, *foo* and *fum*, are defined, use

```
#if defined(foo) || defined(fum)
```

#elif *constant-expression*

An arbitrary number of **#elif** directives is allowed between a **#if**, **#ifdef**, or **#ifndef** directive and a **#else** or **#endif** directive. The lines following the **#elif** directive will appear in the output if and only if the preceding test directive evaluates to zero, all intervening **#elif** directives evaluate to zero, and the *constant-expression* evaluates to non-zero. If *constant-expression* evaluates to non-zero, all succeeding **#elif** and **#else** directives will be ignored. Any *constant-expression* allowed in a **#if** directive is allowed in a **#elif** directive.

#else The lines following will appear in the output if and only if the preceding test directive evaluates to zero, and all intervening **#elif** directives evaluate to zero. No additional tokens are permitted on the directive line.

The test directives and the possible **#else** directives can be nested.

FILES

INCDIR	standard directory list for **#include** files, usually /usr/include
LIBDIR	usually /lib

SEE ALSO

cc(1), lint(1), m4(1).

DIAGNOSTICS

The error messages produced by *cpp* are intended to be self-explanatory. The line number and file name where the error occurred are printed along with the diagnostic.

NOTES

The unsupported **-W** option enables the **#class** directive. If it encounters a **#class** directive, *cpp* will exit with code 27 after finishing all other processing. This option provides support for "C with classes".

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

```
#include <file.h>
```

should be used, rather than one with an absolute path, like:

```
#include "/usr/include/file.h"
```

cpp warns about the use of the absolute pathname.

NAME

cprs - compress a common object file

SYNOPSIS

cprs [**-p**] file1 file2

DESCRIPTION

The *cprs* command reduces the size of a common object file, *file1*, by removing duplicate structure and union descriptors. The reduced file, *file2*, is produced as output.

The sole option to *cprs* is:

-p Print statistical messages including: total number of tags, total duplicate tags, and total reduction of *file1*.

SEE ALSO

strip(1), a.out(4), syms(4).

NAME

cpset - install object files in binary directories

SYNOPSIS

cpset [**-o**] object directory [mode owner group]

DESCRIPTION

The *cpset* command is used to install the specified *object* file in the given *directory*. The *mode*, *owner*, and *group*, of the destination file may be specified on the command line. If this data is omitted, two results are possible:

If the user of *cpset* has administrative permissions (that is, the user's numerical ID is less than 100), the following defaults are provided:

mode - 0755

owner - bin

group - bin

If the user is not an administrator, the default, owner, and group of the destination file is that of the invoker.

An optional argument of **-o** forces *cpset* to move *object* to OLD*object* in the destination directory before installing the new object.

For example:

cpset echo /bin 0755 bin bin

cpset echo /bin

cpset echo /bin/echo

All the examples above have the same effect (assuming the user is an administrator). The file **echo** is copied into **/bin** and is given **0755, bin, bin** as the mode, owner, and group, respectively.

cpset utilizes the file **/usr/src/destinations** to determine the final destination of a file. The locations file contains pairs of pathnames separated by spaces or tabs. The first name is the "official" destination (for example: **/bin/echo**). The second name is the new destination. For example, if *echo* is moved from **/bin** to **/usr/bin**, the entry in **/usr/src/destinations** would be:

/bin/echo /usr/bin/echo

When the actual installation happens, *cpset* verifies that the *old* pathname does not exist. If a file exists at that location, *cpset* issues a warning and continues. This file does not exist on a distribution tape; it is used by sites to track local command movement. The procedures used to build the source are responsible for defining the *official* locations of the source.

Cross Generation

The environment variable **ROOT** is used to locate the destination file (in the form **$ROOT/usr/src/destinations**). This is necessary in the cases where cross generation is being done on a production system.

SEE ALSO

install(1M), make(1).

NAME

crash - examine system images

SYNOPSIS

/etc/crash [**-d** dumpfile] [**-n** namelist] [**-w** outputfile]

DESCRIPTION

The *crash* command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to *crash* are *dumpfile*, *namelist*, and *outputfile*.

Dumpfile is the file containing the system memory image. The default *dumpfile* is **/dev/kmem**. The system image can also be slice zero of the raw disk that contains the dump area (for example, **/dev/rdsk/c0d0s0**); or it can be the pathname of a file produced using *dd* to copy slice zero or just the dump area; or in the case of a tape dump, the second file on the tape.

The text file *namelist* contains the symbol table information needed for symbolic access to the system memory image to be examined. The default *namelist* is **/etc/lddrv/unix.exec** if examining a running system or **/etc/lddrv/prev.unix.exec** if examining a dump. If neither of these files exists, the default is **/unix**. If a system image from another machine is to be examined, the corresponding **prev.unix.exec** must be copied from that machine. The **prev.unix.exec** is preferred to **/unix** because it also contains the *namelist* for all the loaded drivers at the correct addresses.

When the *crash* command is invoked, a session is initiated. The output from a *crash* session is directed to *outputfile*. The default *outputfile* is the standard output.

Input during a *crash* session is of the form:

function [argument ...]

where *function* is one of the *crash* functions described in the FUNCTIONS section of this manual page, and *arguments* are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to *crash* functions wherever they are semantically valid:

-e	Display every entry in a table.
-f	Display the full structure.
-p	Interpret all address arguments in the command line as *physical* addresses.
-s process	Specify a process slot other than the default.
-w file	Redirect the output of a function to *file*.

Note that if the **-p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions *mode*, *defproc*, and *redirect* correspond to the function options **-p**, **-s**, and **-w**. The *mode* function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; *defproc* sets the value of the process slot argument for subsequent functions; and *redirect* redirects all subsequent output.

Output from *crash* functions may be piped to another program in the following way:

function [argument ...] ! shell_command

For example,

mount ! grep rw

will write all mount table entries with an *rw* flag to the standard output. The redirection option (**-w**) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table address arguments larger than the size of the function table will be interpreted as hexadecimal addresses; those smaller will be assumed to be decimal slots in the table. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by *0x* and as octal if it is preceded by *0*. Decimal override is designated by *0d*, and binary by *0b*.

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as *p* for *proc*, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number, a physical

address, a virtual address, a symbol, a range, or an expression. A range of slot numbers may be specified in the form *a-b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, -, *, /, &, or |. An operand which is a number should be preceded by a radix prefix if it is not a decimal number (*0* for octal, *0x* for hexidecimal, *0b* for binary). The expression must be enclosed in parentheses ( ). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to *crash* functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = table entry | address | symbol | range | expression

start_addr = address | symbol | expression

FUNCTIONS

? [**-w** file] List available functions.

!cmd Escape to the shell to execute a command.

adv [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print the advertise table.

base [**-w** file] number ...
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other then decimal should be preceded by a prefix that indicates its radix as follows: *0x*, hexidecimal; *0*, octal; and *0b*, binary.

buffer [**-w** file] [-format] bufferslot

or

buffer [**-w** file] [-format] [**-p**] start_addr
Alias: **b**.
Print the contents of a buffer in the designated format. The following format designations are recognized: **-b**, byte: **-c**, character; **-d**, decimal; **-x**, hexadecimal; **-o**, octal; **-r**, directory; and **-i**, inode. If no format is given, the previous format is used. The default format at the beginning of a *crash* session is hexadecimal.

bufhdr [**-f**] [**-w** file] [[**-p**] table_entry ...]
Alias: **buf**.
Print system buffer headers.
The **-f** option produces different output depending on whether the buffer is local or remote (contains RFS data).

callout [**-w** file]
Alias: **c**.
Print the callout table.

cblk [**-e**] [**-p**] [**-w** file] [**-t** type] [table_entry ...]
Display contents of cblocks.

clist [**-e**] [**-p**] [**-w** file] [**-t** type] [table_entry ...]
Display usage of clists.

conbuf [**-w** file]
Display console buffer.

dballoc [**-w** file] [class ...]
Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed.

dbfree [**-w** file] [class ...]
Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [**-e**] [**-w** file] [**-c** class ...]

or

dblock [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print allocated streams data block headers. If the class option (**-c**) is used, only data block headers for the class specified will be printed.

defproc [**-w** file] [**-c**]

or

defproc [**-w** file] [slot]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (**-c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a *crash* session, the process slot is set to the current process.

dis [**-w** file] [**-a**] start_addr [count]
Disassemble from the start address for *count* instructions. The default count is 1. The absolute option (**-a**) specifies a non-symbolic disassembly.

disk [**-w** file]
Display disk information.

ds [**-w** file] virtual_address ...
Print the data symbol whose address is closest to, but not greater than, the address entered.

fcallout [**-w** file]
Alias: **fc**.
Print the fast callout table.

file [**-e**] [**-w** file] [[**-p**] table_entry ...]
Alias: **f**.
Print the file table.

findaddr [**-w** file] table slot
Print the address of *slot* in *table*. Only tables available to the *size* function are available to *findaddr*.

findslot [**-w** file] virtual_address ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the *size* function are available to *findslot*.

fs [**-w** file] [[**-p**] table_entry ...]
Print the file system information table.

gdp [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the gift descriptor protocol table.

gt Equivalent to

tty -t gt

(See **tty** function below.)

help [**-w** file] function ...
Print a description of the named function, including syntax and aliases.

inode [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Alias: **i**.
Print the inode table, including file system switch information.

kfp [**-w** file] [**-s** process] [**-r**]

or

kfp [**-w** file] [**-s** process] [value]
Print the frame pointer for the start of a kernel stack trace. The kfp value can be set using the value argument or the reset option (**-r**), which sets the kfp from the saved kfp in a dump. If no argument is entered, the current value of the kfp is printed.

lck [**-e**] [**-w** file] [[**-p**] table_entry ...]
Alias: **l**.
Print record locking information. If the **-e** option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

linkblk [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print the linkblk table.

major [**-w** file] [entry ...]
Print the MAJOR table.

map [**-w** file] mapname ...
Print the map structure of the given mapname.

mbfree [**-w** file]
Print free streams message block headers.

mblock [**-e**] [**-w** filename] [[**-p**] table_entry ...]
Print allocated streams message block headers.

mode [**-w** file] [mode]
Set address translation of arguments to virtual (**v**) or physical (**p**) mode. If no mode argument is given, the current mode is printed. At the start of a *crash* session, the mode is virtual.

mount [**-e**] [**-w** file] [[**-p**] table_entry ...]
Alias: **m**.
Print the mount table.

msg [**-e**] [**-f**] [**-p**] [**-w** file] [**-s** process]
[table_entry ...]
Display IPC message queue headers.

msginfo [**-p**] [**-w** file]
Display IPC message information.

msgtext [**-e**] [**-p**] [**-w** file] [**-s** process]
[table_entry ...]
Display IPC message data.

nm [**-w** file] symbol ...
Print value and type for the given symbol.

notify [**-e**] [**-p**] [**-w** file] symbols

od [**-p**] [**-w** file] [-format] [-mode] [**-s** process]
start_addr [count]
Alias: **rd**.

Print *count* values starting at the start address in one of the following formats: character (**-c**), decimal (**-d**), hexadecimal (**-x**), octal (**-o**), ASCII (**-a**), or hexadecimal/character (**-h**), and one of the following modes: long (**-l**), short (**-t**), or byte (**-b**). The default mode for character and ASCII formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format **-h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a *crash* session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

pdt [**-e**] [**-w** file] [**-s** process] section segment

or

pdt [**-e**] [**-w** file] [**-s** process] [**-p**] start_addr [count]

S/640 Only:

The page descriptor table of the designated memory *section* and *segment* is printed. Alternatively, the page descriptor table starting at the start address for *count* entries is printed. If no count is entered, 1 is assumed.

pfdat [**-e**] [**-w** file] [[**-p**] table_entry ...]

Print the pfdata table.

pfree [**-e**] [**-p**] [**-w** file] table_entry ...

Display free list entries.

phash [**-e**] [**-p**] [**-w** file]

Display page hash table.

proc [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ... #procid ...]

or

proc [**-f**] [**-w** file] [**-r**]

Alias: **p**.

Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a #. Alternatively, process table information for runnable processes may be specified with the runnable option (**-r**).

pt Equivalent to

tty -t pt

(See **tty** function below.)

qrun [**-w** file]
Print the list of scheduled streams queues.

queue [**-e**] [**-w** file] [[**-p**] table_entry ...]
Print streams queues.

quit Alias: **q**.
Terminate the *crash* session.

rcvd [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the receive descriptor table.

redirect [**-w** file] [**-c**]

or

redirect [**-w** file] [file]
Used with a file name, redirects output of a *crash* session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (**-c**) closes the previously set file and redirects output to the standard output.

region [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the region table.

scsi [**-w** file]
Display SCSI tables.

scsirqb [**-f**] [**-w** file] [tbl_entry | start_addr]
Display SCSI request blocks.

sdt [**-e**] [**-w** file] [**-s** process] section

or

sdt [**-e**] [**-w** file] [**-s** process] [**-p**] start_addr [count]
S/640 Only:
The segment descriptor table for the named memory section is printed. Alternatively, the segment descriptor table starting at start address for *count* entries is printed. If no count is given, a count of 1 is assumed.

search [**-p**] [**-w** file] [**-m** mask] [**-s** process] pattern start_addr length

Print the words in memory that match *pattern*, beginning at the start address for *length* words. The mask is anded (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

ser Equivalent to

tty -t ser

(See **tty** function below.)

shm [**-e**] [**-f**] [**-p**] [**-w** file] table_entry ...

Display IPC shared memory headers.

shminfo [**-p**] [**-w** file]

Display system IPC shared memory information.

size [**-w** file] [**-x**] [structure_name ...]

Print the size of the designated structure. The (**-x**) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]

Print the send descriptor table.

sptb [**-e**] [**-p**] [**-w** file] [start_addr]

Display sptballoc maps.

srmount [**-e**] [**-w** file] [[**-p**] table_entry ...]

Print the server mount table.

stack [**-w** file] [**-u**] [process]

or

stack [**-w** file] [**-k**] [process]

or

stack [**-w** file] [[**-p**] **-i** start_addr]

Alias: **s**.

Dump stack. The (**-u**) option prints the user stack. The (**-k**) option prints the kernel stack. The (**-i**) option prints the interrupt stack starting at the start address. If no arguments are entered, the kernel stack for the current process is printed. The interrupt stack and the stack for the current process are not available on a running system.

stat [**-w** file]
Print system statistics.

stream [**-e**] [**-f**] [**-w** file] [[**-p**] table_entry ...]
Print the streams table.

strstat [**-w** file]
Print streams statistics.

swap Display swap map statistics.

swapinfo
Display swap statistics.

trace [**-w** file] [**-r**] [process]

or

trace [**-w** file] [[**-p**] **-i** start_addr]
Alias: **t**.
Print stack trace. The kfp value is used with the **-r** option. The interrupt option prints a trace of the interrupt stack beginning at the start address. The interrupt stack trace and the stack trace for the current process are not available on a running system.

ts [**-w** file] virtual_address ...
Print closest text symbol to the designated address.

tty [**-e**] [**-f**] [**-w** file] [**-t** type [[**-p**] table_entry ...]]

or

tty [**-e**] [**-f**] [**-w** file] [[**-p**] start_addr]
Valid types: **ser, pt, gt, vt.**
Print the tty table. If no arguments are given, the tty table for all tty types is printed. If the **-t** option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified from the start address.

unnotify [**-e**] [**-p**] [**-w** file] [**-s** process] symbols
Display queued notifications for process.

user [**-f**] [**-w** file] [process]
Alias: **u**.
Print the ublock for the designated process.

var [**-w** file]
Alias: **v**.
Print the tunable system parameters.

vt Equivalent to

tty -t vt

(See **tty** function above.)

vtop [**-w** file] [**-s** process] start_addr ...
Print the physical address translation of the virtual start address.

FILES

/dev/kmem	system image of currently running system
/dev/rdsk/c?d?s0	used to access system image on disk

NAME

createdev - create device nodes for assorted device types

SYNOPSIS

createdev [**-d** device] [**-c** controller] [**-v**] [**-r**] [**-p**] [**-t**]

DESCRIPTION

The *createdev* command is used to create device nodes of various types. After parsing various parameters, the command invokes *mknod* to create the specified device node or sets of device nodes.

The **-d** option specifies the device number (for example, unit, drive, or line number), and is required for every invocation.

The **-c** option specifies the controller number of the specified device.

The **-v** option specifies that disk devices are to be created. Both block-type and character-type device nodes of the form **/dev/rdsk/cxdysz** and **/dev/dsk/cxdysz** are added. Each invocation creates as many slices as the disk supports on CTIX. (currently 16). For the **-v** option, both the **-d** and **-c** options are required.

The **-r** option specifies that devices are created to provide access to streaming tape drives; these are the character-type device nodes. These devices are of the form: **/dev/rmt/cxdy**, **/dev/rmt/cxdyc**, **/dev/rmt/cxdyh**, **/dev/rmt/cxdyhn**, **/dev/rmt/cxdyl**, **/dev/rmt/cxdyln**, **/dev/rmt/cxdym**, **/dev/rmt/cxdymn**, and **/dev/rmt/cxdyn**.

This option is useful when adding SCSI tape devices. For the **-r** option, both the **-d** and **-c** options are required.

The **-t** option allows devices to be created of the type **/dev/tty***xxx*. These are character-type devices, typically used for terminals, line printers, as well as other perpherals. For this option, the **-d** option is required.

The **-p** option allows devices to be created of the type **/dev/ttyp***xx*. These are character-type devices, typically used for virtual login sessions. An example of this is an ethernet connection. The **-d** option is required.

FILES

/dev/tty*
/dev/ttyp*
/dev/dsk/*
/dev/rdsk/*
/dev/rmt/*

SEE ALSO

mknod(1M).

NAME

cron - clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in *crontab* files in the directory **/usr/spool/cron/crontabs**. Users can submit their own *crontab* file via the *crontab*(1) command. Commands which are to be executed only once may be submitted via the *at*(1) command.

cron only examines *crontab* files and *at* command files during process initialization and when a file changes via *crontab* or *at*. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since *cron* never exits, it should be executed only once. This is done routinely through **/etc/rc2.d/S75cron** at system boot time. **/usr/lib/cron/FIFO** is used as a lock file to prevent the execution of more than one *cron*.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/FIFO	used as a lock file
/usr/lib/cron/log	accounting information
/usr/spool/cron	spool area

SEE ALSO

at(1), crontab(1), sh(1).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

A history of all actions taken by *cron* are recorded in **/usr/lib/cron/log.**

NAME

crontab - user crontab file

SYNOPSIS

crontab [file]

crontab -r

crontab -l

DESCRIPTION

The *crontab* command copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The -r option removes a user's crontab from the crontab directory; the -l option lists the crontab file for the invoking user.

Users are permitted to use *crontab* if their names appear in the file **/usr/lib/cron/cron.allow.** If that file does not exist, the file **/usr/lib/cron/cron.deny** is checked to determine if the user should be denied access to *crontab*. If neither file exists, only root is allowed to submit a job. If **cron.deny** exists and is empty, global usage is permitted. If **cron.allow** exists and is empty, no usage is permitted. If **cron.allow** exists, **cron.deny** is ignored. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0-59),
hour (0-23),
day of the month (1-31),
month of the year (1-12),
day of the week (0-6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) 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.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh.** Users who desire to have their *.profile* executed must explicitly do so in the crontab file. *Cron* supplies a default environment for every shell, defining **HOME, LOGNAME, SHELL(=/bin/sh),** and **PATH(=:/bin:/usr/bin:/usr/local/bin).**

If you do not redirect the standard output and standard error of your commands, any generated output or errors are mailed to you.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

cron(1M), sh(1).
S/Series CTIX Administrator's Guide.

BUGS

If you inadvertently enter the **crontab** command with no argument(s), do not attempt to get out with a CTRL-d. This causes all entries in your **crontab** file to be removed. Instead, exit with a DEL.

NAME

crypt - encode/decode

SYNOPSIS

crypt [password]

crypt [-k]

DESCRIPTION

The *crypt* command reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no argument is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. If the **-k** option is used, *crypt* will use the key assigned to the environment variable CRYPTKEY. *crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

Files encrypted by *crypt* are compatible with those treated by the editors *ed*(1), *edit*(1), *ex*(1), and *vi*(1) 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 clear text 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, that is, 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.

If the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. The choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), edit(1), ex(1), makekey(1), ps(1), stty(1), vi(1).

WARNING

The standard CTIX distribution is the international version, which does not support encryption.

If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

BUGS

If output is piped to *nroff* and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state [see *stty*(1)].

NAME

csh - a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**-cefinstvVxX**] [arg ...]

DESCRIPTION

The *csh* interpreter is a first implementation of a command language interpreter incorporating a history mechanism (see "History substitutions"), job control facilities (see "Jobs"), and a C-like syntax.

An instance of *csh* begins by executing commands from the file **.cshrc** in the *home* directory of the invoker. If this is a login shell, then it also executes **/etc/cprofile** and commands from the file **.login** there. It is typical for users on terminals to put *tset*(1) in their **.login** files.

In the normal case, the shell then begins reading commands from the terminal, prompting with a percent sign (%). Processing of arguments and the use of the shell to process files containing command scripts is described later.

The shell repeatedly performs the following actions:

- A line of command input is read and broken into *words*.
- This sequence of words is placed on the command history list and then parsed.
- Finally each command in the current line is executed.

When a login shell terminates, it executes commands from the file **.logout** in the user's home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters & | ; < > (form separate words. If doubled in &&, | |, << or >> these pairs form single words. These parser metacharacters can be made part of other words, or prevented their special meaning, by preceding them with \. A newline preceded by \ is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, single or double, form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. The semantics of quotations are described later. Within pairs of quotes, a newline preceded by a \ gives a true newline character.

When the shell's input is not a terminal, the character # introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by \ and within double or single quotes.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by the pipe (|) character forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines separated by a semicolon (;), are executed sequentially. A sequence of pipelines can be executed in the backgrouind (so you don't have to wait for it to finish before giving another command); follow the command with an ampersand (&).

Any of the above can be placed in parentheses, (), to form a simple command (which can be a component of a pipeline or sequence). Separate pipelines with || or && to indicate, as in the C language, that the second is to be executed only if the first succeeds or fails, respectively (see "Expressions").

Jobs

The shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the *jobs* command, and assigns them small integer numbers. When a job is started in the background by using &, the shell prints a line that looks like that shown below to indicate that the job was job number 1 and had one (top-level) process, whose process ID was 1234.

```
[1] 1234
```

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a plus sign (+) and the previous job with a minus sign (-).

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, the shell notifies you immediately of changes of status in background jobs. There is also a shell command *notify*, which marks a single process so that its status changes are immediately reported. By default *notify* marks the current process. To mark a specific background job, type **notify** after starting the job.

Substitutions

The following paragraphs describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the exclamation point character (!) and can begin *anywhere* in the input stream (provided that they *do not* nest). The ! character can be preceded by \ to prevent its special meaning; for convenience, a ! is passed unchanged when it is followed by a blank, tab, newline, = or (. (History substitutions also occur when an input line begins with ↑. This special abbreviation is described later.) Any input line that contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal that consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of which is controlled by the *history* variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

Consider the following output from the *history* command:

```
 9 write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing ! in the prompt string.

With the current event 13 we can refer to previous events by event number **!11**, relatively as in **!-2** (referring to the same event), by a prefix of a command word as in **!d** for event 12 or **!wri** for event 9, or by a string contained in a word in the command as in **!?mic?** also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case **!!** refers to the previous command; thus **!!** alone is essentially a *redo*.

To select words from an event, follow the event specification with a colon (:) and a designator for the desired words. The words of an input line are

numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, and so on.

The basic word designators follow:

0	first (command) word
n	*n*th argument
↑	first argument: that is, 1
$	last argument
%	word matched by (immediately preceding) ?*s*? search
x-*y*	range of words
-*y*	abbreviates **0**-*y*
*	abbreviates ↑-**$**, or nothing if only one word in event
*x**	abbreviates *x*-**$**
x-	like *x** but omitting word '$'

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ↑, $, *, -, or %. A sequence of modifiers, each preceded by :, can be placed after the optional word designator; the modifiers are defined as follows:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing .xxx component, leaving the root name.
e	Remove all but the extension .xxx part.
s/*l*/*r*/	Substitute *l* for *r*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above (&): for example, **g&**.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like **q**, but break into words at blanks, tabs and newlines.

Unless preceded by a **g**, the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character can be used as the delimiter in place of /; a \ quotes the delimiter into the *l* " " and *r* " " strings. The & character in the right hand side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from an *l* or from a contextual scan string *s* " " in !?*s*?. The trailing delimiter in the substitution can be omitted if a newline follows immediately, as can the trailing ? in a contextual scan.

A history reference can be given without an event specification: for example, **!$**. In this case, the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus **!?foo?↑ !$** gives the first and last arguments from the command matching **?foo?**.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is ↑; this is equivalent to **!:s↑**, providing a convenient shorthand for substitutions on the text of the previous line. Thus **↑lb↑lib** fixes the spelling of **lib** in the previous command. Finally, a history substitution can be surrounded with curly braces, { and }, if necessary, to insulate it from the characters which follow. Thus, after **ls -ld ~paul** you can use **!{l}a** to mean **ls**-ld ~paula, while **!la** would look for a command starting **la**.

Quotations with ´ and "

The quotation of strings by ´ and **can be used** to prevent all or some of the remaining substitutions. Strings enclosed in single quotes, ´, are prevented any further interpretation. Strings enclosed in double quotes, can be expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see "Command substitition" below) does a double quoted (**single quoted strings** (´ ´) never do.

Alias substitution

The shell maintains a list of aliases that can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, the text that is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, the argument list is left unchanged.

Thus if the alias for **ls** is **ls -l**, the command **ls /usr** maps to **ls -l /usr**, the argument list here being undisturbed. Similarly if the alias for **lookup** is **grep !↑ /etc/passwd**, then **lookup bill** maps to **grep bill /etc/passwd**.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Aliases can introduce parser metasyntax; so the syntax **alias print 'pr \!* | lpr'** creates a command that *pr's* its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of the variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables can be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell, a number are toggles; the shell does not use thir values, but needs to know whether they are set or not. For instance, the *verbose* variable is a toggle that causes command input to be echoed. The setting of this variable results from the **-v** command line option.

Other operations treat variables numerically. The **@** command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by dollar sign ($) characters. The expansion can be prevented by preceding the $ with a \, except within double quotes, where it *always* occurs, and within backslashes, where it *never* occurs. Strings between single quotes are interpreted later (see "Command substitution" below) so $ substitution does not occur there until later, if at all. A $ is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in double quotes or given the **:q** modifier, the results of variable substitution can eventually be command and filename substituted.

Within double quotes a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the **:q** modifier is applied to a substitution, the variable expands to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable that is not set.

$name
${name}

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}

Can be used to select only some of the words from the value of *name*. The selector is subjected to **$** substitution and can consist of a single number or two numbers separated by a **-**. The first word of a variables value is numbered **1**. If the first number of a range is omitted it defaults to **1**; if the last member of a range is omitted it defaults to **$#name**. The selector * selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}

Gives the number of words in the variable. This is useful for later use in a [selector].

$0 Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}

Equivalent to **$argv[number]**.

$* Equivalent to **$argv[*]**.

The modifiers **:h**, **:t**, **:r**, **:q**, and **:x** can be applied to the substitutions above as can **:gh**, **:gt**, and **:gr**. If braces { } appear in the command form then the modifiers must appear within the braces. The current implementation allows only one : modifier on each $ expansion.

The following substitutions cannot be modified with : modifiers.

$?name
${?name}
Substitutes the string **1** if name is set, **0** if it is not.

$?0 Substitutes **1** if the current input filename is known, **0** if it is not.

$$ Substitute the (decimal) process number of the (parent) shell.

$< Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of built-in commands. This means that portions of expressions that are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in single quotes. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within double quotes, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters *, ?, [, or { or begins with the character ~, that word is a candidate for filename substitution, also known as "globbing." This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name,

but it is not required for each pattern to match. Only the metacharacters *, ?, and [imply pattern matching, the characters ~, and { being more like abbreviations.

In matching filenames, the dot character (.) at the beginning of a filename or immediately following a /, as well as the character / must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence [...] matches any one of the characters enclosed. Within [...], a pair of characters separated by - matches any character lexically between the two.

The character ~ at the beginning of a filename is used to refer to home directories; standing alone, that is, ~, it expands to the invoker's home directory as reflected in the value of the variable *home*. When ~ is followed by a name consisting of letters, digits and - characters, the shell searches for a user with that name and substitutes their home directory; thus **~ken** might expand to **/usr/ken** and **~ken/chmach** to **/usr/ken/chmach**. If the character ~ is followed by a character other than a letter or /, or appears not at the beginning of a word, it is left undisturbed.

The metanotation **a{b,c,d}e** is a shorthand for **abe ace ade** . Left-to-right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct can be nested. For example, the filename **~source/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/ls.c** whether or not these files exist without any chance of error if the home directory for **source** is **/usr/source**. Similarly **./{memo,*box}** might expand to **./memo ./box ./mbox**. (Note that **memo** was not sorted with the results of matching ***box**.) As a special case {,} and {} are passed undisturbed.

Input/output

The standard input and standard output of a command can be redirected with the following syntax:

< name

Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word

Read the shell input up to a line which is identical to *word. Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, ', ", or ` appears in *word* variable and command substitution is performed on the intervening lines, allowing \ to quote $, \, and `. Commands that are substituted have all blanks, tabs, and newlines

preserved, except for the final newline, which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name

The file *name* is used as standard output. If the file does not exist, it is created; if the file exists, it is truncated, its previous contents being lost.

If the variable *noclobber* is set, the file must not exist or be a character special file (for example, a terminal or **/dev/null**); otherwise an error results. This helps prevent accidental destruction of files. In this case the **!** forms can be used and suppress this check.

The forms involving **&** route the standard error as well as standard output into the specified file. *Name* is expanded in the same way as **<** input filenames. You can use the following syntax to route standard output to one file and standard error to another:

(*cmd* > *file1*) >& *file2*

>> name
>>& name
>>! name
>>&! name

Uses file *name* as standard output like **>**, but places output at the end of the file. If the variable *noclobber* is set, the file must not exist unless one of the **!** forms is given. Otherwise, similar to **>**.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The **<<** mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output can be directed through a pipe with the standard output. Simply use the form **|&** rather than just **|**.

Expressions

A number of the built-in commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the *@*, *exit*, *if*, and *while* commands. The following operators are available:

| | && | ↑ & == != =~ !~ <= >= < > << >> + - * / % ! ~ ( )

Here the precedence increases to the right, ==, !=, =~, and !~, <=, >=, <, and >, <<, and >>, +', and -, *, /, and % being, in groups, at the same level. The ==, !=, =~, and !~ operators compare their arguments as strings; all others operate on numbers. The operators =~ and !~ are like != and == except that the right hand side is a *pattern* (containing, for example, *'s, ?'s and instances of [...]) against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings that begin with **0** are considered to be octal numbers. Null or missing arguments are considered **0**. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions that are syntactically significant to the parser (& |, <, >, (,)) they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in { and } and file enquiries of the form *-lname* where *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, all enquiries return false (0). Command executions succeed, returning true (1) if the command exits with status 0; otherwise they fail, returning false (0). If more detailed status information is required, the command should be executed outside of an expression and the variable *status* examined.

Control flow

The shell contains a number of commands that can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach, switch,* and *while* statements, as well as the *if-then-else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below, under "Built-in commands."

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Built-in commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

alias
alias name
alias name wordlist

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*.

break

Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw

Causes a break from a *switch*, resuming after the *endsw*.

case label:

A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name

Change the shells working directory to directory *name*. If no argument is given then change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory (and does not begin with /, ./, or ../), each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with /, this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing *while* or *foreach*. The remaining commands on the current line are executed.

default:
Labels the default case in a *switch* statement. The default should come after all *case* labels.

dirs Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist
echo -n wordlist
The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the **-n** option is specified. Note that this differs from **/bin/echo**.

else
end
endif
endsw
See the description of the *foreach, if, switch,* and *while* statements below.

eval arg ...
[As in *sh*(1).] The arguments are read as input to the shell and the resulting command(s) executed in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of using *eval*.

exec command
The specified command is executed in place of the current shell.

exit
exit(expr)
The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)

...

end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The built-in command *continue* can be used to continue the loop prematurely and the built-in command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with ? before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist

Like *echo* but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word

The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

history

history *n*

history -r *n*

Displays the history event list; if *n* is given only the *n* most recent events are printed. The **-r** option reverses the order of printout to be most recent first rather than oldest first.

if (expr) command

If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif

If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else.)*

jobs
jobs -l

Lists the active jobs; given the -l options lists process id's in addition to the normal information.

kill %job
kill -sig %job ...
kill pid
kill -sig pid ...
kill -l

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h,* stripped of the prefix "SIG"). The signal names are listed by **kill -l**. There is no default, saying just **kill** does not send a signal to the current job.

limit
limit *resource*

Limits the consumption by the current process and each process it creates to not individually exceed the specified *resource*. If no resource is given, then all limitations are given.

Resources controllable currently include *filesize* (the largest single file which can be created).

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

login

Terminate a login shell, replacing it with an instance of **/bin/login.** This is one way to log off, included for compatibility with *sh*(1).

logout
Terminate a login shell. Especially useful if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command
The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user can specify negative niceness by using **nice -number ....** Command is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

nohup
nohup command
The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with & are effectively *nohup'ed.*

notify
notify %job ...
Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

onintr
onintr -
onintr label
Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form **onintr** - causes all interrupts to be ignored. The final form causes the shell to execute a **goto***label* when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd
popd +*n*
Pops the directory stack, returning to the new top directory. With a argument '+*n*' discards the *n* th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd
pushd name
pushd +n

With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (a la *cd)* and pushes the old current working directory (as in *cwd)* onto the directory stack. With a numeric argument, rotates the *n* th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word.* The fourth form sets the *index' th* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist.* In all cases the value is command and filename expanded.

These arguments can be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variables USER, TERM, PATH, and CDPATH are automatically imported to and exported from the *csh* variables *user*, *term*, *path*, and *cdpath;* there is no need to use *setenv* for these.

shift

shift variable

The members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name

The shell reads commands from *name*. *Source* commands can be nested; if they are nested too deeply the shell can run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands.

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters *, ?, and [...] can be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control can fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time

time command

With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

ulimit -f *n*

imposes a size limit of *n*.
-f imposes a size limit of *n* blocks on files written by child processes (files of any size can be read). With no argument, the current limit is printed.

umask
umask value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern

All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be *unaliased.*

unhash

Use of the internal hash table to speed location of executed programs is disabled.

unset pattern

All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset *'; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset.*

unsetenv pattern

Removes all variables whose name match the specified pattern from the environment. See also the *setenv* command above and *printenv*(1).

wait

All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end

While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* can be used to terminate or continue the loop prematurely. (The

while and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@
@ name = expr
@ name[index] = expr

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains <, >, &, or |, then at least this part of the expression must be placed within ( ). The third form assigns the value of *expr* to the *index'th* argument of *name*. Both *name* and its *index'th* component must already exist. Beware of conflicts between the kill character and this use of **@**.

The operators *=, +=, etc., are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix ++ and -- operators increment and decrement *name* respectively, that is, **@ i++**.

Pre-defined and environment variables

The following variables have special meaning to the shell. Of these, *argv, cwd, home, path, cdpath, prompt, shell* and *status* are always set by the shell. Except for *cwd* and *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file **.cshrc** as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it.

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, that is, **$1** is replaced by **$argv[1]**, etc.

cdpath Gives a list of alternate directories searched to find subdirectories in *chdir* commands.

cwd The full pathname of the current directory.

echo Set when the **-x** command line option is given. Causes each command and its arguments to be echoed just before it is

executed. For non-built-in commands all expansions occur before echoing. Built-in commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character !. The second character of its value replaces the character ↑ in quick substitutions.

history Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of *history* can run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the invoker, initialized from the environment. The filename expansion of ~ refers to this variable.

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by code-D's.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says 'New mail in *name*' when there is mail in the file *name*.

noclobber As described in the section on *Input/output*, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that >> redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, that is, **echo** [still gives an error.

notify If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a prompt.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full path names will execute. The usual search path is ., **/bin**, and **/usr/bin**, but this can vary from system to system. For the super-user the default search path is /etc, **/bin**, and **/usr/bin**. A shell which is given neither the **-c** nor the **-t** option will normally hash the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be found.

prompt The string which is printed before each command is read from an interactive terminal input. If a **!** appears in the string it will be replaced by the current event number unless a preceding \ is given. Default is **%**, or **#** for the super-user.

shell The file in which the shell object code resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Non-built-in Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Built-in commands which fail return exit status **1**, all other built-in commands set status **0**.

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose Set by the **-v** command line option, causes the words of each command to be printed after history substitution.

Non-built-in command execution

When a command to be executed is found to not be a built-in command, the shell attempts to execute the command via *execv*(2). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a **-c** nor a **-t** option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a **-c** or **-t** argument, and in any case for each directory component of *path* which does not begin with a */*, the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus **(cd ; pwd) ; pwd** prints the *home* directory; leaving you where you were (printing this after the home directory), while **cd ; pwd** leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (for example, **$shell**). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell is - then this is a login shell. The flag arguments are interpreted as follows:

-c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

-e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

-f The shell will start faster, because it will neither search for nor execute commands from the file **.cshrc** in the invoker's home directory.

-i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

-n Commands are parsed, but not executed. This aids in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A \ can be used to escape the newline at the end of this line and continue onto another line.

-v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

-x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

-V Causes the *verbose* variable to be set even before **.cshrc** is executed.

-X Is to **-x** as **-V** is to **-v**.

After processing of flag arguments if arguments remain but none of the **-c, -i, -s,** or **-t** options was given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by **$0**. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a "standard" shell if the first character of a script is not a #, that is, if the script does not start with a comment. Remaining arguments initialize the variable *argv*.

Signal handling

The shell normally ignores *quit* signals. Jobs running detached (by the & command) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file **.logout**.

FILES

/etc/cprofile	Read by the login shell before .cshrc.
~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after .cshrc at login.
~/.logout	Read by login shell, at logout.

/bin/sh	Standard shell, for shell scripts not starting with a #.
/tmp/sh*	Temporary file for <<.
/etc/passwd	Source of home directories for ~name.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions can substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), shl(1), access(2), fork(2), pipe(2), umask(2), wait(2), a.out(5).

NOTES

The *csh* interpreter might not be compatible with some shell commands, such as *at*(1), *newgrp*(1), and *wm*(1).

If the first character in an executable file is #, the file is interpreted as a *csh* script. Because # is interpreted as a comment delimiter by *sh*, it is recommended that *sh* scripts begin with a blank line.

BUGS

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by ?, are not placed in the *history* list. *csh* should parse the control structure rather recognizing built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

NAME

csplit - context split

SYNOPSIS

csplit [**-s**] [**-k**] [**-f** prefix] file arg1 [. . . argn]

DESCRIPTION

The *csplit* command reads and separates it into n+1 sections, defined by the arguments *arg1* . . . *argn*. By default the sections are placed in xx00 . . . xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*.

01: From the line referenced by *arg1* up to the line referenced by *arg2*.

:

:

n+1: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a - then standard input is used.

The options to *csplit* are:

-s *csplit* normally prints the character counts for each file created. If the **-s** option is present, *csplit* suppresses the printing of all character counts.

-k *csplit* normally removes created files if an error occurs. If the **-k** option is present, *csplit* leaves previously created files intact.

-f *prefix* If the **-f** option is used, the created files are named *prefix***00** . . . *prefix*n. The default is **xx00** . . . **xx***n*.

The arguments (*arg1* . . . *argn*) to *csplit* can be a combination of the following:

/rexp/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or - some number of lines (for example, **/Page/-5**).

%rexp% This argument is the same as */rexp/*, except that no file is created for the section.

lnno A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{num} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, **cobol00 . . . cobol03**. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

SEE ALSO

ed(1), sh(1), regexp(5).

DIAGNOSTICS

Self-explanatory except for:

arg - out of range

which means that the given argument did not reference a line between the current position and the end of the file.

NAME

ct - spawn getty to a remote terminal

SYNOPSIS

ct [**-wn**] [**-xn**] [**-h**] [**-v**] [-sspeed] telno ...

DESCRIPTION

ct dials the telephone number of a modem that is attached to a terminal, and spawns a *getty* process to that terminal. *Telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. (The set of legal characters for *telno* is 0 thru 9, -, =, *, and #. The maximum length *telno* is 31 characters). If more than one telephone number is specified, *ct* will try each in succession until one answers; this is useful for specifying alternate dialing paths.

ct will try each line listed in the file **/usr/lib/uucp/Devices** until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, *ct* will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. *ct* will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the **-w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

The **-x***n* option is used for debugging; it produces a detailed output of the program execution on stderr. The debugging level, *n*, is a single digit; **-x9** is the most useful value.

Normally, *ct* will hang up the current line, so the line can answer the incoming call. The **-h** option will prevent this action. The **-h** option will also wait for the termination of the specified *ct* process before returning control to the user's terminal. If the **-v** option is used, *ct* will send a running narrative to the standard error output stream.

The data rate may be set with the **-s** option, where *speed* is expressed in baud. The default rate is 1200.

After the user on the destination terminal logs out, there are two things that could occur depending on what type of getty is on the line (*getty* or *uugetty*). For the first case, *ct* prompts, **Reconnect?** If the response begins with the letter **n**, the line will be dropped; otherwise, *getty* will be started again and the **login:** prompt will be printed. In the second case, there is already a getty (*uugetty*) on the line, so the **login:** message will appear.

To log out properly, the user must type **control D**.

Of course, the destination terminal must be attached to a modem that can answer the telephone.

FILES

/usr/lib/uucp/Devices
/usr/adm/ctlog

SEE ALSO

cu(1C), getty(1M), login(1), uucp(1C), uugetty(1M).
S/Series CTIX Administrator's Guide.

WARNING

For a shared port, one used for both dial-in and dial-out, the *uugetty* program running on the line must have the **-r** option specified [see *uugetty*(1M)].

NAME

ctags - create a tags file

SYNOPSIS

ctags [**-u**] [**-v**] [**-w**] [**-x**] name ...

DESCRIPTION

ctags creates a tags file, *tags*, from the specified C, Pascal, and FORTRAN sources. The *ex*(1) **tags** command uses a tags file to find specified objects, functions in this case, in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition, with the fields separated by blanks or tabs.

If a file's name ends with **.c** or **.h**, it is searched for C function and macro definitions. The *main* function is treated as a special case, so as to permit multiple programs in one directory: the tag is the name of the file, stripped of leading directory names and trailing **.c**, with **M** prepended.

If a file's name does not end with **.c** or **.h**, it is searched for Pascal definitions, then for FORTRAN definitions, then for C definitions.

These are the options:

-w No warning diagnostics.

-u Update the tags file. (It is usually faster just to rebuild the tags file.)

-a Append new definitions to the end of the tags file.

-x Process a list of function definitions, with line numbers and file names.

FILES

tags output tags file

SEE ALSO

ex(1), vi(1).

WARNING

Recognition of FORTRAN and Pascal objects is done in a very simpleminded way. No attempt is made to deal with block structure.

NAME

ctinstall - install software

SYNOPSIS

/usr/local/bin/ctinstall [update | install] [groups ...]

DESCRIPTION

The *ctinstall* command is used to install software from quarter-inch tape and diskette media. It should be invoked in single-user mode.

Note: Not all software can be installed with *ctinstall*; check the software Release Notice for applicability.

Before executing *ctinstall*, the user should *cd* to the directory under which files will be installed. (Normally this is /.) The user must ensure that all necessary mounted file systems are mounted.

If no arguments are provided to *ctinstall*, the user will be prompted for the required information. The option *install* is for raw, or first installs; *update* is for software updates; *silent* is the same as *update* but with fewer questions asked (*silent* is recommended); *groups* is any number of group names specified in the software product's associated proto file.

EXAMPLE

A sample installation session is illustrated here. User responses are shown in **bold** type.

 # cd /
 # shutdown
Ok To Stop Or Reset Processor
mount /usr
 # /usr/local/bin/ctinstall

@(#)ctinstall.sh 6.31

Positioning the Tape for Product Installation.

Update or new installation of ISAM 5.00 ('update', 'silent', or 'install')?: install

Please enter your group choices for ISAM separated by blanks.
Your choices are:

ISAM

If you'd like all of the groups, type 'all': ISAM

This procedure will install the following ISAM 5.00 group(s) on your system:

ISAM

BE SURE YOU BACK UP ANYTHING YOU HAVE CHANGED
BEFORE PROCEEDING.

Type 'yes' to confirm: yes

Starting to Install Group(s) ISAM.
Installing Group ISAM.

Calculating size required for group ISAM.
NNNN 512 byte blocks will be used on /
NNNN inodes will be used on /
NNNN 512 byte blocks will be freed on /usr

Installing required ISAM files.
install/IsamRel
usr/include/isam.h
usr/include/iserc.h
usr/lib/isam/IsamConfig
usr/lib/isam/IsamCreate
usr/lib/isam/IsamProtect
usr/lib/isam/IsamReorg
usr/lib/isam/IsamStat
usr/lib/isam/IsamStop
usr/lib/isam/IsamTransfer
usr/lib/isam/IxFilter
usr/lib/isam/IxSpec
usr/lib/isam/isam

Checking permissions, modes and omissions on new ISAM commands.
Completed Installation of Group ISAM.
Rewinding tape.

Installation Complete.

SEE ALSO

qlist(1), qinstall(1).
Appropriate Release Notice for the software product you are installing.

NAME

ctrace - C program debugger

SYNOPSIS

ctrace [options] [file]

DESCRIPTION

The *ctrace* command allows you to follow the execution of a C program, statement-by-statement. The effect is similar to executing a shell procedure with the **-x** option. *ctrace* reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc*(1) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(1) or *tail*(1) commands.

The options commonly used are:

-f *functions*	Trace only these *functions*.
-v *functions*	Trace all but these *functions*.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:

-o	Octal
-x	Hexadecimal
-u	Unsigned
-e	Floating point

These options are used only in special circumstances:

-l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

-s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

-t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.

-P Run the C preprocessor on the input before tracing it. You can also use the **-D**, **-I**, and **-U** *cpp*(1) options.

These options are used to tailor the run-time trace package when the traced program will run in an environment other than CTIX or other UNIX-compatible systems:

-b Use only basic functions in the trace code, that is, those in *ctype*(3C), *printf*(3S), and *string*(3C). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(2), *fflush*(3S), *longjmp*(3C), or *setjmp*(3C).

-p *string*
Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.

-r *f* Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **-p** option).

EXAMPLE

If the file *lc.c* contains this C program:

```
1 #include <stdio.h>
2 main() /* count lines in input */
3 {
4       int c, nl;
5
6       nl = 0;
7       while ((c = getchar()) != EOF)
8               if (c = '\n')
```

```
9                               ++nl;
10          printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
(cntl-d)
```

the program will be compiled and executed. The output of the program will be the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
2 main()
6           nl = 0;
            /* nl == 0 */
7           while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
            /* c == 49 or '1' */
8                   if (c = '\n')
                    /* c == 10 or '\n' */
9                           ++nl;
                            /* nl == 1 */
7           while ((c = getchar()) != EOF)
            /* c == 10 or '\n' */
8                   if (c = '\n')
                    /* c == 10 or '\n' */
9                           ++nl;
                            /* nl == 2 */
7           while ((c = getchar()) != EOF)
```

If you now enter an end of file character (cntl-d) the final output will be:

```
            /* c == -1 */
```

```
10      printf("%d\n", nl);
        /* nl == 2 */
        return
```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equality operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the **-f** or **-v** options to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding *ctroff*() and *ctron*() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from *sdb*(1) if you compile with the **-g** option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
r
```

You can also turn the trace off and on by setting static variable tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly.

FILES

/usr/lib/ctrace/runtime.c	run-time trace package

SEE ALSO

bfs(1), tail(1), signal(2), ctype(3C), fclose(3S), printf(3S), setjmp(3C), string(3C).

DIAGNOSTICS

This section contains diagnostic messages from both *ctrace* and *cc*(1), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

ctrace Diagnostics

warning: some variables are not traced in this statement

Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the **-t** option to increase this number.

warning: statement too long to trace

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use -P option

This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

'if ... else if' sequence too long

Split the sequence by removing an **else** from the middle.

possible syntax error, try -P option

Use the **-P** option to preprocess the *ctrace* input, along with any appropriate **-D**, **-I**, and **-U** preprocessor options. If you still get the error message, check the Warnings section below.

Cc Diagnostics

warning: illegal combination of pointer and integer
warning: statement not reached
warning: sizeof returns 0

Ignore these messages.

compiler takes size of function

See the *ctrace* "possible syntax error" message above.

yacc stack overflow

See the *ctrace* "'if ... else if' sequence too long" message above.

out of tree space; simplify expression

Use the -t option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

redeclaration of signal

Either correct this declaration of *signal*(2), or remove it and #include <signal.h>.

WARNINGS

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, for example, "int EOF;", will cause a syntax error.

BUGS

ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. *ctrace* may choose to print the address of an aggregate or use the wrong format (for example, 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

NAME

cu - call another UNIX system

SYNOPSIS

cu [-sspeed] [-lline] [**-h**] [**-t**] [**-d**] [**-o**] [**-e**] [**-n**] [**-m**] telno

cu [**-s** speed] [**-h**] [**-d**] [**-o**] [**-e**] [**-m**] **-l** line

cu [**-h**] [**-d**] [**-o**] [**-e**] [**-m**] systemname

DESCRIPTION

The *cu* program calls up another computer system or a terminal. It manages an interactive conversation with possible transfers of ASCII files.

The following options and arguments are valid to *cu*:

-s*speed* Specifies the transmission speed (300, 1200, 2400, 4800, 9600); The default value is **Any**, which depends on the order of the lines in the **/usr/lib/uucp/Devices** file.

-l*line* Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the **-l** option is used without the **-s** option, the speed of a line is taken from the Devices file. When the **-l** and **-s** options are both used together, *cu* searches the Devices file to check if the requested speed for the requested line is available. If so, the connection is made at the requested speed; otherwise an error message is printed and the call is not made. The specified device is generally a directly connected asynchronous line (like **/dev/tty***xxx*) in which case a telephone number (*telno*) is not required. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* does not give the desired result (see *systemname* below).

Note that modem control is ignored if the **-l** option is used.

-h Emulates local echo, supporting calls to other computer systems that expect terminals to be set to half-duplex mode.

-t	Used to dial an ASCII terminal that is set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.
-d	Causes diagnostic traces to be printed.
-o	Designates that odd parity is to be generated for data sent to the remote system.
-m	Designates a direct line that has modem control.
-n	For added security, prompts the user to provide the telephone number to be dialed rather than taking it from the command line.
-e	Designates that even parity is to be generated for data sent to the remote system.
telno	When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.
systemname	A *uucp* system name can be used rather than a telephone number; in this case, *cu* obtains an appropriate direct line or telephone number from **/usr/lib/uucp/Systems**. Note that the *systemname* option should not be used in conjunction with the **-l** and **-s** options, as *cu* connects to the first available line for the system name specified, ignoring the requested line and speed.

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~ (tilde), passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following user-initiated commands:

~.	Terminate the conversation.
~!	Escape to an interactive shell on the local system.
~!*cmd*...	Run *cmd* on the local system (through **sh -c**).

~$*cmd*...	Run *cmd* locally and send its output to the remote system.
~%**cd**	Change the directory on the local system. Note that ~!**cd** causes the command to be run by a subshell, probably not the intent.
~%**take** *from* [*to*]	Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%**put** *from* [*to*]	Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places. For both ~%**take** and **put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
~~ *line*	Send the line ~ *line* to the remote system.
~%**break**	Transmit a **BREAK** to the remote system (which can also be specified as ~%**b**).
~%**debug**	Toggle the **-d** debugging option on or off (which can also be specified as ~%**d**).
~**t**	Print the values of the **termio** structure variables for the user's terminal (useful for debugging).
~**l**	Print the values of the **termio** structure variables for the remote communication line (useful for debugging).
~%**nostop**	Toggle between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with ~.

Data from the remote is diverted (or appended, if >> is used) to *file* on the local system. The trailing ~> marks the end of the diversion.

The use of **~%put** requires *stty*(1) and *cat*(1) on the remote side, and can be used only with **sh**. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of *echo*(1) and *cat*(1) on the remote system, and can be used only with **sh**. Also, *tabs* mode should be set on the remote system if tabs are to be copied without expansion to spaces [see *stty*(1)].

When *cu* is used on system X to connect to system Y, and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. Executing a tilde command reminds the user of the local system uname. For example, *uname* can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!uname
Y
```

In general, ~ causes the command to be executed on the original machine, ~~ causes the command to be executed on the next machine in the chain.

EXAMPLES

The following command dials a system with telephone number is 9 201 555 1212, using 1200 baud (where a dialtone is expected after the 9):

```
cu -s1200 9=12015551212
```

If the speed is not specified, "Any" is the default value.

The following command logs in to a system connected by a direct line:

```
cu -l /dev/ttyXXX
```

or

```
cu -l ttyXXX
```

The following command dials a system with the specific line and a specific speed:

```
cu -s1200 -l ttyXXX
```

The following command dials a system using a specific line associated with an auto dialer:

```
cu -l culXXX 9=12015551212
```

The following command uses a system name:

```
cu systemname
```

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/spool/locks/LCK..(tty-device)

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uucp(1C), uname(1).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

Exit code is zero for normal exit, otherwise, one.

WARNINGS

The *cu* command does not perform any integrity checking on data it transfers. Data fields with special *cu* characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a ~. to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands. If the remote system is using *uugetty*(1M), a carriage-return may be needed to get a prompt.

BUGS

There is an artificial slowing of transmission by *cu* during the **~%put** operation so that loss of data is unlikely.

NAME

cut - cut out selected fields of each line of a file

SYNOPSIS

cut -clist [file ...]

cut -flist [**-d** char] [**-s**] [file ...]

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, that is, character positions as on a punched card (**-c** option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (**-f** option). *cut* can be used as a filter; if no files are given, the standard input is used. In addition, a file name of "-" explicitly refers to standard input.

The meanings of the options are:

list A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges [for example, **1,4,7**; **1-3,8**; **-5,10** (short for **1-5,10**); or 3- (short for third through last field)].

-c*list* The *list* following **-c** (no space) specifies character positions (for example, **-c1-72** would pass the first 72 characters of each line).

-f*list* The *list* following **-f** is a list of fields assumed to be separated in the file by a delimiter character (see **-d**); for example, **-f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless **-s** is specified.

-d*char* The character following **-d** is the field delimiter (**-f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.

-s Suppresses lines with no delimiter characters in case of **-f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the **-c** or **-f** option must be specified.

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (that is, horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLES

To map user IDs to names:

```
cut -d: -f1,5 /etc/passwd
```

To set **name** to current login name:

```
name=`who am i | cut -f1 -d" "`
```

SEE ALSO

grep(1), paste(1).

DIAGNOSTICS

ERROR: line too long

A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c/f option

Missing **-c** or **-f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

ERROR: no fields

The *list* is empty.

ERROR: no delimeter

Missing *char* on **-d** option.

ERROR: cannot handle multiple adjacent backspaces

Adjacent backspaces cannot be processed correctly.

WARNING: cannot open <filename>

Either *filename* cannot be read or does not exist. If multiple filenames are present, processing continues.

NAME

cw, checkcw - prepare constant-width text for troff

SYNOPSIS

cw [**-lxx**] [**-rxx**] [**-fn**] [**-t**] [**+t**] [**-d**] [files]

checkcw [**-lxx**] [**-rxx**] files

DESCRIPTION

cw is a preprocessor for *troff*(1) input files that contain text to be typeset in the constant-width (CW) font.

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. (An earlier version of this font was used in typesetting *The C Programming Language* by B. W. Kernighan and D. M. Ritchie.) It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

Because the CW font contains a ''non-standard'' set of characters and because text typeset with it requires different character and inter-word spacing than is used for ''standard'' fonts, documents that use the CW font must be preprocessed by *cw*.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$&()''*+@.,/:;=?[]|-_^~"<>{}#
```

plus eight non-ASCII characters represented by four-character *troff*(1) names (in some cases attaching these names to ''non-standard'' graphics):

Character	*Symbol*	*Troff Name*
''Cents'' sign	¢	\(ct
EBCDIC ''not'' sign	¬	\(no
Left arrow	←	\(<-
Right arrow	→	\(->
Down arrow	↓	\(da
Vertical single quote	´	\(fm
Control-shift indicator	†	\(dg

Visible space indicator	□	\(sq
Hyphen	-	\(hy

The hyphen is a synonym for the unadorned minus sign (-). Certain versions of *cw* recognize two additional names: \(ua for an up arrow and \(lh for a diagonal left-up (home) arrow.

cw recognizes five request lines, as well as user-defined delimiters. The request lines look like *troff*(1) macro requests, and are copied in their entirety by *cw* onto its output; thus, they can be defined *by the user* as *troff*(1) macros; in fact, the .CW and .CN macros *should* be so defined (see *HINTS* below). The five requests are:

.CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the *cw* command line.

.CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the *cw* command line.

.CD Change delimiters and/or settings of other options; takes the same options as are available on the *cw* command line.

.CP *arg1 arg2 arg3 ... argn*
All the arguments (which are delimited like *troff*(1) macro arguments) are concatenated, with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

.PC *arg1 arg2 arg3 ... argn*
Same as .CP, except that the even-numbered arguments are set in the CW font and the odd-numbered ones in the prevailing font.

The .CW and .CN requests are meant to bracket text (for example, a program fragment) that is to be typeset in the CW font "as is." Normally, *cw* operates in the *transparent* mode. In that mode, except for the .CD request and the nine special four-character names listed in the table above, every character between .CW and .CN request lines stands for itself. In particular, *cw* arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes (\) everywhere to be "hidden" from *troff*(1). The transparent mode can be turned off (see below), in which case normal *troff*(1) rules apply; in particular, lines that begin with . and ' are passed through untouched (except if they contain delimiters-see below). In either case, *cw* hides the effect of the font changes generated by the .CW and .CN requests; *cw* also defeats all ligatures (fi, ff, etc.) in the CW font.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

The user can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text (see example under *BUGS* below). *cw* treats text between delimiters in the same manner as text enclosed by .CW/.CN pairs, except that, for aesthetic reasons, spaces and backspaces inside .CW/.CN pairs have the same width as other CW characters, while spaces and backspaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text (but are *not* adjustable). Font changes due to delimiters are *not* hidden.

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

-l*xx* The one- or two-character string *xx* becomes the left delimiter; if *xx* is omitted, the left delimiter becomes undefined, which it is initially.

-r*xx* Same for the right delimiter. The left and right delimiters may (but need not) be different.

-f*n* The CW font is mounted in font position *n*; acceptable values for *n* are 1, 2, and 3 (default is 3, replacing the bold font). This option is only useful at the beginning of a document.

-t Turn transparent mode *off*.

+t Turn transparent mode *on* (this is the initial default).

-d Print current option settings on file descriptor 2 in the form of *troff*(1) comment lines. This option is meant for debugging.

cw reads the standard input when no *files* are specified (or when - is specified as the last argument), so it can be used as a filter. Typical usage is:

cw *files* | troff ...

checkcw checks that left and right delimiters, as well as the .CW/.CN pairs, are properly balanced. It prints out all offending lines.

HINTS

Typical definitions of the .CW and .CN macros meant to be used with the *mm*(5) macro package:

```
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ...
..
```

```
.de CN
.ta .5i 1i 1.5i 2i 2.5i 3i ...
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *troff*(1) no-fill (.nf) mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point *smaller* than the prevailing point size (the displayed definitions of .CW and .CN above are one point smaller than the running text on this page). The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: *cw*, *tbl*, and *eqn*. Usually, the tables contained in such documents will not contain any CW text, although it is entirely possible to have *elements* of the table set in the CW font; of course, care must be taken that *tbl*(1) format information not be modified by *cw*. Attempts to set equations in the CW font are not likely to be either pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting <- represent a backspace, d<-<-\(dg yields ‡. (Because backspaces are half as wide between delimiters as inside .CW/.CN pairs-see above-two backspaces are required for each overstrike between delimiters.)

FILES

/usr/lib/font/ftCW CW font-width table

SEE ALSO

eqn(1), mmt(1), tbl(1), troff(1), mm(5), mv(5).

WARNINGS

If text preprocessed by *cw* is to make any sense, it must be set on a typesetter equipped with the CW font or on a STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

BUGS

Periods (.), backslashes (), and double quotes (") do not work well as delimiters or as arguments to .CP and .PC.

Certain CW characters don't concatenate gracefully with certain Times Roman characters, for example, a CW ampersand (&) followed by a Times Roman comma(,); in such cases, judicious use of *troff*(1) half- and quarter-spaces (\

and \^) is most salutary. For example, use _&_\^, (rather than _&_,) to obtain &, (assuming that _ is used for both delimiters).

Using *cw* with *nroff* is not worthwhile.

The output of *cw* is hard to read.

See also *BUGS* under *troff*(1).

NAME

cxref - generate C program cross-reference

SYNOPSIS

cxref [options] files

DESCRIPTION

The *cxref* command analyzes a collection of C files and attempts to build a cross-reference table. *cxref* uses a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or, with the **-c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **-D**, **-I** and **-U** options [which are interpreted just as they are by *cc*(1) and *cpp*(1)], the following *options* are interpreted by *cxref*:

-c Print a combined cross-reference of all input files.

-w*<num>*
Width option which formats output no wider than <num> (decimal) columns. This option will default to 80 if <num> is not specified or is less than 51.

-o file Direct output to *file*.

-s Operate silently; do not print input file names.

-t Format listing for 80-column width.

FILES

LLIBDIR	usually /usr/lib
LLIBDIR/xcpp	special version of the C preprocessor.

SEE ALSO

cc(1), cpp(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files.

BUGS

cxref considers a formal argument in a *#define* macro definition to be a declaration of that symbol. For example, a program that *#includes* **ctype.h**, will contain many declarations of the variable **c**.

NAME

date - print and set the date

SYNOPSIS

date [+ format]
date [mmddhhmm [[yy] | [ccyy]]]
date [-]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set (only by superuser).

The system has a time-of-day clock that can be used to set the current system date. The command

date -

sets the system time to that of the time-of-day clock. If arguments are given, *date* changes the time on the time-of-day clock as well as the system time.

The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *cc* is the century minus one and is optional; *yy* is the last 2 digits of the year number and is optional. For example:

date 10080045

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

If the argument begins with +, the output of *date* is under the control of the user. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character. If the argument contains embedded blanks it must be quoted (see the **EXAMPLE** section).

Specifications of native language translations of month and weekday names are supported. The language used depends on the value of the environment variable **LANGUAGE** [see *environ*(5)]. The month and weekday names used for a language are taken from strings in the file for that language in the **/lib/cftime** directory [see *cftime*(4)].

After successfully setting the date and time, *date* will display the new date according to the format defined in the environment variable **CFTIME** [see *environ*(5)].

Field Descriptors (must be preceded by a %):

- **a** abbreviated weekday name
- **A** full weekday name
- **b** abbreviated month name
- **B** full month name
- **d** day of month - 01 to 31
- **D** date as mm/dd/yy
- **e** day of month - 1 to 31 (single digits are preceded by a blank)
- **h** abbreviated month name (alias for %b)
- **H** hour - 00 to 23
- **I** hour - 01 to 12
- **j** day of year - 001 to 366
- **m** month of year - 01 to 12
- **M** minute - 00 to 59
- **n** insert a new-line character
- **p** string containing ante-meridiem or post-meridiem indicator (by default, AM or PM)
- **r** time as *hh:mm:ss pp* where *pp* is the ante-meridiem or post-meridiem indicator (by default, AM or PM)
- **R** time as hh:mm
- **S** second - 00 to 59
- **t** insert a tab character
- **T** time as *hh:mm:ss*
- **U** week number of year (Sunday as the first day of the week) - 01 to 52
- **w** day of week - Sunday = 0
- **W** week number of year (Monday as the first day of the week) - 01 to 52
- **x** Country-specific date format
- **X** Country-specific time format
- **y** year within century - 00 to 99
- **Y** year as *ccyy* (4 digits)
- **Z** timezone name

EXAMPLE

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

would have generated as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

DIAGNOSTICS

No permission
if you are not the super-user and you try to change the date;

bad conversion
if the date set is syntactically incorrect.

SEE ALSO

cftime(4), environ(5).
S/Series CTIX Administrator's Guide.

NOTE

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dbconsole - change the kernel debugger system console port

SYNOPSIS

dbconsole [**-v**] [**-s** port] [**-i** flags] [**-q** flags] [**-p** dest_op]

DESCRIPTION

The *dbconsole* command is used to specify the port the kernel debugger uses for the system console. (By default, the kernel debugger uses **/dev/tty000**). The *dbconsole* command can also be used to enable, disable, and change the destination of debug prints.

Options to *dbconsole* are interpreted as follows:

-v Print the current settings of the console port and debug print settings.

-s *port* Change the port the kernel debugger uses for its console to *port*, where *port* is the number of an RS-232 port (**0** for **/dev/tty000**, **1** for **/dev/tty001**, and so forth).

-i *flags*

-q *flags* Toggle the debugger <ki> and <kq> flags. Possible flags follow:

a - z
{ | } ~ _ `

(Where flags have special significance to a shell, they must be enclosed in quotes or escaped with \.) The meanings of the various flags are described in the file **/usr/include/sys/kprintf.h**.

-p *dest_op*

Disable/enable kernel prints; change the destination of kernel prints. Legal values for *dest_op* follow:

0 kernel debugging prints disabled

1 kernel debugging prints enabled(screen)

2 kernel debugging prints enabled(printer)

3 kernel debugging prints enabled(screen+printer)

4 kernel debugging prints enabled(memory log)

12 kernel debugging prints enabled(log->file)

13 kernel debugging prints enabled(log->file+screen)

RETURN VALUE

The *dbconsole* command returns either 255 (for any error) or the current debugger console port number.

EXAMPLES

```
dbconsole -q "fgh" -p 4
```

Sends *fprintf*, *gprintf*, and *hprintf* output to the memory log.

```
dbconsole -s 1
```

Sets the current console port to **/dev/tty001**.

FILES

/dev/console

NAME

dc - desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. [See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.] 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 are printed.

q Exits the program. If executing a string, the recursion level is popped by two.

Q Exits the program. 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 evaluated 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 CTIX system 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*(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

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

SEE ALSO

bc(1).

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.

NAME

dcopy - copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [**-sX**] [**-an**] [**-d**] [**-v**] [**-f** fsize[: isize]] inputfs outputfs

DESCRIPTION

The *dcopy* command copies file system *inputfs* to *outputfs*. *Inputfs* is the device file for the existing file system; *outputfs* is the device file to hold the reorganized result. For the most effective optimization *inputfs* should be the raw device and *outputfs* should be the block device. Both *inputfs* and *outputfs* should be unmounted file systems. When using *dcopy* on the normal **root** file system, perform this procedure while booted from the maintenance tape).

With no options, *dcopy* copies files from *inputfs*, compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options follow:

-s*X* Supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the **-s** option of *fsck*(1M).

-a*n* Place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified, no movement occurs.

-d Leave order of directory entries as is (default is to move subdirectories to the beginning of directories).

-v Currently reports how many files were processed and how big the source and destination freelists are.

-f*size*[**:***isize*] Specify the *outputfs* file system and inode list sizes (in blocks). If the option (or **:***isize*) is not given, the values from the *inputfs* are used.

The *dcopy* command catches interrupts and quits, and reports on its progress. To terminate *dcopy* send a quit signal, followed by an interrupt or quit.

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

NAME

dd - convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

The *dd* command 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=*file*	input file name; standard input is default
of=*file*	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 in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
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 block to *ibs*
c1 , *c2*	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 multiplication.

cbs is used only if *conv=ascii* or *conv=ebcdic* is specified. In the former case, *cbs* characters are placed into the conversion buffer (converted to ASCII).

Trailing blanks are trimmed and a 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). Blanks are added to make up an output block of size *cbs*.

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

DIAGNOSTICS

f+p blocks in(out)

numbers of full and partial blocks read(written)

NAME

delta - make a delta (change) to an SCCS file

SYNOPSIS

delta [**-r**SID] [**-s**] [**-n**] [**-g**listi] [**-m**[mrlist]] [**-y**[comment]] [**-p**] files

DESCRIPTION

The *delta* command permanently introduces into the named SCCS file changes made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

The *delta* command makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

The *delta* command can issue prompts on the standard output depending upon certain keyletters specified and flags [see *admin*(1)] that may be present in the SCCS file (see **-m** and **-y** keyletters below).

Keyletter arguments apply independently to each named file.

-r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *get*s for editing (**get -e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **-r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command [see *get*(1)]. A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

-s Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

-g*list* a *list* [see *get*(1) for the definition of *list*] of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

-m*[mrlist]* If the SCCS file has the **v** flag set [see *admin*(1)] then a Modification Request (**MR**) number *must* be supplied as the reason for creating the new delta.

If **-m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **-y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value [see *admin*(1)], it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, *delta* terminates. (It is assumed that the **MR** numbers were not all valid.)

-y*[comment]* Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

g-file Existed before the execution of *delta*; removed after completion of *delta*.

p-file Existed before the execution of *delta*; may exist after completion of *delta*.

q-file Created during the execution of *delta*; removed after completion of *delta*.

x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4).
UNIX System V Release 3.2 Programmer's Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS [see *sccsfile*(4)] and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (-) is specified on the *delta* command line, the **-m** (if necessary) and **-y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

NAME

deroff - remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [**-mx**] [**-w**] [files]

DESCRIPTION

The *deroff* command reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **-m** option may be followed by an **m**, **s**, or **l**. The **-mm** option causes the macros to be interpreted so that only running text is output (that is, no text from macro lines.) The **-ml** option forces the **-mm** option and also causes deletion of lists associated with the **mm** macros.

If the **-w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (&), and apostrophes (´); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1).

BUGS

deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.
The **-ml** option does not handle nested lists correctly.

NAME

devnm - device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

The *devnm* command identifies the special file associated with the mounted file system where the argument *name* resides.

This command is most commonly used by **/etc/brc** (see *brc*(1M)) to construct a mount table entry for the **root** device.

EXAMPLE

The command:

```
/etc/devnm /usr
```

produces:

```
/dev/dsk/c0d0s3 usr
```

if **/usr** is mounted on **/dev/dsk/c0d0s3**.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M).

NAME

df - report number of free disk blocks and i-nodes

SYNOPSIS

df [**-lt**] [**-f**] [file-system | directory | mounted-resource]

DESCRIPTION

The *df* command prints out the number of free 512-byte blocks and free i-nodes in mounted file systems, directories, or mounted resources by examining the counts kept in the super-blocks.

file-system may be specified either by device name (for example, **/dev/dsk/c0d0s3**) or by mount point directory name (for example, **/usr**).

directory can be a directory name. The report presents information for the device that contains the directory.

mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

The *df* command uses the following options:

-l only reports on local file systems.

-t causes the figures for total allocated blocks and i-nodes to be reported as well as the free blocks and i-nodes.

-f an actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free i-nodes are not reported). This option will not print any information about mounted remote resources.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mount(1M), fs(4), mnttab(4).

NOTE

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

NAME

diff - differential file comparator

SYNOPSIS

diff [**-efbhB**] 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 with the name *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 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 >.

The **-b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **-e** option produces a script of *a*, *c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **-f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

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

Option **-h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **-e** and **-f** are unavailable with **-h**.

The **-B** option is similar to the **-b** option except that it causes a null white space string to compare equal to a non-null white space string.

FILES

/tmp/d?????
/usr/lib/diffh for **-h**

SEE ALSO

bdiff(1), cmp(1), comm(1), ed(1).

DIAGNOSTICS

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

WARNINGS

Missing newline at end of file X

Indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

BUGS

Editing scripts produced under the **-e** or **-f** option are naive about creating lines consisting of a single period (.).

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

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*, that is, 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/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat **-e**.
Files longer than 64K bytes will not work.

NAME

diffmk - mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff* or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file **macs** contains:

```
.pl 1
.ll 77
.nf
.eo
.nc `
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** and **.nc** requests are probably needed only for C programs.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

SEE ALSO

diff(1), nroff(1), troff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, that is, replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

NAME

dircmp - directory comparison

SYNOPSIS

dircmp [**-d**] [**-s**] [**-w***n*] [**-o**] dir1 dir2

DESCRIPTION

dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

-d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

-s Suppress messages about identical files.

-w*n* Change the width of the output line to *n* characters. The default width is 72.

-o Ignore time and date stamp differences when comparing the contents of binary files.

SEE ALSO

cmp(1), diff(1).

NAME

dis - object code disassembler

SYNOPSIS

dis [**-o**] [**-V**] [**-L**] [**-s**] [**-d** sec] [**-da** sec]
[**-F** function] [**-t** sec] [**-l** string] file ...

DESCRIPTION

The *dis* command produces an assembly language listing of *file*, which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

The following *options* are interpreted by the disassembler and may be specified in any order.

-o Print numbers in octal. The default is hexadecimal.

-V Print, on standard error, the version number of the disassembler being executed.

-L Lookup source labels in the symbol table for subsequent printing. This option works only if the file was compiled with additional debugging information [for example, the **-g** option of *cc*(1)].

-s Perform symbolic disassembly - that is, specify source symbol names for operands where possible. Symbolic disassembly output will appear on the line following the instruction. For maximal symbolic disassembly to be performed, the file must be compiled with additional debugging information [for example, the **-g** option of *cc*(1)]. Symbol names will be printed using C syntax.

-d *sec* Disassemble the named section as data, printing the offset of the data from the beginning of the section.

-da *sec* Disassemble the named section as data, printing the actual address of the data.

-F*function* Disassemble only the named function in each object file specified on the command line. The **-F** option may be specified multiple times on the command line.

-t *sec* Disassemble the named section as text.

-l *string* Disassemble the library file specified by *string*. For example, one would issue the command **dis -l x -l z** to disassemble **libx.a** and **libz.a**. All libraries are assumed to be in *LIBDIR*.

If the **-d**, **-da** or **-t** options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as **[5]**, represents that the break-pointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information [for example, the **-g** option of *cc*(1)]. An expression such as **<40>** in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A function name will appear in the first column, followed by **( )**.

FILES

LIBDIR usually /lib.

SEE ALSO

as(1), cc(1), ld(1), a.out(4).

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NAME

diskusg - generate disk accounting data by user ID

SYNOPSIS

diskusg [options] [files]

DESCRIPTION

The *diskusg* command generates intermediate disk accounting information from data in *files*, or the standard input if omitted. *diskusg* outputs lines on the standard output, one per user, in the following format:

uid login #blocks

where:

uid — is the numerical user ID of the user.

login — is the login name of the user; and

#blocks — is the total number of 512-byte disk blocks allocated to this user.

diskusg normally reads only the i-nodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

diskusg recognizes the following options:

-s — the input data is already in *diskusg* output format. *diskusg* combines all lines for a single user into a single line.

-v — verbose. Print a list on standard error of all files that are charged to no one.

-i*fnmlist* — ignore the data on those file systems whose file system name is in *fnmlist*. *fnmlist* is a list of file system names separated by commas or enclosed within quotes. *diskusg* compares each name in this list with the file system name stored in the volume ID [see *labelit(1M)*].

-p*file* — use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.

-u*file* — write records to *file* of files that are charged to no one. Records consist of the special file name, the i-node number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk* [see *acct*(1M)] which generates total accounting records that can be merged with other accounting records. *diskusg* is normally run in *dodisk* [see *acctsh*(1M)].

EXAMPLES

The following will generate daily disk accounting information:

```
for i in s1 s3; do
        diskusg /dev/dsk/c0d0$i > dtmp.`basename $i` &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

/etc/passwd	used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M), acct(4).
S/Series CTIX Administrator's Guide.

NAME

dname - print Remote File Sharing domain and network names

SYNOPSIS

dname [**-D** domain] [**-N** netspec] [**-dna**]

DESCRIPTION

The *dname* command prints or defines a host's Remote File Sharing (RFS) domain name or the network used by RFS as transport provider. When used with **d**, **n**, or **a** options, *dname* can be run by any user to print the domain name, network name or both, respectively. Only a user with **root** permission can use the **-D** *domain* option to set the domain name for the host or **-N** *netspec* to set the network specification used for RFS. (The value of *netspec* is the network device name, relative to the */dev* directory. For example, if the transport provider is TCP, the value for *netspec* is **inet/tcp**).

The *domain* field must consist of no more than 14 characters, in any combination of letters (upper and lower case), digits, hyphens (-), and underscores (_)

When *dname* is used to change a domain name, the host's password is removed. The administrator is prompted for a new password the next time RFS is started [*rfstart*(1M)]. The RFS domain name is set by **/etc/rc0** when going multi-user. The domain name is taken from **/etc/rcopts/DOMAIN.**

If *dname* is used with no options, it defaults to *dname* **-d**.

ERRORS

You cannot use the **-N** or **-D** options while RFS is running.

SEE ALSO

rfstart(1M).

NAME

du - summarize disk usage

SYNOPSIS

du [**-sar**] [names]

DESCRIPTION

du reports the number of 512-byte blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, the current directory is used.

The optional arguments are as follows:

-s causes only the grand total (for each of the specified *names*) to be given.

-a causes an output line to be generated for each file.

If neither **-s** or **-a** is specified, an output line is generated for each directory only.

-r will cause *du* to generate messages about directories that cannot be be read, files that cannot be opened, etc., rather than being silent (the default).

A file with two or more links is only counted once.

BUGS

If the **-a** option is not used, non-directories given as arguments are not listed.

If there are links between files in different directories where the directories are on separate branches of the file system hierarchy, *du* will count the excess files more than once.
Files with holes in them will get an incorrect block count.

NAME

dump - dump selected parts of an object file

SYNOPSIS

dump [options] files

DESCRIPTION

The *dump* command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

-a Dump the archive header of each member of each archive file argument.

-g Dump the global symbols in the symbol table of an archive.

-f Dump each file header.

-o Dump each optional header.

-h Dump section headers.

-s Dump section contents.

-r Dump relocation information.

-l Dump line number information.

-t Dump symbol table entries.

-z name Dump line number entries for the named function.

-c Dump the string table.

-L Interpret and print the contents of the *.lib* sections.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

-d number Dump the section number, *number*, or the range of sections starting at *number* and ending at the *number* specified by **+d**.

+d number Dump sections in the range either beginning with first section or beginning with section specified by **-d**.

-n name Dump information pertaining only to the named entity. This *modifier* applies to **-h**, **-s**, **-r**, **-l**, and **-t**.

-p Suppress printing of the headers.

-t index Dump only the indexed symbol table entry. The **-t** used in conjunction with **+t**, specifies a range of symbol table entries.

+t index Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **-t** option.

-u Underline the name of the file for emphasis.

-v Dump information in symbolic representation rather than numeric (for example, C_STATIC instead of **0X02**). This *modifier* can be used with all the above options except **-s** and **-o** options of *dump*.

-z name,number
Dump line number entry or range of line numbers starting at *number* for the named function.

+z number Dump line numbers starting at either function *name* or *number* specified by **-z**, up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **-z** option may be replaced by a blank.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

SEE ALSO

a.out(4), ar(4).

NAME

echo - echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\0*n*	where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

CAVEATS

When representing an 8-bit character by using the escape convention \0*n*, the *n* must **always** be preceded by the digit zero (0).

For example, typing: **echo** ´**WARNING:\07**´ will print the phrase **WARNING:** and sound the ''bell'' on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the ''\'' that precedes the ''07''.

For octal equivalents of each character, see ascii(5).

NAME

ed, red - text editor

SYNOPSIS

ed [**-s**] [**-p** string] [**-x**] [**-C**] [file]

red [**-s**] [**-p** string] [**-x**] [**-C**] [file]

DESCRIPTION

The *ed* program is the standard text editor. If the *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. If multiple *file* arguments are given, the % argument of the *e* command becomes useful.

-s Suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*. Also, see the **WARNING** section at the end of this manual page.

-p Allows the user to specify a prompt string.

-x Encryption option; when used, *ed* simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **-x** option. See *crypt*(1). *NOTE:* the standard CTIX distribution is the international version, which does not support encryption. (This is described also in the **WARNING** section at the end of this manual page.)

-C Encryption option; the same as the **-x** option, except that *ed* simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.

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

red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal

in **stty -tabs** or **stty tab3** mode [see *stty*(1)], the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. Note that when you are entering text into the file, this format is not in effect; instead, because of being in **stty -tabs** or **stty tab3** mode, tabs are expanded to every eighth column.

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

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

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (for example, *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([ ]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([ ]) (see 1.4 below).

c. $ (dollar sign), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (that is, delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, **[0-9]** is equivalent to **[0123456789]**. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); for example, **[ ]a-f]** matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly* *m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression \n matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A dollar sign ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^*entire RE*$ constrains the entire RE to match the entire line.

The null RE (for example, //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

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; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. 'x addresses the line marked with the mark name character *x*, which must be an ASCII lower-case letter (a-z). Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the

buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.

7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (respectively 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; for example, **-5** is understood to mean **.-5**.

9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -.) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.

10. For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see Rules 5 and 6, above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, 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; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

C

Same as the **X** command, except that *ed* assumes all text read in for the **e** and **r** commands is encrypted unless a null key is typed in.

(.,.)d

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

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). If **%** is given in place of a file name, the next name on the command line argument list is used. The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell [*sh*(1)] command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *f*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE*/*command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE*/

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be an ASCII lower-case letter (**a**-**z**). The address *'x* then addresses this line; . is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (for example, *tab, backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done; however, see *DIAGNOSTICS*, below.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(**$**)**r** *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. 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 set to the last line read in. If *file* is replaced by **!**, the rest of the line is taken to be a shell [*sh*(1)] command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)**s**/*RE*/*replacement*/ or
(.,.)**s**/*RE*/*replacement*/**g** or
(.,.)**s**/*RE*/*replacement*/**n** n = 1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* 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. If a number n appears after the command, only the n th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more

general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*RE*/*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 that does *not* match the RE.

(1,$)V/*RE*/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting [see *umask*(1)] dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell [*sh*(1)]

command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

x

A key is prompted for, and it is used in subsequent **e**, **r**, and **w** commands to decrypt and encrypt text using the *crypt*(1) algorithm. An educated guess is made to determine whether text read in for the **e** and **r** commands is encrypted. A null key turns off encryption. Subsequent *e*, *r*, and *w* commands will use this key to encrypt or decrypt the text [see *crypt*(1)]. An explicitly empty key turns off encryption. Also, see the **-x** option of *ed*. Due to export restrictions, encryption features are not available in the standard CTIX distribution.

(**$**)**=**

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 the CTIX system shell [*sh*(1)] to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(**.+1**)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, and 64 characters in the pathname of a file (counting slashes). The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters.

If a file is not terminated by a new-line character, **ed** adds one and outputs a message explaining what it did.

If the closing delimiter of a RE or of a replacement string (for example, /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

$TMPDIR	if this environmental variable is not null, its value is used in place of /usr/tmp as the directory name for the temporary work file.
/usr/tmp	default directory for temporary work file.
/tmp	if the environmetal variable **TMPDIR** does not exist or is null, and if **/usr/tmp** does not exist, then **/tmp** is used as the directory name for the temporary work file.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file.
	(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands. It prints **?** and allows one to continue editing. A second *e* or *q* command at this point will take effect. The **-s** command-line option inhibits this feature.

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1), fspec(4), regexp(5).

BUGS

A *!* command cannot be subject to a *g* or a *v* command.

The *!* command and the **!** escape from the *e*, *r*, and *w* commands cannot be used if the editor is invoked from a restricted shell [see *sh*(1)].

The sequence \n in a RE does not match a new-line character.

If the editor input is coming from a command file (for example, ed file < ed-cmd-file), the editor will exit at the first failure.

WARNINGS

Due to export restrictions, encryption features are not available in the standard CTIX distribution.

The - option, although supported in this release for upward compatibility, will no longer be supported in the next major release of the system. Convert shell scripts that use the - option to use the -s option, instead.

NAME

edit - text editor (variant of ex for casual users)

SYNOPSIS

edit [**-r**] [**-x**] [**-C**] *name...*

DESCRIPTION

The *edit* facility is a variant of the text editor *ex* recommended for new or casual users who want to use a command-oriented editor. It operates precisely as *ex*(1) with the following options automatically set:

novice	ON
report	ON
showmode	ON
magic	OFF

These options can be turned on or off via the **set** command in *ex*(1).

-r Recover file after an editor or system crash.

-x Encryption option; when used the file will be encrypted as it is being written and will require an encryption key to be read [see *crypt*(1)]. *NOTE:* the standard CTIX distribution is the international version, which does not support encryption. (This is described also in the **WARNING** section at the end of this manual page.)

-C Encryption option; the same as **-x** except that *edit* assumes files are encrypted.

The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

To edit the contents of an existing file you begin with the command "edit name" to the shell. *edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, you also begin with the command **edit** with a filename: **edit** *name*; the editor will tell you it is a **[New File]**.

The *edit* command prompt is the colon (:), which you should see after starting the editor. If you are editing an existing file, you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). When you start editing, *edit* makes the last line of the file the current line. Most commands to *edit* use the current line if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and type carriage return (as you should after all *edit* commands), the current line will be printed. If you **delete** (**d**) the

current line, *edit* will print the new current line, which is usually the next line in the file. If you **delete** the last line, the new last line becomes the current one.

If you start with an empty file or want to add some new lines, the **append** (**a**) command can be used. After you execute this command (typing a carriage return after the word **append**), *edit* will read lines from your terminal until you type a line consisting of just a dot (.); it places these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append**, but places the lines you type before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you execute the command **1**, *edit* will type the first line of the buffer. If you then execute the command **d**, *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command: **s**/*old*/*new*/ where *old* is the string of characters you want to replace and *new* is the string of characters you want to replace *old* with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say **[Modified]** if you have changed the buffer. After modifying a file, you can save the contents of the file by executing a **write** (**w**) command. You can leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will receive the message **No write since last change (:quit! overrides)**, and *edit* will wait for another command. If you do not want to write the buffer out, issue the **quit** command followed by an exclamation point (**q!**). The buffer is then irretrievably discarded and you return to the shell.

By using the **d** and **a** commands and giving line numbers to see lines in the file, you can make any changes you want. You should learn at least a few more things, however, if you will use *edit* more than a few times.

The **change** (**c**) command changes the current line to a sequence of lines you supply (as in **append**, you type lines up to a line consisting of only a dot (.). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, that is, **3,5c**. You can print lines this way too: **1,23p** prints the first 23 lines of the file.

The **undo** (**u**) command reverses the effect of the last command you executed that changed the buffer. Thus if you execute a **substitute** command that does not do what you want, type **u** and the old contents of the line will be restored. You can also **undo** an **undo** command. *edit* will give you a warning message when a command affects more than one line of the buffer. Note that commands such as **write** and **quit** cannot be undone.

To look at the next line in the buffer, type carriage return. To look at a number of lines, type ^**D** (while holding down the control key, press **d**) rather than carriage return. This will show you a half-screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at nearby text by executing the **z** command. The current line will appear in the middle of the text displayed, and the last line displayed will become the current line; you can get back to the line where you were before you executed the **z** command by typing ''. The **z** command has other options: **z-** prints a screen of text (or 24 lines) ending where you are; **z+** prints the next screenful. If you want less than a screenful of lines, type **z.11** to display five lines before and five lines after the current line. (Typing **z.***n*, when *n* is an odd number, displays a total of *n* lines, centered about the current line; when *n* is an even number, it displays *n*-1 lines, so that the lines displayed are centered around the current line.) You can give counts after other commands; for example, you can delete 5 lines starting with the current line with the command **d5** .

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form */text/* to search forward for *text* or ?*text*? to search backward for *text*. If a search reaches the end of the file without finding *text*, it wraps around and continues to search back to the line where you are. A useful feature here is a search of the form */^text/* which searches for *text* at the beginning of a line. Similarly */text*$/ searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has the symbolic name dot (.); this is most useful in a range of lines as in **.,$p** which prints the current line plus the rest of the lines in the file. To move to the last line in the file, you can refer to it by its symbolic name $. Thus the command **$d** deletes the last line in the file, no matter what the current line is. Arithmetic with line references is also possible. Thus the line **$-5** is the fifth before the last and **.+20** is 20 lines after the current line.

You can find out the current line by typing **.=**. This is useful if you want to move or copy a section of text within a file or between files. Find the first and last line numbers you want to copy or move. To move lines 10 through 20, type **10,20d a** to delete these lines from the file and place them in a buffer named **a**.

edit has 26 such buffers named **a** through **z**. To put the contents of buffer **a** after the current line, type **put a**. If you want to move or copy these lines to another file, execute an **edit** (**e**) command after copying the lines; following the **e** command with the name of the other file you want to edit, that is, **edit chapter2**. To copy lines without deleting them, use **yank** (**y**) in place of **d**. If the text you want to move or copy is all within one file, it is not necessary to use named buffers. For example, to move lines 10 through 20 to the end of the file, type **10,20m $**.

SEE ALSO

ed(1), ex(1), vi(1).

WARNING

Due to export restrictions, encryption features are not available in the standard CTIX distribution.

NAME

efl - extended FORTRAN language

SYNOPSIS

efl [options] [files]

DESCRIPTION

efl compiles a program written in the EFL language into clean FORTRAN on the standard output. *Efl* provides the C-like control constructs of *ratfor*(1):

statement grouping with braces.

decision-making:

if, if-else, and **select-case** (also known as **switch-case**);
while, for, FORTRAN **do, repeat**, and **repeat ... until** loops;
multi-level **break** and **next.**

EFL has C-like data structures, for example:

```
struct
        {
        integer flags(3)
        character(8) name
        long real coords(2)
        } table(100)
```

The language offers generic functions, assignment operators (+=, &=, etc.), and sequentially evaluated logical operators (**&&** and **||**). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) } )
```

EFL also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:
this is a comment.

translation of relational and logical operators:
>, >=, &, etc., become **.GT., .GE., .AND.**, etc.

return expression to caller from function:
return (*expression*)

defines:

define *name replacement*

includes:

include *file*

efl understands several option arguments: **-w** suppresses warning messages, **-#** suppresses comments in the generated program, and the default option **-C** causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on.

Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

SEE ALSO

cc(1), ratfor(1).

NAME

egrep - search a file for a pattern using full regular expressions

SYNOPSIS

egrep [options] full regular expression [file ...]

DESCRIPTION

The *egrep* (*expression grep*) command searches files for a pattern of characters and prints all lines that contain that pattern. *egrep* uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

egrep accepts full regular expressions as in *ed*(1), except for \(and \), with the addition of:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses () for grouping.

Be careful using the characters $, *, [, ^, |, (,), and \ in *full regular expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes ´...´.

The order of precedence of operators is [ ], then * ? +, then concatenation, then | and new-line.

If no files are specified, *egrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-i Ignore upper/lower case distinction during comparisons.

-l Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-v Print all lines except those that contain the pattern.

-e *special_expression*
Search for a *special expression* (*full regular expression* that begins with a -).

-f *file* Take the list of *full regular expressions* from *file*.

SEE ALSO

ed(1), fgrep(1), grep(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 (even if matches were found).

BUGS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h.**

NAME

enable, disable - enable/disable LP printers

SYNOPSIS

enable printers

disable [**-c**] [**-r** [reason]] printers

DESCRIPTION

The *enable* command activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers are reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

-c Cancel any requests that are currently printing on any of the designated printers.

-r[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a reason, then a default reason is used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).
S/Series CTIX Administrator's Guide.

NAME

enpstart - configure Ethernet processor

SYNOPSIS

/etc/enpstart addr1 [addr2 ...]

DESCRIPTION

enpstart loads one or more Ethernet processors with a download image, starts them running, and sets the network address to be associated with the interface. If more than one address is specified, each should correspond to one Ethernet processor: the first address is assigned to unit 0, and so on. Units are ordered with VME boards first (by slot number), then CT Combo boards (by slot).

Each network address *addrN* must be included in /etc/hosts, or available via the name server, or specified in Internet dot notation. If the special name " " is given for one or more interfaces, they are skipped over in the initialization process.

FILES

/etc/hosts
/etc/enp/*

SEE ALSO

hosts(4) and *inet*(7), for Internet dot notation.

REQUIREMENTS

Ethernet processors must be correctly installed on the system.

DIAGNOSTICS

en*N* doesn't respond to initialization, skipping...
A problem was detected with Ethernet processor unit N.

NAME

env - set environment for command execution

SYNOPSIS

env [-] [name=value] ... [command args]

DESCRIPTION

The *env* command obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The - flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1), exec(2), profile(4), environ(5).

NAME

eqn, neqn, checkeq - format mathematical text for nroff or troff

SYNOPSIS

eqn [**-dxy**] [**-pn**] [**-sn**] [**-fn**] [files]

neqn [**-dxy**] [**-pn**] [**-sn**] [**-fn**] [files]

checkeq [files]

DESCRIPTION

eqn is a *troff*(1) preprocessor for typesetting mathematical text on a phototypesetter, while *neqn* is used for the same purpose with *nroff* on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent.

If no files are specified (or if - is specified as the last argument), these programs read 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 designate two characters as *delimiters*; subsequent text between delimiters is then 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 the same character; the dollar sign is often used as such a delimiter. 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, new-lines, braces, double quotes, tildes, and circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub j* makes x_j, *a sub k sup 2* produces a_k^2, while $e^{x^2+y^2}$ is made with *e sup {x sup 2 + y sup 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: $\lim_{n\to\infty}\sum_{0}^{n}x_i$ is made with

lim from {n -> inf } sum from 0 to n x sub i. Left and right brackets, braces, etc., of the right height are made with **left** and **right**:

left [x sup 2 + y sup 2 over alpha right] ~=~ 1

produces

$$\left[x^2+\frac{y^2}{\alpha}\right] = 1.$$

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). A **left** *thing* need not have a matching **right** *thing*.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**:

pile {a above b above c}

produces

a
b.
c

Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but 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

x_i 1
y_2 2

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 $\bar{\dot{x}}=f(t)$,

y dotdot bar ~=~ n under is $\bar{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x}=\overleftrightarrow{y}$.

Point sizes and fonts can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Point sizes 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 points 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* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** (∑), **int** (∫), **inf** (∞), and shorthands such as >= (≥), != (≠), and -> (→) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or GAMMA (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as \(**dd** (‡) and \(**bs** (\) may 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*(1) when all else fails. Full details are given in the manual cited below.

SEE ALSO

cw(1), mm(1), mmt(1), nroff(1), tbl(1), troff(1), eqnchar(5), mm(5), mv(5).

BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.

See also *BUGS* under *troff*(1).

NAME

errdead - extract error records and status information from dump

SYNOPSIS

/etc/errdead [**-a** [**e**] [**f**]] [dumpfile] [namelist]

DESCRIPTION

When hardware errors are detected by the system, an error record containing information pertinent to the error is generated. If the error-logging demon *errdemon*(1M) is not active, or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. The *errdead* command examines a system dump (or memory), extracts such error records, and passes them to *errpt*(1M) for analysis.

The following are options to *errdead*:

-a Instead of passing extracted records to *errpt*(1M), append them to **/usr/adm/errfile**, provided that the dump corresponds to the namelist and that the dump is newer than the error file.

-e Valid only if **-a** is also specified. Invoke *errdemon*(1M) when finished. Normally, *errdemon*(1M) is invoked in the *rc2* or *rc3* script through the *errlog* start procedure [see *rc2*(1M)].

-f Valid only if **-a** is also specified. Write extracted records even if the dump is older than the error file.

dumpfile specifies the file (or memory) to be examined; if *dumpfile* is not specified, *errdead* looks for a dump area by scanning the available disks in the same order as does the bootstrap ROM.

namelist specifies the system namelist; if the namelist is not specified, **/etc/lddrv/prev.unix.exec** is used.

FILES

/etc/lddrv/prev.unix.exec	system namelist
/usr/bin/errpt	analysis program
/usr/tmp/errXXXXXX	temporary file
/usr/adm/errfile	repository for error records
/etc/log/confile	console file

DIAGNOSTICS

Diagnostics can come from *errdead* or *errpt*. In either case, they are intended to be self-explanatory.

SEE ALSO

errdemon(1M), errpt(1M).

NAME

errdemon - error-logging demon

SYNOPSIS

/usr/lib/errdemon [**-n**] [**-c** file] [file]

DESCRIPTION

The error logging demon *errdemon* collects error records from the operating system by reading the special file **/dev/error** and places them in *file*. If *file* is not specified when the demon is activated, **/usr/adm/errfile** is used. Note that *file* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon*; that responsibility is left to *errpt*(1M). *errdemon* can also extract console records; the **-n** option disables this, thus forcing all console reports to stay in a circular buffer in the kernel. The **-c** option allows specifying a console file. The default console file is **/etc/log/confile**. The error-logging demon is terminated by sending it a software kill signal [see *kill*(1)]. Only the superuser may start the demon, and only one demon may be active at any time.

FILES

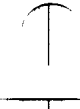
/dev/error	source of error records
/usr/adm/errfile	repository for error records
/etc/log/confile	console records
/dev/console	

SEE ALSO

errpt(1M), errstop(1M), kill(1), err(7).

DIAGNOSTICS

The diagnostics produced by *errdemon* are intended to be self-explanatory.



NAME

errpt - process a report of logged errors

SYNOPSIS

errpt [options] [files]

DESCRIPTION

The *errpt* command processes data collected by the error logging mechanism *errdemon*(1M) and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, *errpt* attempts to use **/usr/adm/errfile** as *file*.

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unabled to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times *errpt* has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped, and any time changes [through *date*(1)] that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report can be limited to certain records in the following ways:

-s *date* Ignore all records posted earlier than *date*, where *date* has the form *mmddhhmmyy*, consistent in meaning with the *date*(1) command.

-e *date* Ignore all records posted later than *date*, whose form is as described above.

-a Produce a detailed report that includes all error types.

-d *devlist* A detailed report is limited to data about devices given in *devlist*, where *devlist* can be one of two forms: a list of device identifiers separated from one another by a comma, or a list of device identifiers enclosed in double quotes and separated from one another by a comma and/or more spaces. *Errpt* is familiar with the block devices **gd0** to **gd15**. Additional identifiers are **int** (stray interrupts), **mem** (memory parity/ECC errors and ECC correction), **qic0** (quarter-inch tape), **tape0** (half-inch tape), **sa0** (SCSI tape), and **tty** (serial asynchronous terminals).

-p *n* Limit the size of a detailed report to *n* pages.

-f In a detailed report, limit the reporting of block device errors to unrecovered errors.

Logical blocks in the filesystem are 1024 bytes. Physical sector numbers are 512-byte blocks.

FILES

/usr/adm/errfile default error file

SEE ALSO

date(1), errdead(1M), errdemon(1M), errfile(4).

NAME

errstop - terminate the error-logging demon

SYNOPSIS

/etc/errstop [namelist]

DESCRIPTION

The error-logging demon *errdemon*(1M) is terminated by using *errstop*. This is accomplished by executing *ps*(1) to determine the demon's identity and then sending the demon a software kill signal [see *signal*(2)]; **/unix** is used as the system namelist if none is specified. Only the super-user can use *errstop*.

FILES

/unix default system namelist

SEE ALSO

errdemon(1M), ps(1), kill(2). signal(2).

DIAGNOSTICS

The diagnostics produced by *errstop* are intended to be self-explanatory.

NAME

ex - text editor

SYNOPSIS

ex [**-s**] [**-v**] [**-t** tag] [**-r** file] [**-L**] [**-R**] [**-x**]
[**-C**] [**-c** command] file ...

DESCRIPTION

ex is the root of a family of editors: *ex* and *vi*. *ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have a CRT terminal, you may want to use a display based editor; in this case see *vi*(1), a command which focuses on the display-editing portion of *ex*.

For ed Users

If you have used *ed*(1) you will find that, in addition to providing all of the *ed*(1) commands, *ex* has a number of additional features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the *ex* editor uses far more of the capabilities of terminals than *ed*(1) does, and uses the terminal capability data base [see *terminfo*(4)] and the type of the terminal you are using from the environmental variable TERM to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1).

ex contains a number of features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Typing **^D** (control-d) causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just typing return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files, unless you edited them, so that you do not accidentally overwrite a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the

telephone, you can use the editor **recover** command (or **-r** *file* option) to retrieve your work. This will get you back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

The editor has a group of buffers whose names are the ASCII lower-case letters (**a-z**). You can place text in these named buffers where it is available to be inserted elsewhere in the file. The contents of these buffers remain available when you begin editing a new file using the **edit** (**e**) command.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition, there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore the case of letters in searches and substitutions. *ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

ex has a set of options which you can set to tailor it to your liking. One option which is very useful is the **autoindent** option that allows the editor to supply leading white space to align text automatically. You can then use **^D** as a backtab and space or tab to move forward to align new code easily.

Miscellaneous useful features include an intelligent **join** (**j**) command that supplies white space between joined lines automatically, commands "**<**" and "**>**" which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*(1).

Invocation Options

The following invocation options are interpreted by *ex* (previously documented options are discussed in the NOTES section at the end of this manual page):

-s Suppress all interactive-user feedback. This is useful in processing editor scripts.

-v Invoke *vi*.

-t *tag* Edit the file containing the *tag* and position the editor at its definition.

-r *file* Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)

-L List the names of all files saved as the result of an editor or system crash.

-R **Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.

-x Encryption option; when used, *ex* simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **-x** option. See *crypt*(1). *NOTE:* the standard CTIX distribution is the international version, which does not support encryption. (This is described also in the **WARNING** section at the end of this manual page.)

-C Encryption option; the same as the **-x** option, except that *ex* simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.

-c *command* Begin editing by executing the specified editor *command* (usually a search or positioning command).

-l **LISP** mode; indents appropriately for lisp code, the () {} [[and]] commands in *vi* are modified to have meaning for *lisp*.

The *file* argument indicates one or more files to be edited.

ex States

Command Normal and initial state. Input prompted for by :. Your line kill character cancels a partial command.

Insert Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert state normally is terminated by a line having only "." on it, or, abnormally, with an interrupt.

Visual Entered by typing **vi**; terminated by typing **Q** or ^\ (control-\).

ex Command Names and Abbreviations

abbrev	**ab**	map		set	**se**
append	**a**	mark	**ma**	shell	**sh**
args	**ar**	move	**m**	source	**so**
change	**c**	next	**n**	substitute	**s**
copy	**co**	number	**nu**	unabbrev	**unab**
delete	**d**	preserve	**pre**	undo	**u**
edit	**e**	print	**p**	unmap	**unm**
file	**f**	put	**pu**	version	**ve**
global	**g**	quit	**q**	visual	**vi**
insert	**i**	read	**r**	write	**w**
join	**j**	recover	**rec**	xit	**x**
list	**l**	rewind	**rew**	yank	**ya**

ex Commands

shell escape	**!**
forced encryption	**C**
heuristic encryption	**X**
lshift	**<**
print next	CR
resubst	**&**
rshift	**>**
scroll	**^D**
window	**z**

ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
-	previous	*'x*	marked with *x*
+n	*n* forward	*''*	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment variable
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option *x*

set no*x*	disable option *x*
set *x*=*val*	give value *val* to option *x*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

Most useful options and their abbreviations

autoindent	**ai**	supply indent
autowrite	**aw**	write before changing files
directory		pathname of directory for temporary work files
ignorecase	**ic**	ignore case of letters in scanning
list		print ^**I** for tab, $ at end
magic		treat **.** **[** ***** special in patterns
modelines		first five lines and last five lines executed as *vi*/*ex* commands if they are of the form **ex:***command*: or **vi:***command*:
number	**nu**	number lines
paragraphs	**para**	macro names that start paragraphs
redraw		simulate smart terminal
report		informs you if the number of lines modified by the last command is greater than the value of the **report** variable
scroll		command mode lines
sections	**sect**	macro names that start sections
shiftwidth	**sw**	for **<** **>**, and input ^**D**
showmatch	**sm**	to) and } as typed
showmode	**smd**	show insert mode in *vi*
slowopen	**slow**	stop updates during insert
term		specifies to **vi** the type of terminal being used (the default is the value of the environmental variable **TERM**)
window		visual mode lines
wrapmargin	**wm**	automatic line splitting
wrapscan	**ws**	search around end (or beginning) of buffer

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any character in *str*
[^*str*]	any character not in *str*
[*x*-*y*]	any character between *x* and *y*
*	any number of preceding characters

FILES

/usr/lib/exstrings	error messages
/usr/lib/exrecover	recover command
/usr/lib/expreserve	preserve command
/usr/lib/terminfo/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve/*login*	preservation directory (where *login* is the user's login)

SEE ALSO

awk(1), ed(1), edit(1), grep(1), sed(1), vi(1), curses(3X), term(4), terminfo(4).

NOTES

Several options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard [see *intro*(1)]. The - option has been replaced by -s, a -r option that is not followed with an option-argument has been replaced by -L, and +*command* has been replaced by -c *command*.

WARNING

Due to export restrictions, encryption features are not available in the standard CTIX distribution.

BUGS

The z command prints the number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line -s option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

expand, unexpand - expand tabs to spaces, and vice versa

SYNOPSIS

expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]
unexpand [**-a**] [file ...]

DESCRIPTION

expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default only leading blanks and tabs are reconverted to maximal strings of tabs. If the **-a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

SEE ALSO

newform(1).

NAME

expr - evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr*
: returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr*
: returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
: returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, - } *expr*
: addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
: multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*
: The matching operator : compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of *ed*(1), except that all patterns are "anchored" (that is, begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

EXAMPLES

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

   adds 1 to the shell variable a.

2. ```
   # 'For $a equal to either "/usr/abc/file" or
   # just "file"'
   expr $a : '.*/\(.*\)' \| $a
   ```

   returns the last segment of a path name (that is, file). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. ```
   # A better representation of example 2.
   expr //$a : '.*/\(.*\)'
   ```

   The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. `expr $VAR : '.*'`

   returns the number of characters in $VAR.

SEE ALSO

ed(1), sh(1).

DIAGNOSTICS

As a side effect of expression evaluation, *expr* returns the following exit values:

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

syntax error
for operator/operand errors

non-numeric argument
if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

NAME

extproc - turn external processing on or off

SYNOPSIS

/etc/riop/extproc device [0 | 1]

DESCRIPTION

Extproc enables or disables external echo and output processing for an individual port of an RIOP. *Device* must be an RIOP tty device. If the argument following *device* is 1, then external processing is enabled. If this argument is 0 then external processing is disabled. This takes effect for the port during its next closed-to-open transition. By default, external echo and output processing is enabled.

SEE ALSO

riopcfg(1M).

NAME

factor - obtain the prime factors of a number

SYNOPSIS

factor [integer]

DESCRIPTION

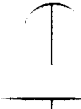
When you use *factor* without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. If it encounters a zero or any non-numeric character, *factor* exits.

If you invoke *factor* with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$; *factor* takes this time when n is prime or the square of a prime.

DIAGNOSTICS

The error message "Ouch" appears for input out of range or for garbage input.



NAME

ff - list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

The *ff* command reads the i-list and directories of the *special* file, assuming it is a file system. I-node data is saved for files which match the selection criteria. Output consists of the path name for each saved i-node, plus other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *-n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

-I	Do not print the i-node number after each path name.
-l	Generate a supplementary list of all path names for multiply-linked files.
-p *prefix*	The specified *prefix* will be added to each generated path name. The default is . (dot).
-s	Print the file size, in bytes, after each path name.
-u	Print the owner's login name after each path name.
-a *n*	Select if the i-node has been accessed in *n* days.
-m *n*	Select if the i-node has been modified in *n* days.
-c *n*	Select if the i-node has been changed in *n* days.
-n *file*	Select if the i-node has been modified more recently than the argument *file*.
-i *i-node-list*	Generate names for only those i-nodes specified in *i-node-list*.

SEE ALSO

find(1), ncheck(1M).

BUGS

If the -l option is not specified, only a single path name out of all possible ones is generated for a multiply-linked i-node. If -l is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

NAME

fgrep - search a file for a character string

SYNOPSIS

fgrep [options] string [file ...]

DESCRIPTION

The *fgrep* (fast *grep*) command seaches files for a character string and prints all lines that contain that string. *fgrep* is different from *grep(1)* and *egrep(1)* because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters $, *, [, ^, |, (,), and \ are interpreted literally by *fgrep*, that is, *fgrep* does not recognize full regular expressions as does *egrep*. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes ´...´.

If no files are specified, *fgrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-i Ignore upper/lower case distinction during comparisons.

-l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-v Print all lines except those that contain the pattern.

-x Print only lines matched entirely.

-e *special_string*
Search for a *special string* (*string* begins with a -).

-f*file* Take the list of *strings* from *file*.

SEE ALSO

ed(1), egrep(1), grep(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 (even if matches were found).

BUGS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

NAME

file - determine file type

SYNOPSIS

file [**-c**] [**-f** ffile] [**-m** mfile] arg ...

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. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0.

-c The -c option causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **-c**.

-f If the **-f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

-m The **-m** option instructs *file* to use an alternate magic file.

file uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

FILES

/etc/magic

SEE ALSO

filehdr(4).

NAME

finc - fast incremental backup

SYNOPSIS

/etc/finc [selection-criteria] file-system raw-tape

DESCRIPTION

The *finc* command selectively copies the input *file-system* to the output *raw-tape*. To be cautious, mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by *labelit*. The selection is controlled by the *selection-criteria*, accepting only those i-nodes/files for whom the conditions are true.

It is recommended that production of a *finc* tape be preceded by the *ff* command, and the output of *ff* be saved as an index of the tape's contents. Files on a *finc* tape may be recovered with the *frec* command.

The argument *n* in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *-n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hours.

-a *n* True if the file has been accessed in *n* days.

-m *n* True if the file has been modified in *n* days.

-c *n* True if the i-node has been changed in *n* days.

-n *file* True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a tape consisting of all files from file-system **/usr** modified in the last 48 hours:

```
finc -m -2 /dev/dsk/c0d0s3 /dev/rmt/c0d0
```

SEE ALSO

cpio(1), ff(1M), frec(1M), labelit(1M).

NAME

find - find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

The *find* command recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, one or more path names) 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*. Valid expressions are:

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

[**-perm**] *-onum* — True if the file permission flags exactly match the octal number *onum* [see *chmod*(1)]. If *onum* is prefixed by a minus sign, only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.

-type *c* — True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, FIFO (named pipe), or plain file respectively.

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

-user *uname* — True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

-group *gname* — True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

-size *n*[c] — True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.

-atime *n* — True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.

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

-ctime *n* — True if the file has been changed in *n* days.

-exec *cmd*	True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.
-ok *cmd*	Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.
-print	Always true; causes the current path name to be printed.
-cpio *device*	Always true; write the current file on *device* in *cpio* (1) format (5120-byte records).
-newer *file*	True if the current file has been modified more recently than the argument *file*.
-inum *n*	True if the current file is inode number *n*.
-depth	Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.
-mount	Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.
-local	True if the file physically resides on the local system.
(*expression*)	True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

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

1. The negation of a primary (**!** is the unary *not* operator).
2. Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
3. 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

FIND(1)

SEE ALSO

chmod(1), cpio(1), sh(1), test(1), stat(2), umask(2), fs(4).

NAME

finger - user information lookup program

SYNOPSIS

finger [options] [name ...]

DESCRIPTION

The *finger* program displays information about users on local and remote machines. Without arguments, *finger* lists the login name, full name (as specified in the fifth field of **/etc/passwd**), terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each logged in user on the local system. (Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.)

The *finger* program generates a longer, more detailed format of user information if at least one of the following arguments is supplied:

1. A user name or a list of user names. A user name can be a login name (first field of **/etc/passwd**) or a first or last name (fifth field of **/etc/passwd**).
2. The **-l** option.

The longer format is multi-line, and includes all the information described above, as well as the user's home directory and login shell, any plan specified in the user's **$HOME/.plan**, and any project specified in the user's **$HOME/.project**.

To use *finger* to lookup users on a remote machine, specify the user as *user@host*. A list of users on the same remote host can be enclosed in double quotes; for example,

```
finger "user1 user2 user3"@host
```

If no user names are supplied (argument to *finger* is *@host*), standard, rather than long, format listing is provided.

The following options are recognized by *finger*:

-m Match arguments only on login name.

-l Force long (rather than standard) output format.

-p Suppress printing of the **.plan** files.

-s Force standard (rather than long) output format.

FILES

/etc/utmp	who file
/etc/passwd	for users names, offices, ...
/usr/adm/lastlog	last login times
$HOME/.plan	plans
$HOME/.project	projects

SEE ALSO

who(1)

BUGS

Only the first line of the **.project** file is printed.

There is no way to pass arguments to the remote machine as *finger* uses an internet standard port.

NAME

fingerd - remote user information server

SYNOPSIS

/etc/fingerd

DESCRIPTION

The *fingerd* server provides a network interface to the *finger*(1) program (or, on some other systems, the *name* program). This interface allows *finger* to display information about remote users.

The *fingerd* server listens for TCP connections on the *finger* port [see *services*(4)]. For each connection, *fingerd* reads a single input line (terminated by a <CRLF>), passes the line to *finger*, and copies the output of *finger* to the user on the client machine.

The *fingerd* service is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

SEE ALSO

finger(1), inetd(1M), inetd.conf(4), services(4).
RFC 742.

WARNING

Connecting to *fingerd* using TELNET [see *telnet*(1)] can have unpredictable consequences and is not recommended.

NAME

fold - fold long lines for finite width output device

SYNOPSIS

fold [-columns] [file ...]

DESCRIPTION

fold produces a folded version of its input, inserting newlines so that none of its output lines is wider than *columns*. If *columns* is omitted, folding is done at 80 columns.

If tabs are present in the input, *columns* should be a multiple of eight.

SEE ALSO

expand(1).

WARNING

Overstriking can be spoiled by folding.

NAME

frec - recover files from a backup tape

SYNOPSIS

/etc/frec [**-p** path] [**-f** reqfile] raw_tape i_number : name ...

DESCRIPTION

The *frec* command recovers files from the specified *raw_tape* backup tape written by *volcopy*(1M) or *finc*(1M), given their *i_numbers*. The data for each recovery request will be written into the file given by *name*.

The **-p** option allows you to specify a default prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified, that is, that do not begin with / or ./. If any directories are missing in the paths of recovery *names* they will be created.

-p *path* Specifies a prefixing *path* to be used to fully qualify any names that do not start with / or ./.

-f *reqfile* Specifies a file which contains recovery requests. The format is i_number:newname, one per line.

EXAMPLES

To recover a file, i-number 1216 when backed-up, into a file named *junk* in your current working directory:

```
frec /dev/rmt/c0d0 1216:junk
```

To recover files with i_numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/joe/a.c**:

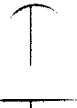
```
frec -p /usr/src/cmd /dev/rmt/c0d0 14156:a 1232:b
3141:/usr/joe/a.c
```

SEE ALSO

cpio(1), ff(1M), finc(1M), labelit(1M).

BUGS

While paving a path (that is, creating the intermediate directories contained in a pathname) *frec* can only recover inode fields for those directories contained on the tape and requested for recovery.



NAME

fsck, dfsck - check and repair file systems

SYNOPSIS

/etc/fsck [**-y**] [**-n**] [**-sc:s**] [**-s**] [**-Sc:s**] [**-S**] [**-t** file] [**-q**] [**-D**] [**-f**] [**-p**] [**-bB**] [**-O**] [**-M**] [file-systems]

/etc/dfsck [options1] filsys1 ... - [options2] filsys2 ...

DESCRIPTION

Fsck

The *fsck* command audits and interactively repairs inconsistent conditions for CTIX file systems. If the file system is consistent, the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent, the operator is prompted for concurrence before each correction is attempted. It should be noted that some corrective actions result in some loss of data. The amount and severity of data lost can be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission, *fsck* defaults to an **-n** action. Upon completion, *fsck* reports the number of used and free 512-byte blocks and the number of files in the file system.

Modifying a mounted (**root**) file system requires special precautions by *fsck*, because a single *sync (2)* undoes all of *fsck's* repair work. To prevent this, *fsck* performs a *uadmin*(A_REMOUNT,0,0) [see *uadmin*(2)]. The system call forces CTIX to reread the super-block from the disk. If there is extensive damage to the mounted file system, *fsck* reboots CTIX.

The following options are interpreted by *fsck*:

-y Assume a yes response to all *fsck* prompts.

-n Assume a no response to all questions asked by *fsck* prompts; do not open the file system for writing.

-sc:*s*

-s Ignore the actual free list or bit map and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent.

If *c*:*s* is given on a standard file system, the free list is organized with *c* blocks-per-cylinder and *s* blocks skipped. If *c*:*s* is omitted, the values originally specified to *mkfs* are used. If these values were not specified, the value *400*:*7* is used.

-S*c:s*

-S Conditionally reconstruct the free list or bit map. This option is like -s, described above, except that the free list or bit map is rebuilt only if no discrepancies are discovered in the file system. Using -S forces a no response to *fsck* prompts. This option is useful for forcing free list or bit map reorganization on uncontaminated file systems.

-t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the -t option is specified, the file named in the next argument is used as the scratch file, if needed. Without the -t flag, *fsck* prompts for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

-q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced FIFOs are silently removed. If *fsck* requires it, counts in the super-block are automatically fixed and the free list or bit map is salvaged.

-D Directories are checked for consistency. This is useful after system crashes. The following inconsistencies are sought:

- Entries with null names but nonzero i-numbers.
- Entries that are not padded to full size with nulls.
- Invalid . and .. entries.
- Names that contain a slash (/).
- Final blocks that are not cleared past end-of-file.

-f Fast check. Check block and sizes (Phase 1) and check the free list or bit map (Phase 5). The free list or bit map is reconstructed (Phase 6) if it is necessary.

-p Preen file systems only; intended for autoboot. The *fsck* program does not prompt for operator input; instead, it applies standard fixes whenever the fix doesn't involve loss of data. Only the following problems are subject to this kind of fix:

- Unreferenced i-nodes.
- Link counts in i-nodes too large.
- Missing blocks in the free list.
- Blocks in the free list also in files.
- Counts in the super-block wrong.

Any problem not of this type causes *fsck* to terminate with an error status. The startup script that runs *fsck* (normally **/etc/bcheckrc**) can specify the -p option to *fsck* and make a normal boot contingent upon a normal *fsck* return status.

-b or **-B**

If the file system being checked is the **root** file system and modifications have been made, resync the file system, or reboot if necessary.

-M Convert file system to new bit map free list format.

-O Convert file system to old free list format.

Both **-M** and **-O** imply **-s**.

If no *file-systems* are specified, *fsck* reads a list of default file systems from the file **/etc/checklist**.

Inconsistencies are checked as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   - Incorrect number of blocks.
   - Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   - File pointing to unallocated i-node.
   - I-node number out of range.
8. Super-block checks:
   - More than 65536 i-nodes.
   - More blocks for i-nodes than exist in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user is notified if the file or directory is empty or not. If it is empty, *fsck* silently removes them. The *fsck* program forces the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. The **lost+found** directory is normally created by running *mklost+found*(1M) just after the file system is created with *mkfs*(1M).

Checking the raw device is almost always faster and should be used with everything but the **root** file system.

Dfsck

The *dfsck* program allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A dash (-) is the separator between the file system groups.

The *dfsck* program permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* prints the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Do not use *dfsck* to check the **root** file system.

FILES

/etc/checklist	default list of file systems to check
/etc/checkall	optimizing *dfsck* shell file

SEE ALSO

clri(1M), init(1M), mklost+found(1M), uadmin(2), ncheck(1M), checklist(4), fs(4).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

If **-p** was specified and preening was inadequate, a nonzero status is returned.

NOTES

Always unmount file systems before running *fsck* except in the case of the **root** file system.

The *block* device must be used with mounted file systems; thus, the **root** file system must always be specified as the block device.

The maintenance tape can be used to check the normal **root** file system as a raw device, unmounted. (In this case, the **root** file system is on the RAM disk.)

The *fsck* program determines the file system type (1K or 4K) on its own.

BUGS

I-node numbers for . and .. in each directory should be checked for validity.

The *fsck* program does not know how to create a **lost+found** directory.

NAME

fsdb - file system debugger

SYNOPSIS

/etc/fsdb special [-]

DESCRIPTION

fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

fsdb contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional - argument or by the use of the **O** symbol. (*fsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to byte address
d	directory slot offset
+,-,*,/	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
F	buffer status

X	hexadecimal or octal address flip-flop (default is hexadecimal)
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed.

The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes
s or **S**	print as superblock
x	print as hexadecimal words
h	print as hexadecimal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character

representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 - 12)
at	access time
mt	modification time
maj	major device number
min	minor device number
si	#inodes field in superblock
sf	#blks field in superblock
sd0	s_dinfo[0] in superblock
sd1	s_dinfo[1] in superblock
=BS	set a blank superblock with file system type 1K and a magic number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.
2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.

d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current i-node as directory entries.
512.ps	prints the superblock

SEE ALSO

fsck(1M), dir(4), fs(4).

NAME

fsize - report file size

SYNOPSIS

/usr/local/bin/fsize

/usr/local/bin/fsize -s

/usr/local/bin/fsize -m [-i] [-c] [-q] [-v] [-b#] [-f#] [-u]

DESCRIPTION

The *fsize* command has three uses, which correspond to the three forms of the command:

1. Read the standard input to obtain a list of path names and report the total size of the combined named files (size is always reported in 512-byte blocks). If the files are nonexistent, *fsize* reports 0. This form of the command takes no arguments.

2. Read the standard input to obtain a list of path names and report the size of each file individually, one per line (size is always reported in 512-byte blocks). This form of the command takes **-s** as a single required argument.

3. Read the standard input to obtain a list of path names and size requirements for each named file (for example, the output of **fsize -s**) and report statistics on a per-file system basis. This form of the command takes **-m** as a required argument and the following additional options:

   -i Report number of i-nodes needed.

   -c Adjust space requirements if there is an existing file with the same name as a file included in the standard input.

   -q If file system is not large enough to contain the named files, print a message and exit with a non-zero status.

   -v Print statistics on each file system.

   -b# Fudge factor for # of blocks.

   -f# Fudge factor for # of i-nodes.

   -u Print usage requirements on a per file system basis.

EXAMPLES

The following command reports the number of blocks used by the files and directories in the current working directory:

```
ls | fsize
```

The following command reports the number of blocks used by the files in a file list:

```
cat filelist | fsize
```

The following comamnd reports the number of blocks used by all the files and directories in the entire subtree of the current working directory and prints information for each file individually, one per line:

```
find . -print | fsize -s
```

The following command calculates whether there is enough space for the group whose files are in **filelist**, adjusts for existing files, prints messages corresponding to various specified options, and so forth [this is an actual example from *ctinstall*(1)]:

```
cat filelist | fsize -mqciu -b100 -f20
```

SEE ALSO

du(1).

BUGS

Remotely mounted file systems are not properly recognized, and are treated as part of the nearest *locally* mounted file system, as demonstrated by the following form of the command: fsize **-mu**.

NAME

fsplit - split FORTRAN, ratfor, or efl files

SYNOPSIS

fsplit [options] [files]

DESCRIPTION

fsplit splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Procedure *X* is put in file *X*.**f**, *X*.**r**, or *X*.**e** depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN*.[**efr**] and unnamed *blockdata* segments in the files *blockdataN*.[**efr**] where *N* is a unique integer value for each file.

The following *options* pertain:

-f (default) Input files are FORTRAN.

-r Input files are *ratfor*.

-e Input files are *efl*.

-s Strip FORTRAN input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), split(1).

NAME

fsstat - report file system status

SYNOPSIS

/etc/fsstat special_file

DESCRIPTION

The *fsstat* comand reports on the status of the file system on *special_file*. During startup, this command is used to determine if the file system needs checking before it is mounted. The *fsstat* command succeeds if the file system is unmounted and appears okay. For the **root** file system, *fsstat* succeeds if the file system is active and not marked bad.

SEE ALSO

fs(4).

DIAGNOSTICS

The command has the following exit codes:

0 The file system is not mounted and appears okay, (except for root where 0 means mounted and okay).

1 The file system is not mounted and needs to be checked.

2 The file system is mounted.

3 The command failed.

NAME

fstyp - determine file system identifier

SYNOPSIS

fstyp *special*

DESCRIPTION

The *fstyp* command allows the user to determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by *mount*(2) and sometimes by *mount*(1M) to mount file systems of different types.

The directory **/etc/fstyp.d** contains a program for each file system type to be checked; each program applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; otherwise it prints error messages on standard error and exits with a non-zero return code. The *fstyp* command runs the programs in **/etc/fstyp.d** in alphabetical order, passing *special* as an argument; if any program succeeds, its file-system type identifier is printed and *fstyp* exits immediately. If no program succeeds, *fstyp* prints "Unknown_fstyp" to indicate failure.

WARNING

The use of heuristics implies that the result of *fstyp* is not guaranteed to be accurate.

SEE ALSO

mount(1M), mount(2), sysfs(2).

NAME

ftp - ARPANET file transfer program

SYNOPSIS

ftp [**-v**] [**-d**] [**-i**] [**-n**] [**-g**] [host]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol. The program copies files to and from a remote node. It is more general than *rcp*(1), because a File Transfer Protocol server is available under a wider range of operating systems.

The client node with which *ftp* is to communicate can be specified on the command line. If this is done, *ftp* immediately attempts to establish a connection to an FTP server on that host; otherwise, *ftp* enters its command interpreter and awaits instructions from the user. When *ftp* is awaiting commands from the user, the prompt "ftp>" is displayed.

(In the discussions below, commands in all capital letters, such as PASV, are internal, rather than user, FTP commands.)

OPTIONS

The following options can be specified at the command line, or to the command interpreter:

-v Force *ftp* to show all responses from the remote server, as well as report on data transfer statistics.

-n Restrain *ftp* from attempting "auto-login" upon initial connection. If auto-login is enabled, *ftp* checks the **.netrc** file in the user's home directory for an entry describing an account on the remote machine (see "The **.netrc** File" below). If no entry exists, *ftp* prompts for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

-i Disable interactive prompting during multiple file transfers.

-d Enable debugging.

-g Disable "globbing" (file name expansion using wildcard characters).

COMMANDS

The following commands are recognized by *ftp*. Optional command arguments can be specified on the command line; otherwise, *ftp* enters its command interpreter and prompts for arguments. Note that each machine session normally begins with an **open** command and ends with a **close** or a **bye** command.

! [*command* [*args*]]
Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

$ *macro-name* [*args*]
Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro "unglobbed."

account [*string*]
Send the ACCT command to the remote system with the given *string* as an argument. If no argument is specified, the user is prompted in non-echoing mode. The interpretation of this command is dependent on the remote system: it is sometimes used for a required second password.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any **ntrans** or **nmap** setting. File transfer uses the current settings for **type**, **format**, **mode**, and **structure**.

ascii Set the file transfer **type** to network ASCII. This is the default type. (Refer to "File Transfer Parameters" below.)

bell Arrange for a bell to be sounded after each file transfer command is completed.

binary Set the file transfer *type* to support binary image transfer.

bye Terminate the FTP session with the remote server and exit *ftp*. An end of file also terminates the session and exits.

case Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage return stripping during ASCII type file retrieval. Records are denoted by a carriage return/linefeed sequence during ASCII type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the UNIX and CTIX single linefeed record delimiter. Records on remote systems that are not running UNIX or CTIX can contain single linefeeds; when an ASCII type transfer is made, these linefeeds can be distinguished from a record delimiter only when **cr** is off.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string "-->".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is -, output comes to the terminal.

disconnect
A synonym for **close**.

form *format*
Set the file transfer form to *format*. The default format is **file**. (*ftp* supports only the default value for format.)

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current **case**, **ntrans**, and **nmap** settings. The current settings for **type**, **form**, **mode**, and **structure** are used while transferring the file.

glob Toggle filename expansion ("globbing") for **mdelete**, **mget** and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and not expanded. Globbing for **mput** is done as in **csh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign

operating system and ftp server, and can be previewed by doing "**mls** *remote-files* -". Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring an archive [created with a copy program such as **tar**(1) or **cpio**(1)] of the subtree (in binary mode).

hash Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is -, the output is sent to the terminal.

macdef *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets **$** and \ as special characters. A **$** followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A **$** followed by an **i** signals the macro processor that the executing macro is to be looped. On the first pass **$i** is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A \ followed by any character is replaced by that character. Use the \ to prevent special treatment of the **$**.

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
Like **dir**, except multiple remote files can be specified. If interactive prompting is enabled, *ftp* prompts the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*

Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names are then processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which can be changed with "**lcd** *directory*"; new local directories can be created with "**! mkdir** *directory*".

mkdir *directory-name*

Make a directory on the remote machine.

mls *remote-files local-file*

Like **ls**, except multiple remote files can be specified. If interactive prompting is on, *ftp* prompts the user to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]

Set the file transfer mode to *mode-name*. The default mode is **stream** mode. (*ftp* supports only the default value for mode.)

mput *local-files*

Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names are then processed according to **ntrans** and **nmap** settings.

nmap [*inpattern outpattern*]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a remote computer with different file naming conventions or practices (for example, a computer running an operating system different from CTIX and UNIX). The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences **$1**, **$2**, ..., **$9** in *inpattern*. Use \ to prevent this special treatment of the **$** character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values. For exmaple, given *inpattern* **$1.$2** and the remote file name

mydata.data, **$1** would have the value **mydata**, and **$2** would have the value **data**. The *outpattern* determines the resulting mapped filename. The sequences **$1**, **$2**, ..., **$9** are replaced by any value resulting from the *inpattern* template. The sequence **$0** is replaced by the original filename. Additionally, the sequence [*seq1*,*seq2*] is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command

```
nmap $1.$2.$3 [$1,$2].[$2,file]
```

would yield the output filename **myfile.data** for input filenames **myfile.data** and **myfile.data.old**, **myfile.file** for the input filename **myfile**, and **myfile.myfile** for the input filename **.myfile**. Spaces can be included in *outpattern*, as in the example:

```
nmap $1 | sed "s/ *$//" > $1
```

Use the \ character to prevent special treatment of the $, [,], and , characters.

ntrans [*inchars* [*outchars*]]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a remote computer with different file naming conventions or practices (for example, a computer running an operating system different from CTIX and UNIX). Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

open *host* [*port*]

Establish a connection to the specified *host* FTP server. An optional port number can be supplied, in which case, *ftp* attempts to contact an FTP server at that port. If the auto-login option is enabled (default), *ftp* also attempts to automatically log the user in to the FTP server (see description of **-n** option above). A connection to a second host can be established using the **proxy open** command.

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** transfers all files, and any **mdelete** deletes all files.

proxy *ftp-command*
Execute an *ftp* command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command **proxy ?** to see what *ftp* commands are executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** does not define new macros during the auto-login process, **close** does not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the FTP protocol PASV command by the server on the secondary control connection.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any *ntrans* or *nmap* settings in naming the remote file. File transfer uses the current settings for **type**, **format**, **mode**, and **structure**.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]
A synonym for **get**.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server as well.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

reset Clear reply queue. This command re-synchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

rmdir *directory-name*
Delete a directory on the remote machine.

runique
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a **.1 is appended to the name.** If the resulting name matches another existing file, a **.2** is appended to the original name. If this process continues up to **.99**, an error message is printed, and the transfer does not take place. The generated unique filename is reported. Note that **runique** does not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
A synonym for **put.**

sendport
Toggle the use of PORT commands. By default, *ftp* attempts to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, *ftp* uses the default data port. When the use of PORT commands is disabled, no attempt is made to use PORT commands for each data transfer. This is useful for certain FTP implementations that do ignore PORT commands but, incorrectly, indicate that they have been accepted.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer structure to *struct-name*. By default **stream** structure is used. (*ftp* supports only the default value for struct.)

sunique
Toggle storing of files on remote machine under unique file names. Remote FTP server must support FTP protocol STOU command for successful completion. The remote server reports unique name. Default value is off.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer type to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* prompts the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user is prompted for it. If an account field is specified, an account command is relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless *ftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for **help**.

Command arguments that have embedded spaces can be quoted with quotation marks (" ").

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually Control-C). Sending transfers are immediately halted. Receiving transfers are halted by sending an FTP protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt does not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence is ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode can result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the FTP protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules:

1. If the file name - is specified, the standard input (for reading) or the standard output (for writing) is used.

2. If the first character of the file name is |, the remainder of the argument is interpreted as a shell command. *ftp* then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; for example, **"| ls -lt"**. A particularly useful example of this mechanism is **dir . | more**.

3. Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the *csh*(1) (refer also to the discussion of the **glob** command). If the *ftp* command expects a single local file (for example, **put**), only the first filename generated by the "globbing" operation is used.

4. For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which can be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename can then be altered if **runique** is on.

5. For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which can be altered by a **ntrans** or **nmap** setting. The resulting filename can then be altered by the remote server if **sunique** is on.

FILE TRANSFER PARAMETERS

The FTP specification describes a number of parameters that can affect a file transfer. The *type* can be one of **ascii**, **image**(binary), **ebcdic**, and **local byte size** (for PDP-10's and PDP-20's mostly). *ftp* supports the **ascii** and **image** types of file transfer, plus local 8-bit byte size for **tenex** mode transfers.

ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

THE .netrc FILE

The **.netrc** file contains login and initialization information used by the auto-login process [see *netrc*(4)]. It resides in the user's home directory. The following tokens are recognized; they can be separated by spaces, tabs, or new-lines:

machine *name*

Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the *ftp* command line or as an **open** command argument. Once a match is made, the subsequent **.netrc** tokens are processed until the end of file is reached or another **machine** token is encountered.

login *name*

Identify a user on the remote machine. If this token is present, the auto-login process initiates a login using the specified name.

password *string*

Supply a password. If this token is present, the auto-login process supplies the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the **.netrc** file, *ftp* aborts the auto-login process if the **.netrc** is readable by anyone besides the user. This is to discourage publically readable passwords and is an important security feature.

account *string*

Supply an additional account password. If this token is present, the auto-login process supplies the specified string if the remote server requires an additional account password, or the auto-login process initiates an ACCT command if the remote machine does not actively request the account password.

macdef *name*

Define a macro. This token functions like the *ftp* **macdef** command functions. A macro is defined with the specified name; its contents begin with the next **.netrc** line and continue until a null line (consecutive new-line characters) is encountered. If a macro named **init** is defined, it is automatically executed as the last step in the auto-login process.

SEE ALSO

ftpd(1M), netrc(4).
CTIX Network Administrator's Guide.

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in previous releases of CTIX (4.2BSD UNIX) ASCII-mode transfer code has been corrected. This correction can result in incorrect transfers of binary files to and from 4.2BSD servers using the ASCII type. Avoid this problem by using the binary image type.

NAME

ftpd - DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [**-d**] [**-l**] [-ttimeout]

DESCRIPTION

The *ftpd* program is the DARPA Internet File Transfer Prototocol server process. The server uses the TCP protocol and listens at the port specified in the *ftp* service specification; see *services*(4).

The *ftpd* server is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

If the **-d** option is specified, debugging information is written to the syslog.

If the **-l** option is specified, each FTP session is logged in the syslog.

The FTP server times out an inactive session after 15 minutes. If the **-t** option is specified, the inactivity timeout period is set to *timeout*.

The FTP server currently supports the following FTP requests; case is not distinguished.

Request	**Description**
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory (**ls -lg**)
MKD	make a directory
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port

PWD	print the current working directory
QUIT	terminate session
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining FTP requests specified in RFC 959 are recognized, but not implemented.

The FTP server aborts an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in RFC 959.

The *ftpd* server interprets file names according to the "globbing" (file name expansion) conventions used by *csh*(1), allowing users to specify the metacharacters "*?[]{}~".

The *ftpd* server authenticates users according to the following rules:

1. The user name must be in the password data base, **/etc/passwd**, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.
2. The user name must not appear in the file **/etc/ftpusers**.
3. The user must have a standard shell.
4. If the user name is **anonymous** or **ftp** an anonymous *ftp* account must be present in the password file (user **ftp**). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the **ftp** user. To ensure that system security is not breached, the *ftp* subtree should be constructed with care; the following rules are recommended:

$HOME	Make the home directory owned by "ftp" and unwritable by anyone.
$HOME/bin	Make this directory owned by the superuser and unwritable by anyone. The program *ls*(1) must be present to support the list commands. This program should have mode 111.
$HOME/etc	Make this directory owned by the superuser and unwritable by anyone. The files *passwd*(4) and *group*(4) must be present for the *ls* command to work properly. These files should be mode 444.
$HOME/pub	Make this directory mode 777 and owned by **ftp.Users** should then place files which are to be accessible via the anonymous account in this directory.
$HOME/shlib	Make this directory owned by the super-user and unwritable by anyone. The file **libc2sw_s** (copied from **/shlib**) must be present for the *ls* command to work.

SEE ALSO

ftp(1), inetd(1M), inetd.conf(4), services(4).

BUGS

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user-ID of the logged-in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

fumount - forced unmount of an advertised resource

SYNOPSIS

fumount [-w sec] resource

DESCRIPTION

fumount unadvertises *resource* and disconnects remote access to the resource. The -w *sec* causes a delay of *sec* seconds prior to the execution of the disconnect.

When the forced unmount occurs, an administrative shell script is started on each remote computer that has the resource mounted (**/usr/bin/rfuadmin**). If a grace period of seconds is specified, *rfuadmin* is started with the **fuwarn** option. When the actual forced unmount is ready to occur, *rfuadmin* is started with the **fumount** option. See *rfuadmin*(1M) for information on the action taken in response to the forced unmount.

This command is restricted to the super-user.

ERRORS

If *resource* (1) does not physically reside on the local machine, (2) is an invalid resource name, (3) is not currently advertised and is not remotely mounted, or (4) the command is not run with super-user privileges, an error message will be sent to standard error.

SEE ALSO

adv(1M), mount(1M), rfuadmin(1M), rfudaemon(1M), rmount(1M), unadv(1M).

NAME

fusage - disk access profiler

SYNOPSIS

fusage [[mount_point] | [resource] |
[block_special_device] [...]]

DESCRIPTION

When used with no options, *fusage* reports block I/O transfers, in kilobytes, to and from all locally mounted file systems and remote resources (RFS or NFS) on a per-client basis. The count data are cumulative since the time of the mount. When used with an option, *fusage* reports on the named file system, remote resource, or block special device.

The report includes one section for each file system and remote resource, and has one entry for each machine that has the directory remotely mounted, ordered by decreasing usage. Sections are ordered by device name; resources that are not complete file systems immediately follow the sections for the file systems they are in.

SEE ALSO

adv(1M), mount(1M), df(1M), crash(1M).

NAME

fuser - identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**-ku**] files | resources [-] [[**-ku**] files | resources]

DESCRIPTION

The *fuser* command displays the process IDs of the processes using the *files* or remote *resources* specified as arguments. Each process ID is followed by a letter code, interpreted as follows:

1. If the process is using the file as its current directory, the code is **c**.
2. If the process is using the file as the parent of its current directory (only when the file is being used by the system), the code is **p**.
3. If the process is using the file as its *root* directory, the code is **r**.

For a regular type of *file* (text file, executable, directory, and so on), *fuser* reports only about the processes using that file. For block special devices with mounted file systems, *fuser* reports about all processes using any file on that device. If *resources* is used (meaning any remotely-mounted NFS or RFS resource), you must use the resource name, *not* the mount point of the resource. When you use the resource name, *fuser* reports about all processes using any file associated with that remote *resource*. (If you use the mount point of the resource, *fuser* reports only about those processes using that specific file.)

The following options can be used with *fuser*:

-u The user login name appears, in parentheses, following the process ID.

-k The SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages might not show up immediately [see *kill*(2)].

If more than one group of files are specified, the options can be respecified for each additional group of files. A lone dash (-) cancels the options currently in force; the new set of options applies to the next group of files.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

Note that you cannot list processes using a particular file from a remote resource mounted on your machine; you must use the resource name as an argument.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use *fuser*. Only the super-user can terminate another user's process.

FILES

/unix	for system namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M), ps(1), kill(2), signal(2).

NAME

fwtmp, wtmpfix - manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**-ic**]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

fwtmp

fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formatted ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument **-ic** is used to denote that input is in ASCII form, and output is to be written in binary form.

wtmpfix

wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctcon*(1) will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to **/etc/wtmp**. The first record is the old date denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field of the **<utmp.h>** structure. The second record specifies the new date and is denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. *wtmpfix* uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, *wtmpfix* will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to **INVALID** and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctcon*(1) will fail when processing connect accounting records.

FILES

/etc/wtmp
/usr/include/utmp.h

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), ed(1), runacct(1M), acct(2), acct(4), utmp(4).

NAME

gdev: hpd, erase, hardcopy, tekset, td - graphical device routines and filters

SYNOPSIS

hpd [- options] [GPS file ...]
erase
hardcopy
tekset
td [**-ern***n*] [GPS file ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** [see *graphics*(1G)].

hpd *hpd* translates a GPS [graphical primitive string; see *gps*(4)] to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **-u** or **-r** *option* is provided. If no *file* is given, the standard input is assumed. *options* are:

- **c***n* Select character set *n*, *n* between 0 and 5 (see the *HP7221A Plotter Operating and Programming Manual*, *Appendix A*).
- **p***n* Select pen numbered *n*, *n* between 1 and 4 inclusive.
- **r***n* Window on GPS region *n*, *n* between 1 and 25 inclusive.
- **s***n* Slant characters *n* degrees clockwise from the vertical.
- **u** Window on the entire GPS universe.
- **xd***n* Set x displacement of the viewport's lower left corner to *n* inches.
- **xv***n* Set width of viewport to *n* inches.
- **yd***n* Set y displacement of the viewport's lower left corner to *n* inches.
- **yv***n* Set height of viewport to *n* inches.

erase *Erase* sends characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, *hardcopy* generates a screen copy on the unit.

tekset *tekset* sends characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td *td* translates a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the **-u** or **-r** *option* is provided. If no *file* is given, the standard input is assumed. Options are:

- **e** Do not erase screen before initiating display.
- **r***n* Display GPS region *n*, *n* between 1 and 25 inclusive.
- **u** Display the entire GPS universe.

SEE ALSO

ged(1G), graphics(1G), gps(4).

NAME

ged - graphical editor

SYNOPSIS

/usr/bin/graf/ged [**-euRrn**] [GPS file ...]

DESCRIPTION

The *ged* editor is an interactive graphical editor used to display, construct, and edit GPS files on Tektronix 4010 series display terminals. If GPS *file*(s) are given, *ged* reads them into an internal display buffer and displays the buffer. The GPS in the buffer can then be edited. If - is given as a file name, *ged* reads a GPS from the standard input.

The *ged* editor accepts the following command line options:

- **e** Do not erase the screen before the initial display.
- **r***n* Display region number *n*.
- **u** Display the entire GPS *universe*.
- **R** Restricted shell invoked on use of !.

A GPS file is composed of instances of three graphical objects: *lines*, *arc*, and *text*. *Arc* and *lines* objects have a start point, or *object-handle*, followed by zero or more points, or *point-handles*. *Text* has only an object-handle. The objects are positioned within a Cartesian plane, or *universe*, having 64K (-32K to +32K) points, or *universe-units*, on each axis. The universe is divided into 25 equal sized areas called *regions*. Regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

The *ged* editor maps rectangular areas, called *windows*, from the universe onto the display screen. Windows allow the user to view pictures from different locations and at different magnifications. The *universe-window* is the window with minimum magnification: the window that views the entire universe. The *home-window* is the window that completely displays the contents of the display buffer.

COMMANDS

The *ged* commands are entered in *stages*; Typically each stage ends with a **<cr>** (Return). Prior to the final **<cr>** the command can be aborted by typing **rubout**. The input of a stage can be edited during the stage using the erase and kill characters of the calling shell. The prompt * indicates that *ged* is waiting at stage 1.

Each command consists of a subset of the following stages:

1. *Command line* — A *command line* consists of a command *name* followed by *argument*(s) followed by a **<cr>**. A command *name* is a single character. Command *arguments* are either *option*(s) or a *file-name*. *Options* are indicated by a leading **-**.

2. *Text* — *Text* is a sequence of characters terminated by an unescaped **<cr>** (120 lines of text maximum).

3. *Points* — *Points* is a sequence of one or more screen locations (maximum of 30) indicated either by the terminal crosshairs or by name. The prompt for entering *points* is the appearance of the crosshairs. When the crosshairs are visible, you can type any of the following:

sp (space_ enters the current location as a *point*, which is identified with a number.

$*n* enters the previous *point* numbered *n*.

>*x* labels the last *point* entered with the upper case letter *x*.

$*x* enters the *point* labeled *x*.

. establishes the previous *points* as the current *points*. At the start of a command the previous *points* are those locations given with the previous command.

= echoes the current *points*.

$.*n* enters the *point* numbered *n* from the previous *points*.

erases the last *point* entered.

@ erases all of the *points* entered.

4. *Pivot* — The *pivot* is a single location, entered by typing **<cr>** or by using the **$** operator, and indicated with a *****.

5. *Destination* — The *destination* is a single location entered by typing **<cr>** or by using **$**.

COMMAND SUMMARY

In the summary, characters typed by the user are printed in **bold**. Command stages are printed in *italics*. Arguments surrounded by brackets "[ ]" are optional. Parentheses "()" surrounding arguments separated by "or" means that exactly one of the arguments must be given.

Construct commands:

Arc	[-echo,style,weight] *points*
Box	[-echo,style,weight] *points*
Circle	[-echo,style,weight] *points*
Hardware	[-echo] *text points*
Lines	[-echo,style,weight] *points*
Text	[-angle,echo,height,mid-point,right-point,text,weight] *text points*

Edit commands:

Delete	(- (universe or view) or *points*)
Edit	[-angle,echo,height,style,weight] (- (universe or view) or *points*)
Kopy	[-echo,points,x] *points pivot destination*
Move	[-echo,points,x] *points pivot destination*
Rotate	[-angle,echo,kopy,x] *points pivot destination*
Scale	[-echo,factor,kopy,x] *points pivot destination*

View commands:

coordinates	*points*
erase	
new-display	
object-handles	(- (universe or view) or *points*)
point-handles	(- (labelled-points or universe or view) or *points*)
view	(- (home or universe or region) or [-x] *pivot destination*)
x	[-view] *points*
zoom	[-out] *points*

Other commands:

quit or Quit	
read	[-angle,echo,height,mid-point,right-point,text,weight *file-name* [*destination*]
set	[-angle,echo,factor,height,kopy,mid-point,points, right-point,style,text,weight,x]

write *file-name*

!*command*

?

Options:

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specifed as an *option*, the default value for the parameter will be used (see set below). The format of command *options* follows:

-*option* [,*option*]

where *option* is *keyletter*[*value*]. Flags take on the *values* of true or false indicated by + and - respectively. If no *value* is given with a flag, true is assumed.

Object options:

angle*n*	Angle of *n* degrees.
echo	When true, echo additions to the display buffer.
factor*n*	Scale factor is *n* percent.
height*n*	Height of *text* is *n* universe-units ($0 \leq n < 1280$).
kopy	When true, copy rather than move.
mid-point	When true, mid-point is used to locate text string.
points	When true, operate on points; otherwise operate on objects.
right-point	When true, right-point is used to locate *text* string.
style*type*	Line style set to one of following *types*: **so** solid **da** dashed **dd** dot-dashed **do** dotted **ld** long-dashed
text	When false, *text* strings are outlined rather than drawn.
weight*type*	Sets line weight to one of following *types*: **n** narrow **m** medium **b** bold

Area options:

home	Reference the home-window.
out	Reduce magnification.
region*n*	Reference region *n*.
universe	Reference the universe-window.
view	Reference those objects currently in view.
x	Indicate the center of the referenced area.

COMMAND DESCRIPTIONS

Construct commands:

Arc and Lines

behave similarly. Each consists of a *command line* followed by *points*. The first *point* entered is the object-handle. Successive *points* are point-handles. Lines connect the handles in numerical order. Arc fits a curve to the handles (currently a maximum of 3 points will be fit with a circular arc; splines will be added in a later version).

Box and Circle

are special cases of Lines and Arc, respectively. Box generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle would connect the first *point* entered with the last *point*. The first *point* is the object-handle. Point-handles are created at each of the vertices. Circle generates a circular arc centered about the *point* numbered zero and passing through the last *point*. The circle's object-handle coincides with the last *point*. A point-handle is generated 180 degrees around the circle from the object-handle.

Text and Hardware

generate *text* objects. Each consists of a *command line*, *text* and *points*. *Text* is a sequence of characters delimited by <cr>. Multiple lines of text may be entered by preceding a cr with a backslash (that is, \cr). The Text command creates software-generated characters. Each line of software text is treated as a separate *text* object. The first *point* entered is the object-handle for the first line of text. The Hardware command sends the characters in *text* uninterpreted to the terminal.

Edit commands:

Edit commands operate on portions of the display buffer called *defined areas*. A defined area is referenced either with an area *option* or interactively. If an area *option* is not given, the perimeter of the defined area is indicated by *points*. If no *point* is entered, a small defined area is built around the location of the

<cr>. This is useful to reference a single *point*. If only one *point* is entered, the location of the <cr> is taken in conjunction with the *point* to indicate a diagonal of a rectangle. A defined area referenced by *points* will be outlined with dotted lines.

Delete
removes all objects whose object-handle lies within a defined area. The universe option removes all objects and erases the screen.

Edit modifies the parameters of the objects within a defined area. Parameters that can be edited are:

angle	angle of *text*
height	height of *text*
style	style of *lines* and *arc*
weight	weight of *lines*, *arc*, and *text*.

Kopy (or Move)
copies (or moves) object- and/or point-handles within a defined area by the displacement from the *pivot* to the *destination*.

Rotate
rotates objects within a defined area around the *pivot*. If the kopy flag is true then the objects are copied rather than moved.

Scale For objects whose object handles are within a defined area, point displacements from the *pivot* are scaled by factor percent. If the kopy flag is true then the objects are copied rather than moved.

View commands:

coordinates
prints the location of *point*(s) in universe- and screen-units.

erase clears the screen (but not the display buffer).

new-display
erases the screen then displays the display buffer.

object-handles (or point-handles)
labels object-handles (and/or point-handles) that lie within the defined area with O (or P). Point-handles identifies labeled points when the labelled-points flag is true.

view moves the window so that the universe point corresponding to the *pivot* coincides with the screen point corresponding to the *destination*. Options for home, universe, and region display particular windows in the universe.

x indicates the center of a defined area. Option view indicates the center of the screen.

zoom

decreases (zoom out) or increases the magnification of the viewing window based on the defined area. For increased magnification, the window is set to circumscribe the defined area. For a decrease in magnification the current window is inscribed within the defined area.

Other commands:

quit or Quit

exit from *ged*. Quit responds with ? if the display buffer has not been written since the last modification.

read inputs the contents of a file. If the file contains a GPS it is read directly. If the file contains text it is converted into *text* object(s). The first line of a text file begins at *destination*.

set when given *option*(s) resets default parameters, otherwise it prints current default values.

write outputs the contents of the display buffer to a file.

! escapes *ged* to execute a CTIX system command.

? lists *ged* commands.

SEE ALSO

gdev(1G), graphics(1G), sh(1), gps(4).

WARNING

See Appendix A of the *Tektronix 4014 Computer Display Terminal User's Manual* for a discussion of the appropriate terminal strap options.

NAME

gencc - create a front-end to the cc command

SYNOPSIS

gencc

DESCRIPTION

The *gencc* command is an interactive command designed to aid in the creation of a front-end to the *cc* command. Since hard-coded pathnames have been eliminated from the C Compilation System (CCS), it is possible to move pieces of the CCS to new locations without recompiling the CCS. The new locations of moved pieces can be specified through the **-Y** option to the *cc* command. However, it is inconvenient to supply the proper **-Y** options with every invocation of the *cc* command. Further, if a system administrator moves pieces of the CCS, such movement should be invisible to users.

The front-end to the *cc* command which *gencc* generates is a one-line shell script which calls the *cc* command with the proper **-Y** options specified. The front-end to the *cc* command will also pass all user supplied options to the *cc* command.

(Note that *cc1sw*(1), *cc2sw*, and *cc2fp* are also available as front-ends to the *cc* command. These programs were themselves generated with *gencc*.)

gencc prompts for the location of each tool and directory which can be respecified by a **-Y** option to the *cc* command. If no location is specified, it assumes that that piece of the CCS has not been relocated. After all the locations have been prompted for, *gencc* will create the front-end to the *cc* command.

gencc creates the front-end to the *cc* command in the current working directory and gives the file the same name as the *cc* command. Thus, *gencc* can not be run in the same directory containing the actual *cc* command. Further, if a system administrator has redistributed the CCS, the actual *cc* command should be placed somewhere which is not typically in a user's PATH (for example, /lib). This will prevent users from accidentally invoking the *cc* command without using the front-end.

CAVEATS

gencc does not produce any warnings if a tool or directory does not exist at the specified location. Also, *gencc* does not actually move any files to new locations.

FILES

./cc	front-end to cc

SEE ALSO

cc(1), cc1sw(1).

NAME

get - get a version of an SCCS file

SYNOPSIS

get [**-r**SID] [**-c**cutoff] [**-i**list] [**-x**list] [**-w**string] [**-a**seq-no.] [**-k**] [**-e**] [**-l**[**p**] [**-p**] [**-m**] [**-n**] [**-s**] [**-b**] [**-g**] [**-t**] file ...

DESCRIPTION

The *get* command generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with -. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the **-e** keyletter is also used), as a function of the SID specified.

-c*cutoff* *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **-c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "**-c77/2/2 9:22:25**". Note that this implies that one may use the %E% and %U% identification

keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

```
~!get "-c%E% %U%" s.file
```

-i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1.

-x*list* A *list* of deltas to be excluded in the creation of the generated file. See the **-i** keyletter for the *list* format.

-e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The **-e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file [see *admin*(1)]. Concurrent use of **get -e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **-e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **-k** keyletter in place of the **-e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file [see *admin*(1)] are enforced when the **-e** keyletter is used.

-b Used with the **-e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file [see *admin*(1)] or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note that a branch *delta* may always be created from a non-leaf *delta*. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** keyletter is implied by the **-e** keyletter.

-l[p] Causes a delta summary to be written into an *l-file*. If **-lp** is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **-s** keyletter is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **-m** keyletter generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

-t Used to access the most recently created delta in a given release (for example, **-r1**), or release and level (for example, **-r1.2**).

-w *string* Substitute *string* for all occurrences of %W% when getting the file.

-a*seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved [see *sccsfile*(4)]. This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter. If both the **-r** and **-a** keyletters are specified, only the **-a** keyletter is used. Care should be taken when using the **-a** keyletter in conjunction with the **-e** keyletter, as the SID of the delta to be created may not be what one expects. The **-r** keyletter can be used with the **-a** and **-e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** keyletter is used included deltas are listed following the notation "Included"; if the **-x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	-b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	-	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (that is, maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

***

This is used to force creation of the *first* delta in a *new* release.

\# Successor.

† The **-b** keyletter is effective only if the **b** flag [see *admin*(1)] is present in the file. An entry of - means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file [see *admin*(1)], or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).

%H% Current date (MM/DD/YY).

%T% Current time (HH:MM:SS).

%E% Date newest applied delta was created (YY/MM/DD).

%G% Date newest applied delta was created (MM/DD/YY).

%U% Time newest applied delta was created (HH:MM:SS).

%Y% Module type: value of the **t** flag in the SCCS file [see *admin*(1)].

%F% SCCS file name.

%P% Fully qualified SCCS file name.

%Q% The value of the **q** flag in the file [see *admin*(1)].

%C% Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.

%Z% The 4-character string **@(#)** recognizable by *what*(1).

%W% A shorthand notation for constructing *what*(1) strings for CTIX system program files. %W% = %Z%%M%<horizontal-tab>%I%

%A% Another shorthand notation for constructing *what*(1) strings for non-CTIX system program files.
%A% = %Z%%Y% %M% %I%%Z%

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **-p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **-k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **-l** keyletter is

used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
* otherwise.

b. A blank character if the delta was applied or was not applied and ignored;
* if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was not applied:
"I": Included.
"X": Excluded.
"C": Cut off (by a **-c** keyletter).

d. Blank.

e. SCCS identification (SID).

f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.

h. Blank.

i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an **-e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **-e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, [see *admin*(1)] is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **-i** keyletter argument if it was present, followed by a blank and the **-x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (that is, *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

FILES

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1).
UNIX System V Release 3.2 Programmer's Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the **-e** keyletter is used.

NAME

getopt - parse command options

SYNOPSIS

set -- `getopt** optstring **$*`

DESCRIPTION

WARNING: Start using the new command *getopts* (1) in place of *getopt* (1). *getopt* (1) will not be supported in the next major release. For more information, see the **WARNINGS** section, below.

getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *optstring* is a string of recognized option letters [see *getopt*(3C)]; if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option -- is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The positional parameters ($1 $2 ...) of the shell are reset so that each option is preceded by a - and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)        FLAG=$i; shift;;
        -o)             OARG=$2; shift 2;;
        --)             shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

getopts(1), sh(1), getopt(3C).

DIAGNOSTICS

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

WARNINGS

getopt (1) does not support the part of Rule 8 of the command syntax standard [see *intro* (1)] that permits groups of option-arguments following an option to be separated by white space and quoted. For example,

```
cmd -a -b -o "xxx z yy" file
```

is not handled correctly). To correct this deficiency, use the new command *getopts* (1) in place of *getopt* (1).

getopt (1) will not be supported in the next major release. For this release a conversion tool has been provided, *getoptcvt*. For more information about *getopts* and *getoptcvt*, see the *getopts* (1) manual page.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring* (referring to the earlier EXAMPLE section, but using the following command line: *getopt* will always treat **-a** as an option-argument to **-o**; it will never recognize **-a** as an option. For this case, the **for** loop in the example will shift past the *file* argument.

NAME

getopts, getoptcvt - parse command options

SYNOPSIS

getopts optstring name [arg ...]

/usr/lib/getoptcvt [**-b**] file

DESCRIPTION

getopts is used by shell procedures to parse positional parameters and to check for legal options. It supports all applicable rules of the command syntax standard [see Rules 3-10, *intro*(1)]. It should be used in place of the *getopt*(1) command. (See the **WARNING**, below.)

optstring must contain the option letters the command using *getopts* will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, *getopts* will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, *getopts* places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, *getopts* exits with a non-zero exit status. The special option "**--**" may be used to delimit the end of the options.

By default, *getopts* parses the positional parameters. If extra arguments (*arg ...*) are given on the *getopts* command line, *getopts* will parse them instead.

/usr/lib/getoptcvt reads the shell script in *file*, converts it to use *getopts*(1) instead of *getopt*(1), and writes the results on the standard output.

-b the results of running */usr/lib/getoptcvt* will be portable to earlier releases of the CTIX system. */usr/lib/getoptcvt* modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke *getopts*(1) or *getopt*(1).

So all new commands will adhere to the command syntax standard described in *intro*(1), they should use *getopts*(1) or *getopt*(3C) to parse positional parameters and check for options that are legal for that command (see **WARNINGS**, below).

EXAMPLE

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
done
shift `expr $OPTIND - 1`
```

This code will accept any of the following as equivalent:

```
cmd -a -b -o "xxx z yy" file
cmd -a -b -o "xxx z yy" -- file
cmd -ab -o xxx,z,yy file
cmd -ab -o "xxx z yy" file
cmd -o xxx,z,yy -b -a file
```

SEE ALSO

intro(1), sh(1), getopts(3C).

WARNING

Although the following command syntax rule [see *intro*(1)] relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section above, **a** and **b** are options, and the option **o** requires an option-argument:

cmd -aboxxx file

(Rule 5 violation: options with option-arguments must not be grouped with other options)

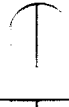
cmd -ab -oxxx file

(Rule 6 violation: there must be white space after an option that takes an option-argument)

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.



NAME

getservaddr - get network address of service host

SYNOPSIS

getservaddr host service

DESCRIPTION

getservaddr writes to standard output the network address of the specified *host* and *service*. It gets the address from the **/etc/hosts** file [or the name server *named*(1M)] and the **/etc/services** file. The address is written as hexadecimal ASCII, preceded by \x.

The intended use of *getservaddr* is to provide network addresses for Remote File Sharing (RFS). For example, if the node Convgt has the internet address of 3.180.0.7, the execution of the following command:

getservaddr Convgt nlsgen

will produce the following to standard output:

\x0002040103b40007

FILES

/etc/hosts
/etc/services

SEE ALSO

hosts(4), services(4).

NAME

getty - set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**-h**] [**-t** timeout] line [speed [type [linedisc]]]

/etc/getty -c file

DESCRIPTION

getty is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with the CTIX system. It can only be executed by the super-user; that is, a process with the user-ID of **root**. Initially *getty* generates a system identification message from the values returned by the *uname*(2) system call. Then, if **/etc/issue** exists, it outputs this to the user's terminal, followed finally by the login message field for the entry it is using from **/etc/gettydefs**. *getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used. It does this by using the options and arguments specified.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **-h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **-t** flag plus *timeout* (in seconds), specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds.

Speed, the optional second argument, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a <*break*> character). The default *speed* is 9600 baud.

Type, the optional third argument, is a character string describing to *getty* what type of terminal is connected to the line in question. *getty* recognizes the following types:

none	default
ds40-1	Dataspeed40/1
tektronix,tek	Tektronix
vt61	DEC vt61
vt100	DEC vt100

hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none**; that is, any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition.

Linedisc, the optional fourth argument, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISC0**.

When given no optional arguments, *getty* sets the *speed* of the interface to 9600 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

After the user's name has been typed in, it is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately [see *ioctl*(2)].

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

In addition to the standard UNIX system erase and kill characters (**#** and **@**), *getty* also understands **\b** and **^U**. If the user uses a **\b** as an erase, or **code-U** as a kill character, *getty* sets the standard erase character and/or kill character to match.

getty also understands the "standard" ESS (AT&T hardware-specific Electronic Switching System) protocols for erasing, killing and aborting a line, and terminating a line. If *getty* sees the ESS erase character, **_**, or kill character, **$**, or abort character, **&**, or the ESS line terminators, **/** or **!**, it arranges for this set of characters to be used for these functions.

Finally, *login* is **exec**'d with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment [see *login*(1)].

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1C), init(1M), login(1), ioctl(2), gettydefs(4), inittab(4), tty(7).

BUGS

While *getty* understands simple single character quoting conventions, it is not possible to quote certain special control characters used by *getty*. Thus, you cannot login via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. *getty* uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.

NAME

glossary - definitions of common CTIXT system terms and symbols

SYNOPSIS

[**help**] **glossary** [term]

DESCRIPTION

The CTIX system Help Facility command *glossary* provides definitions of common technical terms and symbols.

Without an argument, *glossary* displays a menu screen listing the terms and symbols that are currently included in *glossary*. A user can choose one of the terms or can exit to the shell by typing **q** (for quit). When a term is selected, its definition is retrieved and displayed. By selecting the appropriate menu choice, the list of terms and symbols can be redisplayed.

A term's definition can also be requested directly from shell level (as shown above), causing a definition to be retrieved and the list of terms and symbols not to be displayed. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following table lists the symbols and their escape sequences.

SYMBOL	ESCAPE SEQUENCE
""	\"\"
''	\'\'
[]	\\[\\]
‘‘	\‘\‘
#	\#
&	\&
*	*
\	\\\\
|	|

From any screen in the Help Facility, a user can execute a command from the shell [*sh*(1)] by typing a ! and the command to be executed. The screen is redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt is redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (nonscrolling), the shell variable **SCROLL** must be set to **no** and exported so it becomes part of your environment. This is done by adding the following line to your *.profile* file [see *profile* (4)]:

```
export SCROLL; SCROLL=no
```

If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

help(1), helpadm(1M), locate(1), sh(1), starter(1), usage(1), term(5).

WARNINGS

If the shell variable **TERM** [see *sh*(1)] is not set in the user's *.profile* file, **TERM** defaults to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

graph - draw a graph

SYNOPSIS

graph [options]

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 *tplot*(1G) filters.

If the coordinates of a point are followed by a non-numeric 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 new-lines.

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 (for example, the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

-s Save screen, do not 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

graphics(1G), spline(1G), tplot(1G).

BUGS

graph stores all points internally and drops those for which there is no room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes cannot be reversed.

NAME

graphics - access graphical and numerical commands

SYNOPSIS

graphics [**-r**]

DESCRIPTION

graphics prefixes the path name **/usr/bin/graf** to the current **$PATH** value, changes the primary shell prompt to ^, and executes a new shell. The directory **/usr/bin/graf** contains all of the Graphics subsystem commands. If the **-r** option is given, access to the graphical commands is created in a restricted environment; that is, **$PATH** is set to

:/usr/bin/graf:/rbin:/usr/rbin

and the restricted shell, *rsh*, is invoked. To restore the environment that existed prior to issuing the *graphics* command, type **EOT** (control-d on most terminals). To logoff from the graphics environment, type **quit**.

The command line format for a command in *graphics* is *command name* followed by *argument*(s). An *argument* may be a *file name* or an *option string*. A *file name* is the name of any CTIX system file except those beginning with -. The *file name* - is the name for the standard input. An *option string* consists of - followed by one or more *option*(s). An *option* consists of a keyletter possibly followed by a value. *Options* may be separated by commas.

The graphical commands have been partitioned into four groups.

Commands that manipulate and plot numerical data; see *stat*(1G).

Commands that generate tables of contents; see *toc*(1G).

Commands that interact with graphical devices; see *gdev*(1G) and *ged*(1G).

A collection of graphical utility commands; see *gutil*(1G).

A list of the *graphics* commands can be generated by typing **whatis** in the *graphics* environment.

SEE ALSO

gdev(1G), ged(1G), gutil(1G), stat(1G), toc(1G), gps(4).

NAME

greek - select terminal filter

SYNOPSIS

greek [-Tterminal]

DESCRIPTION

greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character Teletype Model 37 terminal for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable **$TERM** [see *environ*(5)]. Currently, the following *terminal*s are recognized:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), eqn(1), hp(1), mm(1), nroff(1), tplot(1G), environ(5), term(5).

NAME

grep - search a file for a pattern

SYNOPSIS

grep [options] limited regular expression [file ...]

DESCRIPTION

The *grep* command searches files for a pattern and prints all lines that contain that pattern. *grep* uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with *ed (1)* to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters $, *, [, ^, |, (,), and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes '...'.

If no files are specified, *grep* assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-i Ignore upper/lower case distinction during comparisons.

-l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-s Suppress error messages about nonexistent or unreadable files

-v Print all lines except those that contain the pattern.

SEE ALSO

ed(1), egrep(1), fgrep(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 (even if matches were found).

BUGS

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**. If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

gutil - graphical utilities

SYNOPSIS

command-name [options] [files]

DESCRIPTION

Below is a list of miscellaneous device-independent utility commands found in **/usr/bin/graf**. If no *files* are given, input is from the standard input. All output is to the standard output. Graphical data is stored in GPS format; see *gps*(4).

bel Send bel character to terminal

cvrtopt [= **s***string* **f***string* **i***string* **t***string*] [*args*] - options converter
Cvrtopt reformats *args* (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a file name (a string not beginning with a -, or a - by itself) or an option string (a string of options beginning with a -). Output is of the form:

-option -option . . . file name(s)

All options appear singularly and preceding any file names. Options that take values (for example, **-r**1.1) or are two letters long must be described through options to *cvrtopt*.

Cvrtopt is usually used with *set* in the following manner as the first line of a shell procedure:

set - `cvrtopt =**[*options*] **$@`

Options to *cvrtopt* are:

s*string* *String* accepts string values.

f*string* *String* accepts floating point numbers as values.

i*string* *String* accepts integers as values.

t*string* *String* is a two-letter option name that takes no value.

String is a one- or two-letter option name.

gd [GPS *files*] - GPS dump
Gd prints a human readable listing of GPS.

gtop [**-r***n* **u**] [GPS *files*] - GPS to *plot*(4) filter
Gtop transforms a GPS into *plot*(4) commands displayable by *plot* filters. GPS objects are translated if they fall within the window that circumscribes the first *file* unless an *option* is given.

Options:

r*n* translate objects in GPS region *n*.

u translate all objects in the GPS universe.

pd [*plot*(5) *files*] - *plot*(4) dump
Pd prints a human readable listing of *plot*(4) format graphical commands.

ptog [*plot*(5) *files*] - *plot*(4) to GPS filter
Ptog transforms *plot*(4) commands into a GPS.

quit Terminate session

remcom [*files*] - remove comments
Remcom copies its input to its output with comments removed. Comments are as defined in C (that is, /* comment */).

whatis [**-o**] [*names*] - brief on-line documentation
Whatis prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. The command **whatis *** prints out every description.
Option:

o just print command options

yoo *file* - pipe fitting
Yoo is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that, without *yoo*, this is not usually successful as it causes a read and write on the same file simultaneously.

SEE ALSO

graphics(1G), gps(4), plot(4).

NAME

hd - hexadecimal and ascii file dump

SYNOPSIS

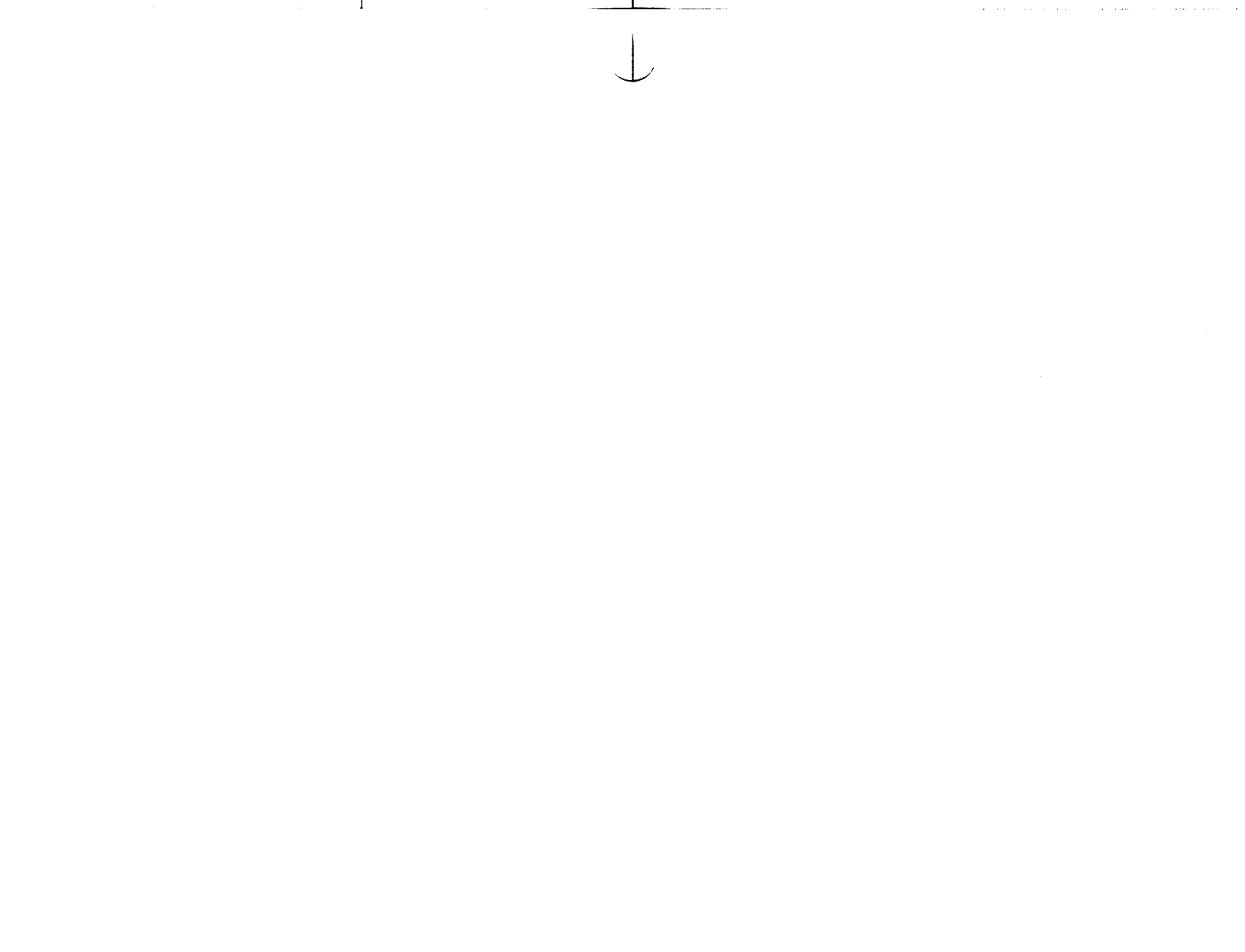
/usr/local/bin/hd *file*

DESCRIPTION

hd prints a hexadecimal listing of *file*, side by side with an ASCII listing.

SEE ALSO

od(1).



NAME

head - give first few lines

SYNOPSIS

/usr/local/bin/head [*-count*] [*file* ...]

DESCRIPTION

head gives the first *count* lines of each of the specified files. If no files are specified, *head* reads the standard input. If you omit *count*, *head* prints the first 10 lines.

SEE ALSO

tail(1).

NAME

help - CTIX system Help Facility

SYNOPSIS

help

[**help**] **starter**

[**help**] **usage** [**-d**] [**-e**] [**-o**] [command_name]

[**help**] **locate** [keyword1 [keyword2] ...]

[**help**] **glossary** [term]

help arg ...

DESCRIPTION

The CTIX system Help Facility provides on-line assistance for CTIX system users, whether they desire general information or specific assistance for use of the Source Code Control System (SCCS) commands.

Without arguments, *help* prints a menu of available on-line assistance commands with a short description of their functions. The commands and descriptions follow:

starter	information about the CTIX system for the beginning user
locate	locate CTIX system commands using function-related keywords
usage	CTIX system command usage information
glossary	definitions of CTIX system technical terms

The user can choose one of the above commands by entering its corresponding letter (given in the menu) or exit to the shell by typing **q** (for "quit").

With arguments, *help* directly invokes the named on-line assistance command, bypassing the initial *help* menu. The commands *starter*, *locate*, *usage*, and *glossary*, optionally preceded by the word *help*, can also be specified at shell level. When executing *glossary* from shell level some of the symbols listed in the glossary must be escaped (preceded by one or more backslashes "\") to be understood by the Help Facility. For a list of symbols and the number of backslashes to use for each, refer to *glossary*(1).

From any screen in the Help Facility, a user can execute a command through shell [*sh*(1)] by typing a ! and the command to be executed. The screen is redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt is redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell

variable SCROLL must be set to **no** and exported so it becomes part of your environment. This is done by adding the following line to your *.profile* file [see *profile* (4)]:

```
export SCROLL ; SCROLL=no
```

If you later decide that scrolling is desired, SCROLL must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on the respective manual pages.

The Help Facility can be tailored to a customer's needs by use of the *helpadm*(1M) command.

If the first argument to *help* is different from *starter*, *usage*, *locate*, or *glossary*, *help* assumes information is being requested about the SCCS Facility. The arguments can be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines that produced the message (for example, **ge3** for message 3 from the *get* command).

type2 Does not contain numerics (as a command, such as *get*).

type3 Is all numeric (for example, 212).

SEE ALSO

admin(1), cdc(1), comb(1), delta(1), get(1), glossary(1), helpadm(1M), locate(1), prs(1), rmdel(1), sact(1), sccsdiff(1), sh(1), starter(1), unget(1), usage(1), val(1), vc(1), what(1), profile(4), sccsfile(4), term(5).

WARNINGS

If the shell variable **TERM** [see *sh*(1)] is not set in the user's *.profile* file, **TERM** defaults to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

helpadm - make changes to the Help Facility database

SYNOPSIS

/etc/helpadm

DESCRIPTION

The CTIX system Help Facility Administration command, *helpadm*, allows CTIX system administrators and command developers to define the content of the Help Facility database for specific commands and to monitor use of the Help Facility. The *helpadm* command can only be executed by login root, login bin, or a login that is a member of group bin.

The *helpadm* command prints a menu of 3 types of Help Facility data which can be modified, and 2 choices relating to monitoring use of the Help Facility. The five choices are:

- modify *startup* data
- add, modify, or delete a *glossary* term
- add, modify, or delete command data (description, options, examples, and keywords)
- prevent monitoring use of the Help Facility (login root and login bin only)
- permit monitoring use of the Help Facility (login root and login bin only)

The user may make one of the above choices by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

If one of the first three choices is chosen, then the user is prompted for additional information; specifically, which *startup* screen, *glossary* term definition, or command description is to be modified. The user may also be prompted for information to identify whether the changes to the database are additions, modifications, or deletions. If the user is modifying existing data or adding new data, then they are prompted to make the appropriate modifications/additions. If the user is deleting a *glossary* term or a command from the database, then they must respond affirmatively to the next query in order for the deletion to be done. In any case, before the user's changes are final, they must respond affirmatively when asked whether they are sure they want their requested database changes to be done.

By default, *helpadm* will put the user into *ed*(1) to make additions/modifications to database information. If the user wishes to be put into a different editor, then they should set the environment variable **EDITOR** in their environment to the desired editor, and then export **EDITOR**.

If the user chooses to monitor/prevent monitoring use of the Help Facility, the choice made is acted on with no further interaction by the user.

SEE ALSO

ed(1), glossary(1), help(1), locate(1), starter(1), usage(1).

WARNINGS

When the CTIX system is delivered to a customer, **/etc/profile** exports the environment variable **LOGNAME**. If **/etc/profile** has been changed so that **LOGNAME** is not exported, then the options to monitor/prevent monitoring use of the Help Facility may not work properly.

FILES

HELPLOG	/usr/lib/help/HELPLOG
helpclean	/usr/lib/help/helpclean

NAME

hinv - hardware inventory

SYNOPSIS

/etc/hinv option

/etc/hinv hardware-item

DESCRIPTION

The *hinv* command provides hardware configuration information. There are two forms of the command: in the first form, an *option* is given and the result is printed on *stdout*; in the second form, a particular hardware item is specified, and *hinv* exits with **1** if it exists, or with **0** otherwise.

Option is one of the following:

-p	Print hardware configuration. Items are printed one per line.
-c	Print CPU type.
-f	Print FPU type.
-s	Print system type.
-u	Returns a meaningless value of 128; included for compatibility only.
-m	Print total physical memory in bytes.

Hardware-item is one of the following:

68881	68881 floating-point processor.
iop	Terminal accelerator board.
422	Any RS-422 cluster board.
422-2	Two-channel RS-422 cluster board.
422-4	Four-channel RS-422 cluster board.
vme	VME interface board.
s*n*	RS-232 board *n*.
scsi	A SCSI interface is present.
S0	On-board SCSI is present.
S*n*	SCSI Combo board *n*.
ipt	Interphase tape controller is in EEPROM.
smd	Interphase SMD controller is in EEPROM.

mpcc	Multiprotocol Communications Controller is in EEPROM.
serial	Gives number of serial ports present.
disks	Gives number of disks present.
eeprom	VME EEPROM valid for UNIX.
enet	Ethernet Combo Board is present or a CMC Ethernet board is in EEPROM.
cmcenp	CMC Ethernet board is in EEPROM.
E*n*	Ethernet Combo board *n*.

BUGS

The *hinv* command does not know about VME cards.

NAME

hostid - set or print identifier of current host system

SYNOPSIS

hostid [identifier]

DESCRIPTION

The *hostid* command prints the identifier of the current host in hexadecimal. This numeric value is expected to be unique across all hosts and is commonly set to the host's Internet address. The super-user can set the hostid by giving a hexadecimal argument or the hostname; to do this automatically when the system is rebooted, add the command to **/etc/rcopts/NETWORK**.

SEE ALSO

gethostid(2), sethostid(2).

NAME

hostname - set or print the Internet host name of the current system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

Without arguments, the *hostname* command prints the name of the current host. When executed by the superuser, *hostname* sets the hostname to the value of the argument *nameofhost*.

hostname is normally used for setting the hostname to the full Internet name, as in

jack-src.MyCompany.COM

Systems running CTIX Internetworking should, in fact, set the hostname to what they consider to be the full Internet name.

CTIX understands the hostname, when set to the full Internet name, as consisting of two components: the left-most qualifier, and everything to the right of the "." following the first component. (In the example above, "jack-src" is the first component and "MyCompany.COM" the second component.) This division into two components makes possible an important interaction between the *hostname* command and the *setuname* and *uname* commands: *hostname* resets the system node name to the value of the left-most qualifier, and correspondingly, *setuname* resets the first component of the hostname. Thus, the node name and first component of the hostname are always identical.

The left-most qualifier can not be greater than eight characters; the rest of the hostname can not be greater than 53 characters (excluding the left-most ".").

When the system is rebooted, if there is a non-zero length file **/etc/rcopts/NODE** and a non-zero length file **/etc/rcopts/INET-DOMAIN**, it does a *hostname* command using the name specified in the NODE file for the left-most qualifier and concatenating to it a "." and the value specified in the INET-DOMAIN file. If there is only a NODE file, it does a *hostname* command using the NODE file value. If neither file exists, it sets the hostname to the value returned by **uname -n**. (To build the hostname contained in the example above, **/etc/rcopts/NODE** would specify **jack-src**, and **/etc/rcopts/INET-DOMAIN** would specify **MyCompany.COM**.)

FILES

/etc/rcopts/NODE
/etc/rcopts/INET-DOMAIN
/etc/rc2

SEE ALSO

uname(1), setuname(1M), gethostname(2), sethostname(2).

NAME

hp - handle special functions of Hewlett-Packard terminals

SYNOPSIS

hp [**-e**] [**-m**]

DESCRIPTION

The *hp* command supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff* output. A typical use is shown below:

```
nroff -h files ... | hp
```

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal supports the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it supports the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags to *hp* are as follows:

-e The terminal is assumed to support the "display enhancements" feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the underline mode. Superscripts are shown in half-bright mode, and subscripts in half-bright underlined mode. If the **-e** flag is omitted, *hp* assumes that your terminal lacks the "display enhancements" feature: all overstruck characters, subscripts, and superscripts are displayed in reverse video mode (that is, dark-on-light, rather than the usual light-on-dark).

-m Requests minimization of output by removal of new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; that is, any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set provided by *300*(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown.

DIAGNOSTICS

line too long

The representation of a line exceeds 1024 characters.

The *hp* command returns a **0** exit code for normal termination, a **2** for all errors.

SEE ALSO

300(1), col(1), eqn(1), greek(1), nroff(1), tbl(1).

BUGS

An "overstriking sequence" is defined as a printing character, followed by a backspace, followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in reverse video; otherwise, only the first printing character is shown (again, underlined or in reverse video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (for example, reverse line-feeds, backspaces) can make text disappear; in particular, tables generated by *tbl*(1) that contain vertical lines are often missing the lines of text that contain the "foot" of a vertical line, unless the input to *hp* is piped through *col*(1).

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hpio - Hewlett-Packard 2645A terminal tape file archiver

SYNOPSIS

hpio -o[rc] file ...

hpio -i[rta] [**-n** count]

DESCRIPTION

The *hpio* command is designed to take advantage of the tape drives on Hewlett-Packard 2645A terminals. Up to 255 CTIX system files can be archived onto a tape cartridge for off-line storage or for transfer to another UNIX or CTIX system. The actual number of files depends on the sizes of the files. One file of about 115,000 bytes will almost fill a tape cartridge. Almost 300 1-byte files will fit on a tape, but the terminal will not be able to retrieve files after the first 255. This manual page is not intended to be a guide for using tapes on Hewlett-Packard 2645A terminals, but tries to give enough information to be able to create and read tape archives and to position a tape for access to a desired file in an archive.

hpio **-o** (copy out) copies the specified *file*(s), together with path name and status information to a tape drive on your terminal (which is assumed to be positioned at the beginning of a tape or immediately after a tape mark). The left tape drive is used by default. Each *file* is written to a separate tape file and terminated with a tape mark. When *hpio* finishes, the tape is positioned following the last tape mark written.

hpio **-i** (copy in) extracts a file(s) from a tape drive (which is assumed to be positioned at the beginning of a file that was previously written by a *hpio* **-o**). The default action extracts the next file from the left tape drive.

hpio always leaves the tape positioned after the last file read from or written to the tape. Tapes should always be rewound before the terminal is turned off. To rewind a tape depress the green function button, then function key 5, and then select the appropriate tape drive by depressing either function key 5 for the left tape drive or function key 6 for the right. If several files have been archived onto a tape, the tape may be positioned at the beginning of a specific file by depressing the green function button, then function key 8, followed by typing the desired file number (1-255) with no RETURN, and finally function key 5 for the left tape or function key 6 for the right. The desired file number may also be specified by a signed number relative to the current file number.

The meanings of the available options are:

r Use the right tape drive.

c Include a checksum at the end of each *file*. The checksum is always checked by *hpio* **-i** for each file written with this option by *hpio* **-o**.

n *count* The number of input files to be extracted is set to *count*. If this option is not given, *count* defaults to 1. An arbitrarily large *count* may be specified to extract all files from the tape. *hpio* will stop at the end of data mark on the tape.

t Print a table of contents only. No files are created. Printed information gives the file size in bytes, the file name, the file access modes, and whether or not a checksum is included for the file.

a Ask before creating a file. *hpio* **-i** normally prints the file size and name, creates and reads in the file, and prints a status message when the file has been read in. If a checksum is included with the file, it reports whether the checksum matched its computed value. With this option, the file size and name are printed followed by a **?**. Any response beginning with **y** or **Y** will cause the file to be copied in as above. Any other response will cause the file to be skipped.

FILES

/dev/tty*???* to block messages while accessing a tape

SEE ALSO

cu(1C).

DIAGNOSTICS

BREAK

An interrupt signal terminated processing.

Can't create 'file'.

File system access permissions did not allow *file* to be created.

Can't get tty options on stdout.:

hpio was unable to get the input-output control settings associated with the terminal.

Can't open 'file'.

File could not be accessed to copy it to tape.

End of Tape.

No tape record was available when a read from a tape was requested. An end of data mark is the usual reason for this, but it may also occur if the wrong tape drive is being accessed and no tape is present.

'file' not a regular file.

File is a directory or other special file. Only regular files will be copied to tape.

Readcnt = rc, termcnt = tc.

hpio expected to read *rc* bytes from the next block on the tape, but the block contained *tc* bytes. This is caused by having the tape improperly positioned or by a tape block being mangled by interference from other terminal I/O.

Skip to next file failed.

An attempt to skip over a tape mark failed.

Tape mark write failed.

An attempt to write a tape mark at the end of a file failed.

Write failed.

A tape write failed. This is most frequently caused by specifying the wrong tape drive, running off the end of the tape, or trying to write on a tape that is write protected.

WARNINGS

Tape I/O operations may copy bad data if any other I/O involving the terminal occurs. Do not attempt any type ahead while *hpio* is running. *hpio* turns off write permissions for other users while it is running, but processes started asynchronously from your terminal can still interfere. The most common indication of this problem, while a tape is being written, is the appearance of characters on the display screen that should have been copied to tape.

The keyboard, including the terminal BREAK key, is locked during tape write operations; the BREAK key is only functional between writes.

hpio must have complete control of the attributes of the terminal to communicate with the tape drives. Interaction with commands such as *cu*(1C) may interfere and prevent successful operation.

BUGS

Some binary files contain sequences that will confuse the terminal.

An *hpio* **-i** that encounters the end of data mark on the tape (for example, scanning the entire tape with *hpio* **-itn 300**), leaves the tape positioned *after* the end of data mark. If a subsequent *hpio* **-o** is done at this point, the data will not

be retrievable. The tape must be repositioned manually using the terminal FIND FILE -1 operation (depress the green function button, function key 8, and then function key 5 for the left tape or function key 6 for the right tape) before the *hpio* -o is started.

If an interrupt is received by *hpio* while a tape is being written, the terminal may be left with the keyboard locked. If this happens, the terminal's RESET TERMINAL key will unlock the keyboard.

NAME

hyphen - find hyphenated words

SYNOPSIS

hyphen [files]

DESCRIPTION

The *hyphen* command finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* can be used as a filter.

EXAMPLE

The following command allows the proofreading of *nroff* hyphenation in *textfile*:

```
mm textfile | hyphen
```

SEE ALSO

mm(1), nroff(1).

BUGS

The *hyphen* command does not deal well with hyphenated *italic* (that is, underlined) words; it often misses them completely or garbles them.

The *hyphen* command gets confused occasionally, but with no ill effects other than spurious extra output.

NAME

id - print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

id outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1), getuid(2).

NAME

idload - Remote File Sharing user and group mapping

SYNOPSIS

idload [**-n**] [**-g** g_rules] [**-u** u_rules]
[directory]
idload -k

DESCRIPTION

The *idload* command is used on Remote File Sharing (RFS) server machines to build translation tables for user and group IDs. It uses the **/etc/passwd** and **/etc/group** files to produce translation tables for user and group IDs from remote machines, according to the rules set down in the *u_rules* and *g_rules* files. If you are mapping by user and group name, you need copies of remote **/etc/passwd** and **/etc/group** files. If no rules files are specified, remote user and group IDs are mapped to MAXUID (an ID number one higher than the highest number you could assign on your system.)

By default, the remote password and group files are assumed to reside in **/usr/nserve/auth.info/** *domain/nodename*/[**passwd** | **group**]. The *directory* argument indicates that some directory structure other than **/usr/nserve/auth.info** contains the *domain/nodename* **passwd** and **group** files. [*nodename* is the name of the computer the files are from and *domain* is the RFS domain the computer is a member of; see *uname*(1) and *dname*(1M), respectively.]

You must run **idload** to put the mapping into place. Global mapping takes effect immediately for machines that have one of your resources currently mounted. Mapping for other specific machines takes effect when each machine mounts one of your resources.

-n This is used to do a trial run of the ID mapping. No translation table is produced; however, a display of the mapping is output to the terminal (*stdout*).

-k This is used to print the ID mapping currently in use. (Specific mapping for remote machines is not shown until that machine mounts one of your resources.)

-u *u_rules* The *u_rules* file contains the rules for user ID translation. The default rules file is **/usr/nserve/auth.info/uid.rules**.

-g *g_rules* The *g_rules* file contains the rules for group ID translation. The default rules file is **/usr/nserve/auth.info/gid.rules**.

This command is restricted to the super-user.

Rules

The rules files have two types of sections (both optional): **global** and **host**. There can be only one global section, though there can be one host section for each computer you want to map.

The **global** section describes the default conditions for translation for any machines that are not explicitly referenced in a **host** section. If the global section is missing, the default action is to map all remote user and group IDs from undefined computers to MAXUID+1. The syntax of the first line of the **global** section follows:

global

A **host** section is used for each machine or group of machines that you want to map differently from the global definitions. The syntax of the first line of each **host** section follows:

host *name* ...

where *name* is replaced by the full name of a computer (*domain.nodename*).

The format of a rules file is described below. (All lines are optional, but must appear in the order shown.)

global
default *local* | **transparent**
exclude *remote_id-remote_id* | *remote_id*
map *remote_id:local*

host *domain.nodename* [*domain.nodename*...]
default *local* | **transparent**
exclude *remote_id-remote_id* | *remote_id* | *remote_name*
map *remote:local* | *remote* | **all**

Each of these instruction types is described below.

The line:

default *local* | **transparent**

defines the mode of mapping for remote users that are not specifically mapped in instructions in other lines. **transparent** means that all remote user and group IDs have the same numeric value locally unless they appear in the **exclude** instruction. Note that *local* can be replaced by a local user name or ID to map all users into a particular local name or ID number. If the default line is omitted, all users that are not specifically mapped are mapped into a "special guest" login ID.

The line:

exclude *remote_id-remote_id* | *remote_id* | *remote_name*

defines remote IDs to be excluded from the **default** mapping. The **exclude** instruction must precede any **map** instructions in a block. You can use a range of ID numbers, a single ID number, or a single name. (*remote_name* cannot be used in a *global* block.)

The line

map *remote:local* | *remote* | **all**

defines the local IDs and names that remote IDs and names are mapped into. *remote* is either a remote ID number or remote name; *local* is either a local ID number or local name. Placing a colon between a *remote* and a *local* gives the value on the left the permissions of the value on the right. A single *remote* name or ID assigns the user or group permissions of the same local name or ID. **all** is a predefined alias for the set of all user and group IDs found in the local **/etc/passwd** and **/etc/group** files. (You cannot map by remote name in **global** blocks.)

DIAGNOSTICS

On successful completion, *idload translation tables and returns a successful exit status. If idload* fails, the command returns an exit status of zero and does not produce a translation table.

If (1) either rules file cannot be found or opened; (2) there are syntax errors in the rules file; (3) there are semantic errors in the rules file; (4) **host** password or group information could not be found; or (5) the command is not run with super-user privileges, an error message is sent to standard error. Partial failures cause a warning message to appear, although the process continues.

FILES

/etc/passwd
/etc/group
/usr/nserve/auth.info/*domain*/*nodename*/[**user** | **group**]
/usr/nserve/auth.info/uid.rules
/usr/nserve/auth.info/gid.rules

SEE ALSO

mount(1M).
S/Series CTIX Administrator's Guide.

NOTES

The *idload* command always outputs warning messages for **map all**, since password files always contain multiple administrative user names with the same

ID number. The first mapping attempt on the ID number succeeds, and each subsequent attempt produces a warning.

Remote File Sharing doesn't need to be running to use *idload*.

NAME

ifconfig - configure network interface parameters

SYNOPSIS

/etc/ifconfig interface address_family [address [dest_address]]
[parameters] **/etc/ifconfig** interface [protocol_family]

DESCRIPTION

The *ifconfig* command is used to assign an address to a network interface and/or configure network interface parameters. The command is normally used at boot time to define the network address of each interface present on a machine; *ifconfig* can also be used after boot to redefine an interface's address or other operating parameters. The *interface* parameter is a string of the form *nameunit:* for example **en0**.

Since an interface can receive transmissions in differing protocols, each of which might require separate naming schemes, you must specify the *address_family*, which can change the interpretation of the remaining parameters. The only address family currently supported is "inet".

For the DARPA Internet family, the address is a host name present in the host name data base, *hosts*(4N), or a DARPA Internet address expressed in the Internet standard "dot notation."

The following parameters can be set by using *ifconfig*:

up Mark an interface up. This enables an interface that is disabled by using **ifconfig down**. The **ifconfig up** is automatic when the first address on an interface is set. If the interface is reset when marked down, the hardware is reinitialized.

down Mark an interface down. When an interface is marked down, the system does not attempt to transmit messages through that interface. If possible, the interface is reset to disable reception as well. This action does not automatically disable routes using the interface.

trailers Request the use of a trailer link level encapsulation when sending (default). If a network interface supports *trailers*, the system, when possible, encapsulates outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when

sending to this host. Similarly, trailer encapsulations are sent to other hosts that have made such requests. Currently used by Internet protocols only.

-trailers Disable the use of a "trailer" link level encapsulation.

arp Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.

-arp Disable the use of the Address Resolution Protocol.

metric *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol. Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

debug Enable driver dependent debugging code; usually, this turns on extra console error logging.

-debug Disable driver dependent debugging code.

netmask *mask* (Inet only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table *networks*(4). The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

dstaddr Specify the address of the correspondent on the other end of a point to point link.

broadcast (Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's.

ifconfig displays the current configuration for a network interface when no optional parameters are supplied. If a protocol family is specified, *ifconfig* reports only the details specific to that protocol family.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

netstat(1), intro(4).

NAME

includes - determine C language preprocessor include files

SYNOPSIS

includes [option ...] file ...

DESCRIPTION

The *includes* command determines the **#include** files necessary to compile a C language source file using the C language preprocessor *cpp*(1); the command is based on *cpp*(1) and takes the same options. Multiple source files can be named on the command line. However, instead of producing preprocessed code, *includes* produces on standard output a list of the **#include** file dependencies (directly or nested) of the named source files.

The output format is suitable for direct use in a *makefile* to be used by the *make*(1) command. For each named source file, the **#include** files are listed, one per line, preceded by the name of the source file (with the last letter of its name changed to the letter **o**). The two names are separated by a colon and a space : . For example, if source file **pgm.c** depends only on the **#include** file **incl.h**, the output of **includes** for the source file **pgm.c** would be:

pgm.o: incl.h

The following *options* to *includes* are recognized:

-U*name* Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of possibly-reserved symbols includes the following:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, vax, mc68k, mc68000, mc68010, mc68020
system variants:	RES, RT

-D*name*

-D*name=def* Define *name* with value *def* as if by a **#define**. If no *=def* is given, *name* is defined with value 1. The **-D** option has lower precedence than the **-U** option. That is, if the same name is used in both a **-U** option and a **-D** option, the name is undefined, regardless of the order of the options.

-T The **-T** option forces *includes* to use only the first eight characters to distinguish preprocessor symbols and is included for backward compatibility.

-I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the

directories on the standard list. Thus, **#include** files with names enclosed in double quotation marks (" ") are searched for first in the directory of the file with the **#include** line, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files with names enclosed in angle brackets (< >**f1**) **the directory of the file with the #include** line is not searched. By default, *includes* searches for the name enclosed in angle brackets in **/usr/include**; however, if the shell variable INCROOT is set, *includes* prepends the value of INCROOT to the standard list; this is particularly useful for cross-compilation.

-Y*dir* Use directory *dir* in place of the standard list of directories when searching for **#include** files. Use of the **-Y** option overrides the value for INCROOT if it is set.

-H Print, one per line on standard error, the path names of included files.

Two special names are understood by *includes*: **__LINE__** is defined as the current line number (as a decimal integer) as known by *includes*, and **__FILE__** is defined as the current file name (as a C string) as known by *includes*. The special names can be used anywhere (including in macros), just as any other defined name.

All *cpp* directives understood by *includes* start with lines begun by #. The directives are:

#define *name token-string*
Replace subsequent instances of *name* with *token-string*.

#define *name*(*arg*, ..., *arg*) *token-string*
Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *includes* re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#ident *"string"*
This directive has no effect.

#undef *name* Cause the definition of *name* (if any) to be forgotten from now on.

#include *"filename"*
#include *<filename>*
Include at this point the contents of *filename* (which is then run through *includes*). When the *<filename>* notation is used, *filename* is searched for only in the standard places; see the descriptions for the **-I** and **-Y** options for more detail.

#line *integer-constant "filename"*
This directive has no effect.

#endif Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name* The lines following is processed if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name* The lines following is not processed if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*
Lines following is processed if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary **–**, **!**, and **~** operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *includes* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#elif *constant-expression*
An arbitrary number of **#elif** directives is allowed between a **#if**, **#ifdef**, or **#ifndef** directive and a **#else** or **#endif** directive. The lines following the **#elif** directive will appear in the output if and only if the preceding test directive evaluates to zero, all intervening **#elif** directives evaluate to zero, and the *constant-expression* evaluates to non-zero. If *constant-expression* evaluates to non-zero, all succeeding **#elif** and **#else** directives

will be ignored. Any *constant-expression* allowed in a **#if** directive is allowed in a **#elif** directive.

#else The lines following will appear in the output if and only if the preceding test directive evaluates to zero, and all intervening **#elif** directives evaluate to zero.

The test directives and the possible **#else** directives can be nested.

FILES

INCDIR	standard directory list for **#include** files, usually /usr/include
LIBDIR	usually /lib

SEE ALSO

cc(1), cpp(1), m4(1).

DIAGNOSTICS

The error messages produced by *includes* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NAME

inetd - internet "super-server"

SYNOPSIS

/etc/inetd [**-d**] [configuration file]

DESCRIPTION

inetd listens on multiple ports for incoming connection requests. When it receives a request, it spawns the appropriate server. The use of a "super-server" allows other servers to be spawned only when needed and to terminate when they have satisfied a particular request.

The mechanism is as follows: At boot time *inetd* is started if **/etc/rcopts/KINET** is present. To obtain information about the servers it needs to spawn, *inetd* reads its configuration file [by default, this is */etc/inetd.conf*(4)] and issues a call to *getservbyname* [see *getservent*(3)]. (Note that **/etc/services** and **/etc/protocols** must be properly configured.) *inetd* then creates a socket for each server and binds each socket to the port for that server. It does a *listen*(2) on all connection-based sockets (that is, stream rather than datagram), and waits, using *select*(2), for a connection or datagram.

- When a connection request is received on a listening (stream) socket, *inetd* does an *accept*(2), thereby creating a new socket. (*inetd* continues to listen on the original socket for new requests). *inetd* forks, dups, and execs the appropriate server, passing it any server program arguments specified in *inetd*'s configuration file. The invoked server has I/O to *stdin*, *stdout*, and *stderr* done to the new socket; this connects the server to the client process. (Some "built-in," internal services are performed via function calls rather than child processes.)
- When there is data waiting on a datagram socket, *inetd* forks, dups, and execs the appropriate server, passing it any server program arguments; unlike a connection-based server, a datagram server has I/O to *stdin*, *stdout*, and *stderr* done to the original socket. If the datagram socket is marked as "wait" (this corresponds to an entry in *inetd*'s configuration file), the invoked server must process the message before *inetd* considers the socket available for new connections. If the datagram socket is marked as "nowait," *inetd* continues to process incoming messages on that port. *tftpd* is an exceptional case: although its entry in *inetd*'s configuration file must be "wait" (this is to avoid contention for the port), *inetd* is able to continue processing new messages on the port.

The following servers may be started by *inetd*: *fingerd*, *ftpd*, *ouucpd*, *rexecd*, *rlogind*, *rshd*, *talkd*, *telnetd*, *tftpd*, and *uucpd*. *inet* must also start several internal services: these are described in *inetd.conf*(4). Do **not** arrange for *inetd* to start *named*, *routed*, *rwhod*, *sendmail*, *slipd*, *listen* (RFS listening server), or any NFS server.

inetd rereads its configuration file when it receives a hangup signal, SIGHUP. Services may be added, deleted or modified when the configuration file is reread.

The **-d** option turns on socket-level debugging and prints debugging information to *stdout*.

FILES

/etc/inetd.conf
/etc/protocols
/etc/services

SEE ALSO

fingerd(1M), ftpd(1M), rexecd(1M), rlogind(1M), rshd(1M), talkd(1M), telnetd(1M), tftpd(1M), uucpd(1M), inetd.conf(4), protocols(4), services(4).
CTIX Network Administrator's Guide.

NAME

infocmp - compare or print out terminfo descriptions

SYNOPSIS

infocmp [**-d**] [**-c**] [**-n**] [**-I**] [**-L**] [**-C**] [**-r**] [**-u**] [**-s d**i[**-v**] [**-V**] [**-1**] [**-w** width] [**-A** directory] [**-B** directory] [termname ...]

DESCRIPTION

The *infocmp* command can be used to compare a binary *terminfo*(4) entry with other terminfo entries, rewrite a *terminfo*(4) description to take advantage of the **use=** terminfo field, or print out a *terminfo*(4) description from the binary file [*term*(4)] in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **-I** option will be assumed. If more than one *termname* is specified, the **-d** option will be assumed.

Comparison Options [-d] [-c] [-n]

infocmp compares the *terminfo*(4) description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **-1** for integer variables, and **NULL** for string variables.

-d produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.

-c produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **-u** option is worth using.

-n produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable **TERM** will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [-I] [-L] [-C] [-r]

The **-I**, **-L**, and **-C** options will produce a source listing for each terminal named.

-I use the *terminfo*(4) names

-L use the long C variable name listed in **<term.h>**

-C use the *termcap* names

-r when using **-C**, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the **-C** option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. *infocmp* will attempt to convert most of the parameterized information, but that which it doesn't will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo*(4), but which are derivable from other *terminfo*(4) variables, will be output. Not all *terminfo*(4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **-r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo*(4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo*(4) format will not necessarily reproduce the original *terminfo*(4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences, are:

Terminfo	*Termcap*	*Representative Terminals*
%p1%c	%.	adm
%p1%d	%d	hp, vt100, ANSI standard
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [-u]

-u produce a *terminfo*(4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler *tic*(1M) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use=** entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. *infocmp* will flag any other *termname* **use=** fields that were not needed.

Other Options [-s d|i|l|c] [-v] [-V] [-1] [-w width]

-s sort the fields within each type according to the argument below:

d leave fields in the order that they are stored in the *terminfo* database.

i sort by *terminfo* name.

l sort by the long C variable name.

c sort by the *termcap* name.

If no **-s** option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **-C** or the **-L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

-v print out tracing information on standard error as the program runs.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to printed out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

-w change the output to width characters.

Changing Databases [-A directory] [-B directory]

The location of the compiled *terminfo*(4) database is taken from the environment variable **TERMINFO**. If the variable is not defined, or the terminal is not found in that location, the system *terminfo*(4) database, usually in */usr/lib/terminfo*, will be used. The options **-A** and **-B** may be used to override this location. The **-A** option will set **TERMINFO** for the first *termname* and the **-B** option will set **TERMINFO** for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo*(4) database for a comparison to be made.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

DIAGNOSTICS

malloc is out of space!

There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use= order dependency found:

A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=term' did not add anything to the description.

A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison to be done.

The **-u**, **-d** and **-c** options require at least two terminal names.

SEE ALSO

captoinfo(1M), tic(1M), curses(3X), term(4), terminfo(4).
UNIX System V Release 3.2 Programmer's Guide.

NOTE

The *termcap* database (from earlier CTIX releases) may not be supplied in future releases.

NAME

init, telinit - process control initialization

SYNOPSIS

/etc/init [0123456SsQq]

/bin/telinit [0123456sSQqabc]

DESCRIPTION

Init

init is a general process spawner. Its primary role is to create processes from information stored in the file **/etc/inittab** [see *inittab*(4)]. This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system. On S/MT systems, the kernel runs *init* as the last step in the boot procedure.

init considers the system to be in a *run-level* at any given time. A *run-level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run-levels* is defined in the *inittab* file. *init* can be in one of eight *run-levels*, **0-6** and **S** or **s**. The *run-level* is changed by having a privileged user run **/etc/init**. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was rebooted, telling it which *run-level* to change to.

init is invoked inside the CTIX system as the last step in the boot procedure. First *init* looks in **/etc/inittab** for the *initdefault* entry [see *inittab*(4)]. If there is one, *init* uses the *run-level* specified in that entry as the initial *run-level* to enter. If this entry is not in */etc/inittab*, *init* requests that the user enter a *run-level* from the virtual system console. If an **S** or an **s** is entered, *init* goes into the SINGLE USER state. This is the only *run-level* that doesn't require the existence of a properly formatted */etc/inittab* file. If it doesn't exist, then by default the only legal *run-level* that *init* can enter is the SINGLE USER state. In the SINGLE USER state the virtual console terminal **/dev/syscon** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the *SINGLE USER run-level* one of two options can be elected. First, if the shell is terminated (via an end-of-file), *init* will reprompt for a new *run-level*. Second, the *init* or *telinit* command can signal *init* and force it to change the *run-level* of the system. *init* always trys to relink */dev/syscon* to a reasonable terminal before opening it. It invokes *conlocate*(1M) to do this.

When *init* prompts for the new *run-level*, the operator may enter only one of the digits **0** through **6** or the letters **S** or **s**. If **S** or **s** is entered, *init* operates as previously described in the SINGLE USER state with the additional result that

/dev/syscon is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the previous system console, saying where the virtual terminal has been relocated.

When *init* comes up initially and whenever it switches out of SINGLE USER state to normal run states, it sets the *ioctl*(2) states of the virtual console, **/dev/syscon**, to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever the SINGLE USER state is entered.

If a **0** through **6** is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be re-prompted. Note that the *run-levels* **0**, **1**, **4**, **5**, and **6** are reserved single-user states (**6** has a very special purpose, as described below); the *run-levels* **2** and **3** are reserved multi-user states.

If this is the first time *init* has entered a *run-level* other than SINGLE USER, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run-level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run-level*.

Run-level **2** is defined to contain all of the terminal processes and daemons that are spawned in the multi-user environment. Hence, it is commonly referred to as the MULTI-USER state. *Run-level* **2** also stops remote file sharing. *Run-level* **3** is defined to start up remote file sharing processes and daemons as well as mounting and advertising remote resources. So, *run-level* **3** extends multi-user mode and is known as the Remote File Sharing state.

In a MULTI-USER environment, the *inittab* file is set up so that *init* will create a process for each terminal on the system that the administrator sets up to respawn.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists [see *who*(1)]. A history of the processes spawned is kept in **/etc/wtmp.**

To spawn each process in the *inittab* file, *init* reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's *run-level*. When one of these conditions occurs,

init re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however, *init* still waits for one of the above three conditions to occur. To get around this, **init Q** or **init q** command wakes *init* to re-examine the *inittab* file immediately.

If *init* receives a *powerfail* signal (*SIGPWR*) it scans *inittab* for special entries of the type *powerfail* and *powerwait*. These entries are invoked (if the *run-levels* permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions during the powerdown of the operating system.

When *init* is requested to change *run-levels* (via *telinit*), *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target *run-level*. *init* waits 5 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

Telinit

Telinit, which is linked to */etc/init*, is used to direct the actions of *init*. It takes a one-character argument and signals *init* via the *kill* system call to perform the appropriate action. The following arguments serve as directives to *init*.

0-6 tells *init* to place the system in one of the *run-levels* **0-6**.

a,b,c tells *init* to process only those **/etc/inittab** file entries having the **a**, **b** or **c** *run-level* set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current *run-level* to change. (Note that the correct *action* field value in this case is **ondemand**, rather than **respawn**. [See *inittab*(4).])

Q,q tells *init* to re-examine the **/etc/inittab** file.

s,S tells *init* to enter the single user environment. When this level change is effected, the virtual system terminal, **/dev/syscon**, is changed to the terminal from which the command was executed. Note that running the *halt* or *shutdown*(1M) program is the preferred way of bringing the system to single user state.

A directive to change to *run-level* **6** receives special priority. Ordinarily, a *run-level* change received while *init* is re-examining *inittab* does not take effect until the re-examination is complete. But a directive to change to *run-level* **6** received while *init* is waiting on a *bootwait* entry is effected as soon as the command in the *bootwait* entry finishes. This special case permits a *bootwait* command to use

telinit to stop the system initialization process before users get access to the system. *Run-level* **6** then handles the transition to single-user state: see **/etc/profile**.

FILES

/etc/bcheckrc
/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/syscon
/etc/profile

SEE ALSO

conlocate(1M), getty(1M), login(1), sh(1), who(1), kill(2), inittab(4), profile(4), utmp(4).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

If *init* finds that it is respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.

WARNINGS

Telinit can be run only by someone who is super-user or a member of group sys.

NAME

install - install commands

SYNOPSIS

/etc/install [**-c** dira] [**-f** dirb] [**-i**] [**-n** dirc] [**-m** mode] [**-u** user] [**-g** group] [**-o**] [**-s**] file [dirx ...]

DESCRIPTION

The *install* command is most commonly used in "makefiles" [See *make*(1)] to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, *install* will search a set of default directories (**/bin**, **/usr/bin**, **/etc**, **/lib**, and **/usr/lib**, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

-c *dira* Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **-s** option.

-f *dirb* Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to **755** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **-o** or **-s** options.

-i Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options except **-c** and **-f**.

-n *dirc* If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options except **-c** and **-f**.

-m *mode* The mode of the new file is set to *mode*. Only available to the superuser.

-u *user* The owner of the new file is set to *user*. Only available to the superuser.

-g *group* The group ID of the new file is set to *group*. Only available to the superuser.

-o If *file* is found, this option saves the "found" file by copying it to OLD*file* in the directory in which it was found. This option is useful when installing a frequently used file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options except **-c**.

-s Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

make(1).

NAME

ipcrm - remove a message queue, semaphore set or shared memory ID

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

-q *msqid* removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

-m *shmid* removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-s *semid* removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

-Q *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

-M *shmkey* removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-S *semkey* removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

NAME

ipcs - report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

The *ipcs* command prints information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information display is controlled by the following *options*:

-q Print information about active message queues.

-m Print information about active shared memory segments.

-s Print information about active semaphores.

If any of the options **-q**, **-m**, or **-s** are specified, information about only those indicated are printed. If none of these three are specified, information about all three are printed subject to these options:

-b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

-c Print creator's login name and group name. See below.

-o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

-p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

-t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.

-a Use all print *options*. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

-C *corefile*
Use the file *corefile* in place of **/dev/kmem**.

-N *namelist*
The argument is taken as the name of an alternate *namelist* (**/unix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities are listed.

T (all) Type of the facility:

- **q** message queue;
- **m** shared memory segment;
- **s** semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

- **R** if a process is waiting on a *msgrcv*;
- **S** if a process is waiting on a *msgsnd*;
- **D** if the associated shared memory segment has been removed. It disappears when the last process attached to the segment detaches it;
- **C** if the associated shared memory segment is to be cleared when the first attach is executed;
- **-** if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the

first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r if read permission is granted;
w if write permission is granted;
a if alter permission is granted;
- if the indicated permission is *not* granted.

OWNER (all) The login name of the owner of the facility entry.

GROUP (all) The group name of the group of the owner of the facility entry.

CREATOR (a,c) The login name of the creator of the facility entry.

CGROUP (a,c) The group name of the group of the creator of the facility entry.

CBYTES (a,o) The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o) The number of messages currently outstanding on the associated message queue.

QBYTES (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p) The process ID of the last process to send a message to the associated queue.

LRPID (a,p) The process ID of the last process to receive a message from the associated queue.

STIME (a,t) The time the last message was sent to the associated queue.

RTIME (a,t) The time the last message was received from the associated queue.

CTIME (a,t) The time when the associated entry was created or changed.

NATTCH (a,o) The number of processes attached to the associated shared memory segment.

SEGSZ (a,b) The size of the associated shared memory segment.

CPID (a,p) The process ID of the creator of the shared memory entry.

LPID (a,p) The process ID of the last process to attach or detach the shared memory segment.

ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2).

WARNING

If the user specifies either the -C or -N flag, the real and effective UID/GID is set to the real UID/GID of the user invoking *ipcs*.

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

iv - initialize and maintain volume

SYNOPSIS

iv -iuostdwlvq special [descriptionfile]

DESCRIPTION

The *iv* command initializes and maintains a disk volume. *Special* and *descriptionfile*, described below, specify the disk and a description file for the disk volume. The *iv* command performs one of five operations, specified by the following options:

-i Completely initialize a volume. This consists of five phases:

1. Initialize *iv's* internal Volume Home Block, based on *descriptionfile* and the disk type. If the disk can support bad block handling, create an internal Bad Block Table. Put bad block data from *descriptionfile* and volume's existing Bad Block Table (if any) in internal Bad Block Table.
2. Format medium.
3. Perform a surface check. If the disk can support bad block handling, add bad blocks to the Bad Block Table. If the disk cannot support bad block handling, the first bad spot causes the disk to be rejected.
4. Write out the Volume Home Block. This has the effect of dividing the volume into slices (partitions).
5. Allocate and write out the files that share the Reserved Area (slice 0) with the Volume Home Block. If the disk can support bad block handling, one of these files is the Bad Block Table. Other files are specified in *descriptionfile*.

-u Update the Volume Home Block. This is the same as **-i**, except that the second and third phases (medium formatting and surface check) are skipped.

-o Output a Volume Home Block and partition 0 to any file; requires a *descriptionfile*. The following command produces a dump tape:

```
iv -o /dev/rmt0 /usr/lib/iv/desc.tdump
```

-s Surface test. Any bad blocks discovered are added to the Bad Block Table.

-t Tell volume description. Display volume home block in human-readable form. No description file is needed. The volume's contents are not affected.

-d Description file display. A description file that describes the current state of the volume is written to the standard output. If the Reserved Area contains a loader, the **loader** keyword's value is written as **/usr/lib/iv/loader**. If the Reserved Area contains a down load image area, the Down Load Area Description lists files whose names are of the following form:

/usr/lib/iv/ws*xxx.yyy*

where *xxx* is the numeric device identification; *yyy* is **422** if *xxx* is even, **232** if *xxx* is odd.

The **-f** option, equivalent to **-u**, is provided for compatibility with older versions of *iv*. It should not be used, as it may disappear in future releases.

In addition to the single operation option (**-i**, **-u**, **-s**, **-t**, or **-d**) you can specify any or all of the following options:

-v Verbose display output. If the display includes the Volume Home Block, also include the Bad Block Table.

-l A normal surface test consists of a single pass over the disk; **-l** specifies ten passes.

-w A normal surface test pass consists of a read pass; **-w** specifies a write pass before each read pass.

-q Print the size of the disk (in megabytes).

File Parameters

Special is the character special file for slice zero on the drive. This name takes the form **/dev/rdsk/c***n***d***t***s0**, where *n* is the controller number and *t* is the drive number.

Descriptionfile is a text file that describes the volume. It is required by the **-i** and **-u** options. The description file consists of five parts:

- General Description
- Reserved Area Description
- Bad Blocks Description
- Partition Table Description
- Down Load Area Description

Each description is separated from the next by a line that contains only a single dollar sign ($). Specifics for each of the five descriptions are given under separate headings below.

General Description

Each line in the General Description begins with a keyword. Some keywords are followed by values; the value is separated from the keyword by spaces or tabs. For example:

```
ecc
cylinders       1024
```

Each keyword is used only once; valid keywords follow:

type Mandatory, unless the volume is already initialized in the appropriate format. Value is disk type: **HD** for onboard ST506 hard disk; **RD** for RAM disk; V3200 for SMD controller; SCSI for SCSI disk; and FD for floppy disk.

name Mandatory, unless the volume is already initialized in the appropriate format. Value is the volume name. Any characters except spaces or tabs are permitted in the volume name; the serial number of the disk is the recommended volume name. The actual name in the Volume Home Block is always exactly six characters; *iv* right truncates names that are too long and right pads with nulls names that are too short.

cylinders Mandatory (for all disks except SCSI), unless the volume is already initialized in the appropriate format. Value is the number of cylinders on the disk.

heads Mandatory (for all disks except SCSI), unless the volume is already initialized in the appropriate format. Value is the number of heads on the disk.

sectors Mandatory (for all disks except SCSI), unless the volume is already initialized in the appropriate format. Value is the number of physical sectors per track.

lsectors The number of logical 512-byte sectors on a SCSI disk. If this value is not supplied, the total number of available logical sectors on the device is used.

formatextra The SMD drive is formatted with an extra sector on each track. (This sector is ignored by CTIX but is required for some disk drives, notably the Eagle-XP.)

steprate Mandatory for ST506, unless the volume is already initialized in the appropriate format. Value is a number that is passed to the disk controller. The normal steprate for ST506 drives is 14; 0 can be used for slower drives. See the disk manufacturer's documentation for further information.

exchangeable If this keyword is present, the disk can be removed from its drive.

hitech (ST506 drives only) If this keyword is present, the reduced write current line to the disk is used for head-select bit 3 to allow more than eight heads.

precomp (ST506 drives only) The value is *c*/16, where *c* is the cylinder at which precompensation should start. See the disk manufacturer's documentation for further information.

ecc The disk has been prepared to function in ECC mode.

gap1
gap2 Gap size for SMD drives. See the disk manufacturer's documentation for further information.

unformattedbytes
The number of unformatted bytes per sector. This value is required if the "cyl head offset" format is used for the bad block table entries.

Reserved Area Description

The Reserved Area Description describes the files that share slice zero with the volume home block. Each line in the Reserved Area Description consists of a keyword followed by one or more parameters; one or more tabs or spaces separates keywords and parameters from each other. Here are the valid keywords and their meanings. (A logical block is 1024 bytes long.)

loader Describes the loader area. The first, mandatory, parameter is the full pathname of an a.out file to put in the loader area. The second, optional, parameter is the size of the loader area in logical blocks. If the second parameter is missing, the size of the a.out file is used.

badblocktable Describes the Bad Block Table. The first, mandatory, parameter is the size of the Bad Block Table in logical blocks. The second, optional, parameter is only used when an existing Bad Block Table contains errors; this parameter is *empty* to clear the Bad Block Table, missing otherwise.

dump Describes area to contain dump after crash. The only, mandatory, parameter, specifies the size of the dump area in logical blocks.

downloadarea Describes area to contain system images for downloading. The only, mandatory, parameter, specifies the size of the download area in logical blocks. (The files actually put in this area are described separately. See the Down Load Area Description heading, below.)

All lines valid for the Reserved Area Description are optional. However, the Bad Block Table is mandatory on a volume which supports bad block handling (other than SCSI); the loader area is mandatory on a volume which is to hold an operating system; and a dump area is recommended on a volume which is to hold an operating system.

Bad Block Description

The Bad Block Description explicitly specifies up to 889 bad blocks to be added to the Bad Block Table. The *iv* command merges specified bad block information with information already in the Bad Block Table (if there already is one) and bad block information discovered through the surface test.

Each bad block entry is a single line. There are three forms:

sector

where *sector* is a physical sector number;

cylinder head offset

where *cylinder* is a cylinder number, *head* is a head number, and *offset* is the byte offset of the bad spot;

cylinder head sector

where *cylinder* is a cylinder number, *head* is a head number, and *sector* is a physical sector number of the bad spot. The third form is selected by placing the keyword **sector** on the line preceding the first entry of this type in the bad block description. Entries using the third form must come last in the list of bad blocks. All three forms condemn a single sector; the second form condemns the sector that contains the specified byte.

The last sector on each track serves as a bad block alternate. *iv* chooses the alternates in a way that minimizes extra seeking for alternate blocks.

Partition Table Description

The Partition Table Description specifies where the slices (partitions) on the disk are to begin and end. Each line in the Description specifies the starting

logical block of a slice. Start blocks must be on track boundaries, except in the case of SCSI drives, where start blocks need only be on a logical block boundary.

Except for overlapping partitions, slices must be listed in ascending numeric order, and the beginning of a slice defines the end of the previous slice.

It is possible to specify overlapping partitions, although care must be taken in doing so. A $ following any block number indicates that the slice extends to the end of the disk, beyond the next boundary number. Any slice with a starting block number that is larger than its successor must extend to the end of the disk (and must therefore be followed by the $ parameter).

For example, the following description specifies five slices; the fifth slice extends from the second slice to the end of the disk:

```
0
16
20016
40016   $
16      $
```

The following example is also possible, although of doubtful utility:

```
0
16
20016   $
40016   $
16
30016   $
```

In this example six slices are specified. The third, fourth, and sixth slices extend to the end of the disk. The fifth slice, however, starts at 16 and ends at 30015 (inclusive); it includes all the second slice, but only part of the third slice.

The first logical block boundary number in the Description must always be 0. The last slice in the Description always extends to the end of the disk ($ is optional).

There can be at most 16 slices on a disk.

It is a fatal error to specify a slice 1 that does not leave enough room in slice 0 for the Volume Home Block and the slice 0 files.

Down Load Area Description

The Down Load Area Description specifies system images to be included in the Down Load Area. Each line in the Description consists of a numeric device

identification and the full path name of the file to be copied into the down load area; the two parts of the line are separated by one or more spaces or tabs.

EXAMPLES

The following example shows a disk description file for a nonbootable disk (bad blocks expressed in "cylinder/head/sector" format):

```
#     MAXTOR 85 MB disk
type                HD
name                Serno
cylinders           1024
heads               8
sectors             17
steprate            14
hitech
ecc
$
badblocktable       1
$
sector
15          5       4
$
0
8
$
$
```

The following file describes a bootable SMD (bad blocks expressed in "cylinder/head/offset" format):

```
type                V3200
name                Serno
cylinders           1489
heads               11
sectors             33
ecc
gap1                16
gap2                16
unformattedbytes    620
$
badblocktable       3
dump                1024
```

```
downloadarea    300
loader          /usr/lib/iv/loader   128
$
12      3       405
187     9       1010
692     4       5228
66      2       657
985     5       3398
$
0
1456
17360
25360
45360
85360
125360
165360  $
$
100     /usr/lib/iv/ws100.422
200     /usr/lib/iv/ws200.422
$
```

The following file describes a bootable Hitachi drive. (bad blocks expressed as physical sector numbers):

```
type            HD
name            Serno
cylinders       823
heads           10
sectors         17
steprate        14
hitech
ecc
$
badblocktable   1
dump            1024
downloadarea    300
loader          /usr/lib/iv/loader   128
$
1048
2441
```

```
5064
15119
15678
23533
23534
42091
43918
60466
60467
$
0
1456
17730
25922
46402   $
$
100     /usr/lib/iv/ws100.422
200     /usr/lib/iv/ws200.422
$
```

The following file describes a drive without a dump area (no bad blocks specified):

```
type            HD
name            Serno
cylinders       645
heads           7
sectors         17
steprate        14
precomp         80
hitech
ecc
$
badblocktable   1
downloadarea    300
loader          /usr/lib/iv/loader   128
$
$
0
432
12328
```

```
18328   $
$
100     /usr/lib/iv/ws100.422
200     /usr/lib/iv/ws200.422
$
```

FILES

/dev/rdsk/*	disk character special files
/usr/lib/iv/desc.*	prototype description files

SEE ALSO

disk(7).
S/Series CTIX Administrator's Guide.
"WD2010-05 Winchester Disk Controller" in *Storage Management Products Handbook.* Irvine, Calif.: Western Digital Corp., 1984.

NOTES

Any device in physical mode (for example, while surface testing or formatting is being done) is an exclusive open device: use the maintenance tape to reformat or run surface tests on the boot device.

A typical disk has fewer bad spots than the total number of megabytes (a 40 megabyte drive should have fewer than 40 bad spots, and so forth).

WARNINGS

The **-i**, **-u**, and -s operations are dangerous or fatal to existing volume data. Always precede these operations with a backup.

When a new bad block is itself an alternate block, *iv* may produce messages that appear spurious but are actually correct. If the bad block is already in use as an alternate, the message can appear twice for one block.

Do not run *mkfs*(1M) on an overlapping partition.

Do not use Partition Table Descriptions from pre-5.0 versions of CTIX that specify partitions by track numbers, rather than by logical block boundaries.

NAME

join - relational database operator

SYNOPSIS

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 [see *sort*(1)].

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*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following 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. Fields are numbered starting with **1**.

-o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.

-t*c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

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* and *awk*(1) are wildly incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

NAME

kill - terminate a process

SYNOPSIS

kill [-signo] PID ...

DESCRIPTION

The *kill* command sends signal 15 (terminate) to the specified processes. This normally kills processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

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

If a signal number preceded by - is given as first argument, that signal is sent instead of terminate [see *signal*(2)] In particular "kill -9 ..." is an almost sure kill.

SEE ALSO

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

NAME

killall - kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

The *killall* command is used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure. It terminates all processes with open files so that the mounted file systems are unbusied and can be unmounted.

The *killall* command sends *signal* [see *kill*(1)] to all processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), kill(1), ps(1), shutdown(1M), signal(2).

WARNINGS

The *killall* command can be run only by the super-user.

NAME

labelit - provide labels for file systems

SYNOPSIS

/etc/labelit special [fsname volume [**-n**] [**-t**]]

DESCRIPTION

labelit can be used to provide labels for unmounted disk file systems or file systems being copied to tape. The **-n** option provides for initial labeling only (this destroys previous contents). The **-t** option tells *labelit* to treat the device as a tape (put tape headers on the media).

With the optional arguments omitted, *labelit* prints current label values.

The *special* name should be the physical disk section (for example, **/dev/dsk/c0d0s6**), or the cartridge tape (for example, **/dev/rmt/c0d0**). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (for example, **root**, **u1**, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette or tape.

For file systems on disk, *fsname* and *volume* are recorded in the superblock.

SEE ALSO

mkfs(1M), sh(1), fs(4).

NAME

ld - link editor for common object files

SYNOPSIS

ld [options] filename

DESCRIPTION

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *ld* combines the objects, producing an object module that can either be executed or, if the **-r** option is specified, used as input for a subsequent *ld* run. The output of *ld* is left in **a.out**. By default this file is executable if no errors occurred during the load. If any input file, *filename*, is not an object file, *ld* assumes it is either an archive library or a text file containing link editor directives. [See *Link Editor Directives* in the *UNIX System V Programmer's Guide* for a discussion of input directives.]

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. The library may be either a relocatable archive library or a shared library. [See *Shared Libraries* in the *UNIX System V Programmer's Guide* for a discussion of shared libraries.] Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table [see *ar*(4)] is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there exist multiple library members defining the same external symbol.

The following options are recognized by *ld*:

-e *epsym*
: Set the default entry point address for the output file to be that of the symbol *epsym*.

-f *fill*
: Set the default fill pattern for "holes" within an output section as well as initialized *bss* sections. The argument *fill* is a two-byte constant.

-l*x*
: Search a library **lib***x***.a**, where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **-l** is significant. By default, libraries are located in *LIBDIR* or *LLIBDIR*. However, if the shell variable LIBROOT is set, the value of LIBROOT is prepended to LIBDIR and LLIBDIR before searching the libraries.

-m
: Produce a map or listing of the input/output sections on the standard output.

-o *outfile*
Produce an output object file by the name *outfile*. The name of the default object file is **a.out**.

-r Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor will not complain about unresolved references, and the output file will not be executable.

-s Strip line number entries and symbol table information from the output object file.

-t Turn off the warning about multiply-defined symbols that are not the same size.

-u *symname*
Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the *ld* line is significant; it must be placed before the library which will define the symbol.

-x Do not preserve local symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

-Z Do not bind anything to address zero. This option will allow runtime detection of null pointers.

-L *dir* Change the algorithm of searching for **libx.a** to look in *dir* before looking in *LIBDIR* and *LLIBDIR*. This option is effective only if it precedes the **-l** option on the command line.

-M Output a message for each multiply-defined external definition.

-V Output a message giving information about the version of ld being used.

-VS *num*
Use *num* as a decimal version stamp identifying the **a.out** file that is produced. The version stamp is stored in the optional header.

-Y*[LU],dir*
Change the default directory used for finding libraries. If **L** is specified the first default directory which *ld* searches, *LIBDIR*, is replaced by *dir*. If **U** is specified and ld has been built with a second default directory, *LLIBDIR*, then that directory is replaced by *dir*. If ld was

built with only one default directory and U is specified a warning is printed and the option is ignored.

-N Put the text section at the beginning of the text segment rather than after all header information, and put the data section immediately following text in the core image. The result is a plain executable file, indicated by magic number 0407 in the operating system header.

-z Put the data section at the next segment boundary following the text section. The result is a shared text file that permits demand paged execution. This type of file is indicated by magic number 0413 in the operating system header. This option is obsolete.

-F Like -z but takes less disk space and can page faster into memory. This type is also indicated by magic number 0413 in the operating system header. It is distinguished by having virtual text and data starting addresses that are equal to the file offsets of the text and data sections, modulo 4096. The **-F** option is on by default. Note that **-F**,**-N**, and **-z** are mutually exclusive.

-G Change the symbol name look-up algorithm as follows: if two names do not initially match, then if one of them is exactly eight characters, then a match is attempted only on the first eight characters. The purpose of this is to allow compatibility between object modules that have been created with the old C compiler and with the new C compiler, which allows variable names more than eight characters long. A warning message is issued in such cases.

-w If **-G** is used, do not print warnings about symbols that partially matched.

FILES

LIBDIR/libx.a	libraries
LLIBDIR/libx.a	libraries
a.out	output file
LIBDIR	usually /lib
LLIBDIR	usually /usr/lib
LIBDIR/ifile.0407	default -N directive file
LIBDIR/ifile.0413	default -z directive file
LIBDIR/ifile.0413-F	default -F directive file
LIBDIR/ifile.r0407	default -r -N directive file
LIBDIR/ifile.r0413	default -r -z directive file
LIBDIR/ifile.r0413-F	default -r -F directive file

SEE ALSO

as(1), cc(1), mkshlib(1), exit(2), end(3C), a.out(4), ar(4).
UNIX System V Release 3.2 Programmer's Guide.

CAVEATS

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the program's address space.
- When the link editor is called through *cc*(1), a startup routine is linked with the user's program. This routine calls exit( ) [see *exit*(2)] after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls exit( ) rather than falling through the end of the entry routine.

The symbols *etext*, *edata*, and *end* [see *end*(3C)] are reserved and are defined by the link editor. It is incorrect for a user program to redefine them.

If the link editor does not recognize an input file as an object file or an archive file, it will assume that it contains link editor directives and will attempt to parse it. This will occasionally produce an error message complaining about "syntax errors."

Arithmetic expressions may only have one forward referenced symbol per expression.

NAME

lddrv - manage loadable drivers

SYNOPSIS

/etc/lddrv/lddrv -a [**ve**] [**-m** master] [**-o** sfile] devname [subdev ...]

/etc/lddrv/lddrv -A [**ve**] [**-m** master] [**-o** sfile] devname [subdev ...]

/etc/lddrv/lddrv -d [**ve**] [**-m** master] devname

/etc/lddrv/lddrv -b [**ve**] [**-m** master] devname [subdev ...]

/etc/lddrv/lddrv -u [**ve**] [**-m** master] devname

/etc/lddrv/lddrv -q [**v**] [**-m** master] devname

/etc/lddrv/lddrv -s [**v**] [**-m** master]

/etc/lddrv/lddrv -e [**v**] [**-m** master]

DESCRIPTION

lddrv allocates/deallocates space for a specified driver, loads/unloads a specified driver, and returns the status of specified driver(s).

In the discussion below, "driver" refers to a character device driver, a block device driver, a software module, a streams module, a streams driver, or a file system type.

Devname specifies the name of the driver: it must must correspond to the first field in the *master* file (see discussion of **-m** below). Note that the relocatable driver code must be in a file named *devname*.**o**. A *subdev* is the name of a driver, whose code is also in *devname*.**o**, that is to be loaded (bound) or to have space allocated with the driver *devname*. Up to four entries, one *devname* and three *subdev* entries, can be loaded (bound) or have space allocated with a single invocation of *lddrv*. This is normally done with drivers that are interrelated. For example, **/etc/lddrv/cluster.o** contains all relocatable driver code for the **cluster**, **tsp**, **tsy**, and **tst** drivers; and all four drivers are loaded with a single invocation of **lddrv**.)

Options are interpreted as follows:

-v Print verbose information on the screen; when used with **-s**, information about other drivers, already loaded, is displayed.

-m *master*
Use file specified by *master* rather than the default **/etc/master**.

-o *sfile* Put driver's executable code (containing the symbol table for that driver) in file specified by *sfile* rather than in file with the name *devname*, which is the default.

-a Allocate space for and load (bind) the driver.

-A Allocate space for the driver.

-d Unload the driver and deallocate its space.

-b Load (bind) the driver.

-u Unload the driver.

-q Return the status of a particular driver. True if driver successfully loads, or if driver was already loaded; else false.

-s Return the status of all loadable drivers.

-e Create *namelist* (**/etc/lddrv/unix.exec**) from the current *ifile* **unix.sym**.

If the specified driver was already loaded, the **-a**, **-A**, and **-b** options fails.

lddrv maintains two *ifiles*: **unix.sym** and **unix.lnk**. In the case of **unix.sym**, **lddrv -a** adds to this file all symbols from the driver that is being loaded and **lddrv -d** removes from this file all symbols from the driver that is being unloaded. In the case of **unix.lnk**, **lddrv -a** adds to this file only the exported symbols from the driver being loaded and **lddrv -d** removes from this file only the exported symbols from the driver that is being unloaded. The exported symbols are specified in the file **dev.export**, if it exists; if the **export** file does not exist, the only symbols exported are the driver entry points. The **export** file consists of one-line entries containing, one per line, the external symbols to export.

EXAMPLES

A status report for all drivers could look like this:

```
DEVNAME  ID  BLK  CHAR   SIZE    ADDR      FLAGS
   lipc   1   -    -    0x5000 0x3dd000 ALLOC BOUND
   plp    2   -    6    0x1000 0x3e2000 ALLOC BOUND
```

FILES

/etc/master	default master file
/etc/drvtbl	loadable driver table
/etc/lddrv	contains *lddrv* and loadable drivers
/etc/lddrv/unix.sym	ifile for running system
/etc/lddrv/unix.lnk	ifile containing only exported symbols of running system

/etc/lddrv/devname.o	unlinked driver
/etc/lddrv/devname	linked driver
/etc/lddrv/devname.export	list of symbols to export from devname
/etc/lddrv/unix.exec	namelist of running system after lddrv -e has been executed

SEE ALSO

syslocal(2), master(4), drivers(7).

NAME

ldeeprom - load EEPROM

SYNOPSIS

ldeeprom [-s system_file]

DESCRIPTION

When called by **/etc/drvload**, *ldeeprom* loads the electrically erasable, programmable read-only memory on the VME interface card.

The *ldeeprom* command reads the **!FILENAMES**, **!VMESLOTS**, and **!VMECODE** sections of the **/etc/system** file and generates binary data that can be written to the EEPROM. *ldeeprom* command reads two variables in the **!FILENAMES** section:

PROM_IFILE — the file that provides *ld*(1) with relocation information for the **!VMECODE** section.

EEPROM_FILE — the file to which to output the EEPROM binary data. If the file name is **/dev/vme/eeprom**, the contents of this special file are written directly to the EEPROM; if any other file name is given, the contents are written to that file instead.

The *ldeeprom* command reads the **!VMESLOTS** section of **/etc/system** for descriptions of the VME boards.

The *ldeeprom* command reads the **!VMECODE** section of **/etc/system** for the names of object code files. These object files are to be loaded into the EEPROM to provide the boot code for a device where an initialization function name is specified.

The -s option can be used to specify a file to use instead of the **/etc/system** file.

FILES

/dev/vme/eeprom — default **EEPROM** file
/etc/drvload
/etc/system

SEE ALSO

system(4), vme(7).
S/Series CTIX Administrator's Guide.

NAME

lex - generate programs for simple lexical tasks

SYNOPSIS

lex [**-rctvn**] [file] ...

DESCRIPTION

The *lex* command generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array.

Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx-z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *****, **+**, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class. The character **.** is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported.

The notation *r*{*d*,*e*} in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than /, but lower than *, ?, +, and concatenation. Thus **[a-zA-Z]+** matches a string of letters. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character **$** at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \.

Three subroutines defined as macros are expected: **input**( ) to read a character; **unput**(*c*) to replace a character read; and **output**(*c*) to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex**( ), and the library contains a **main**( ) which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore**( ) accumulates additional characters into the same *yytext*; and the function **yyless**(*p*) pushes back the portion of the string matched beginning

at *p*, which should be between *yytext* and *yytext+yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes **%%** it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a **%%**, as in YACC. Lines preceding **%%** which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with {}. Note that curly brackets do not imply parentheses; only string substitution is done.

EXAMPLE

```
D       [0-9]
%%
if      printf("IF statement\n");
[a-z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"     skipcommnts();
%%
 skipcommnts()
{
        for (;;)
        {
                while (input() != '*')
                        ;
                if (input() != '/')
                        unput(yytext[yyleng-1]);
                else
                        return;
        }
}
```

The external names generated by *lex* all begin with the prefix **yy** or **YY**.

The flags must appear before any files. The flag **-r** indicates RATFOR actions, **-c** indicates C actions and is the default, **-t** causes the **lex.yy.c** program to be written instead to standard output, **-v** provides a one-line summary of statistics, **-n** will not print out the **-v** summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n*	number of positions is *n* (default 2500)
%n *n*	number of states is *n* (500)
%e *n*	number of parse tree nodes is *n* (1000)
%a *n*	number of transitions is *n* (2000)
%k *n*	number of packed character classes is *n* (1000)
%o *n*	size of output array is *n* (3000)

The use of one or more of the above automatically implies the **-v** option, unless the **-n** option is used.

SEE ALSO

yacc(1).
UNIX System V Release 3.2 Programmer's Guide.

BUGS

The **-r** option is not yet fully operational.

NAME

line - read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1), read(2).

NAME

link, unlink - link and unlink files and directories

SYNOPSIS

/etc/link file1 file2

/etc/unlink file

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the *rm*(1) and *rmdir*(1) commands be used instead of the *unlink* command.

The only difference between *ln*(1) and *link/unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the *link*(2) and *unlink*(2) system calls.

SEE ALSO

rm(1), link(2), unlink(2).

WARNINGS

These commands can be run only by the super-user.

NAME

lint - a C program checker

SYNOPSIS

lint [option] ... file ...

DESCRIPTION

The *lint* command attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected 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 that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **-c** or the **-o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

lint will take all the **.c**, **.ln**, and **llib-lx.ln** (specified by **-lx**) files and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **-p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **-c** option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the **-c** option is used, the **.ln** and the **llib-lx.ln** files are ignored.

Any number of *lint* options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

-a Suppress complaints about assignments of long values to variables that are not long.

-b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints).

-h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

-u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program).

-v Suppress complaints about unused arguments in functions.

-x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

-lx Include additional lint library **llib-lx.ln**. For example, you can include a lint version of the math library **llib-lm.ln** by inserting **-lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

-n Do not check compatibility against either the standard or the portable lint library.

-p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

-c Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for inter-function compatibility.

-o lib Cause *lint* to create a lint library with the name **llib-l***lib***.ln**. The **-c** option nullifies any use of the **-o** option. The lint library produced is the input that is given to *lint*'s second pass. The **-o** option simply causes this file to be saved in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **-x** option is suggested. The **-v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see below).

The **-D**, **-U**, and **-I** options of *cpp*(1) and the **-g** and **-O** options of *cc*(1) are also recognized as separate arguments. The **-g** and **-O** options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc*(1) command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code. [This comment is typically placed just after calls to functions like *exit*(2)].

/*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.

/*ARGSUSED*/
: turns on the -v option for the next function.

/*LINTLIBRARY*/
: at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the -v and -x options.

lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the **-c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **-c** and the **-o** options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the **-c** option. Each of these invocations produces a **.ln** file which corresponds to the .c file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the **-c** option), listing all the **.ln** files with the needed **-lx** options. This will print all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

FILES

LLIBDIR	the directory where the lint libraries specified by the **-lx** option must exist, usually /usr/lib
LLIBDIR/lint[12]	first and second passes
LLIBDIR/llib-lc.ln	declarations for C Library functions (binary format; source is in *LLIBDIR*/llib-lc)

LLIBDIR/llib-port.ln	declarations for portable functions (binary format; source is in *LLIBDIR*/llib-port)
LLIBDIR/llib-lm.ln	declarations for Math Library functions (binary format; source is in *LLIBDIR*/llib-lm)
TMPDIR/*lint*	temporaries
TMPDIR	usually /usr/tmp but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*() in *tmpnam*(3S)].

SEE ALSO

cc(1), cpp(1), make(1).

BUGS

exit(2), *setjmp*(3C), and other functions that do not return are not understood; this causes various lies.

NAME

list - produce C source listing from a common object file

SYNOPSIS

list [**-V**] [**-h**] [**-F** function] source-file ... [object-file]

DESCRIPTION

The *list* command produces a C source listing with line number information attached. If multiple C source files were used to create the object file, *list* accepts multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified, the default object file, **a.out**, is used.

Line numbers are printed for each line marked as breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins anew for each function. Line number 1 is always the line containing the left curly brace ({) that begins the function body. Line numbers are also supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

The following options are interpreted by *list* and may be given in any order:

-V	Print, on standard error, the version number of the *list* command executing.
-h	Suppress heading output.
-F*function*	List only the named function. The **-F** option may be specified multiple times on the command line.

SEE ALSO

as(1), cc(1), ld(1).

CAVEATS

Object files given to *list* must have been compiled with the **-g** option of *cc*(1).

Since *list* does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

DIAGNOSTICS

list produces the error message "list: name: cannot open" if *name* cannot be read. If the source file names do not end in **.c** , the message is "list: name: invalid C source name". An invalid object file causes the message "list: name: bad magic" to be produced. If some or all of the symbolic debugging information is missing, one of the following messages is printed: "list: name: symbols have been stripped, cannot proceed", "list: name: cannot read line numbers", and "list: name: not in symbol table". The following messages are produced when *list* has become confused by **#ifdef**'s in the source file: "list: name: cannot find function in symbol table", "list: name: out of sync: too many }", and "list: name: unexpected end-of-file". The error message "list: name: missing or inappropriate line numbers" means that either symbol debugging information is missing, or *list* has been confused by C preprocessor statements.

NAME

locate - identify a CTIX system command using keywords

SYNOPSIS

[help] locate
[help] locate [keyword1 [keyword2] ...]

DESCRIPTION

The *locate* command is part of the CTIX system Help Facility, and provides on-line assistance with identifying CTIX system commands.

Without arguments, the initial *locate* screen is displayed from which the user may enter keywords functionally related to the action of the desired CTIX system commands they wish to have identified. A user may enter keywords and receive a list of CTIX system commands whose functional attributes match those in the keyword list, or may exit to the shell by typing q (for "quit"). For example, if you wish to print the contents of a file, enter the keywords "print" and "file". The *locate* command would then print the names of all commands related to these keywords.

Keywords may also be entered directly from the shell, as shown above. In this case, the initial screen is not displayed, and the resulting command list is printed.

More detailed information on a command in the list produced by *locate* can be obtained by accessing the *usage* module of the CTIX system Help Facility. Access is made by entering the appropriate menu choice after the command list is displayed.

From any screen in the Help Facility, a user may execute a command via the shell [*sh*(1)] by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable SCROLL must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file [see *profile* (4)]:

```
export SCROLL ; SCROLL=no
```

If you later decide that scrolling is desired, SCROLL must be set to yes.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), sh(1), starter(1), usage(1), term(5).

WARNINGS

If the shell variable **TERM** [see *sh*(1)] is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

login - sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It can be invoked as a command or by the system when a connection is first established. Also, *login* is invoked by the system when a previous user terminates the initial shell by pressing *cntrl-d* to indicate an "end-of-file." (See *How to Get Started* at the beginning of this volume for instructions on logging in initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by using the following command from the initial shell:

exec login

The *login* command prompts for a user name (if not supplied as an argument) and, if appropriate, a password. Echoing is disabled (where possible) as the user enters a password, so the password does not appear on the written record of the session.

At some installations, an option can be invoked that requires a second "dialup" password for dialup connections.

If the login attempt is incorrect, a message informs the user that the login is incorrect, and a new login prompt appears. After five incorrect login attempts, all five are logged in **/usr/adm/loginlog** (if it exists) and the line is dropped.

If the login is not successful within a certain period of time (for example, one minute), the login attempt can be silently disconnected.

After a successful login, the user ID, the group ID, the working directory, and the command interpreter [usually *sh*(1)] are initialized. If the shell **/bin/sh** is running, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day (if any) is printed, and the file **.profile** in the working directory is executed, if it exists. If the shell **/bin/csh** is running, the **.login** and **.cshrc** files in the working directory are executed, if they exist. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is - followed by the last component of the interpreter's path name (for example, **-sh**). If this field in the password file is empty, the default command interpreter, **/bin/sh** is used. If this field is an asterisk (*), the named directory becomes the **root** directory, the starting point for path searches

for path names beginning with a slash (/). At that point, *login* is re-executed at the new level, which must have its own **root** structure, including **/bin/login** and **/etc/passwd**.

The basic *environment* is initialized as follows:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment can be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests a login name. The environment arguments can take the form *xxx* or *xxx=yyy*. Arguments without an equal sign (=) are named as follows as they are placed in the environment:

L*n*=xxx

where *n* is a number starting at 0, and incremented for each new variable. Arguments that contain an equal sign are placed into the environment as specified. If a variable already appears in the environment, the new value replaces the older value. There are exceptions: the variables **PATH, SHELL, HOME, LOGNAME, CDPATH, IFS** cannot be changed. This prevents users logging into restricted shell environments from spawning secondary shells that are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/usr/adm/loginlog	record of failed login attempts
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile (**/bin/sh** only)
/etc/cprofile	system profile (**/usr/local/bin/csh** only)
.profile	user's login profile (**/bin/sh** only)

.cshrc	user startup file (**/usr/local/bin/csh** only)
.login	user login initialization file (**/usr/local/bin/csh** only)

SEE ALSO

csh(1), mail(1), newgrp(1M), passwd(1), sh(1), su(1M), loginlog(4), passwd(4), profile(4), environ(5).

DIAGNOSTICS

login incorrect
The user name or the password cannot be matched.

no shell, *cannot open password file*, or *no directory*
Consult your system administrator.

No utmp entry. You must exec "login" from the lowest level "sh"
You attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname - get login name

SYNOPSIS

logname

DESCRIPTION

logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3X), environ(5).

NAME

lorder - find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* [see *ar*(1)]. The standard output is a list of pairs of object file or archive member 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). Note that the link editor *ld*(1) is capable of multiple passes over an archive in the portable archive format [see *ar*(4)] and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar -cr library `lorder *.o | tsort`
```

FILES

TMPDIR/*symreftemporary files

TMPDIR/*symdef temporary files

TMPDIR is usually /tmp but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*( ) in *tmpnam*(3S)].

SEE ALSO

ar(1), ld(1), tsort(1), ar(4).

CAVEAT

lorder will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be **.o**.

NAME

lp, cancel - send/cancel requests to an LP line printer

SYNOPSIS

lp [**-c**] [**-d** dest] [**-m**] [**-n** number] [**-o** option] [**-s**] [**-t** title] [**-w**] files

cancel [ids] [printers]

DESCRIPTION

The *lp* command arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name - stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they are printed.

lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see the description of *cancel*, later in this page) or find the status [see *lpstat*(1)] of the request.

The following options to *lp* can appear in any order and can be intermixed with file names:

-c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* are not copied, but they are linked whenever possible. If the **-c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **-c** option, any changes made to the named *files* after the request is made but before it is printed are reflected in the printed output.

-d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, the request is printed only on that specific printer. If *dest* is a class of printers, the request is printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, and the like), requests for specific destinations may not be accepted [see *accept*(1M) and *lpstat*(1)]. By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems [see *lpstat*(1)].

-m Send mail [see *mail*(1)] after the files have been printed. By default, no mail is sent upon normal completion of the print request.

-n*number* Print *number* copies (default of 1) of the output.

-o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **-o** keyletter more than once. For more information about what is valid for *options*, see **Models** in *lpadmin*(1M).

-s Suppress messages from *lp*(1) such as *request id is ....*

-t*title* Print *title* on the banner page of the output.

-w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, mail is sent instead.

Cancel cancels line printer requests made by the *lp*(1) command. The command line arguments may be either request *id*s [as returned by *lp*(1)] or *printer* names [for a complete list, use *lpstat*(1)]. Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request currently printing on that printer. In either case, the cancellation of a request currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lpadmin(1M), lpsched(1M), lpstat(1), mail(1).
S/Series CTIX Administrator's Guide.
CTIX Administration Tools Manual.

NAME

lpadmin - configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin -p printer [options]

/usr/lib/lpadmin -x dest

/usr/lib/lpadmin -d[dest]

DESCRIPTION

lpadmin configures line printer (LP) spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination. *lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **-p**, **-d** or **-x** options must be present for every legal invocation of *lpadmin*.

-p*printer* names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.

-x*dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **-x**.

-d[*dest*] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **-d**.

The following *options* are only useful with **-p** and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

-c*class* inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.

-e*printer* copies an existing *printer's* interface program to be the new interface program for *P*.

-h indicates that the device associated with *P* is hardwired. This *option* is assumed when adding a new printer unless the **-l** *option* is supplied.

-i*interface* establishes a new interface program for *P*. *Interface* is the path name of the new program.

-l indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

-m*model* selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP Spooling Utilities (see *Models* below).

-r*class* removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.

-v*device* associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by *lp*. Note that the same *device* can be associated with more than one *printer*. If only the **-p** and **-v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the **-v** option and one of the **-e**, **-i** or **-m** options must be supplied. Only one of the **-e**, **-i** or **-m** options may be supplied. The **-h** and **-l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and _ (underscore).

Models.

Model printer interface programs are supplied with the LP Spooling Utilities. They are shell procedures which interface between *lpsched* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **-m**. Copies of model interface programs may also be modified and then associated with printers using *lpadmin* **-i**. The following describes the *models* which may be given on the *lp* command line using the **-o** keyletter:

dumb interface for a line printer without special functions and protocol. Form feeds are assumed. This is a good model to copy and modify for printers which do not have models.

1640 DIABLO 1640 terminal running at 1200 baud, using XON/XOFF protocol. Options:

-12 12-pitch (10-pitch is the default)

-f do not use the *450*(1) filter. The output has been pre-processed by either *450*(1) or the *nroff*(1) 450 driving table.

hp Hewlett-Packard 2631A line printer at 2400 baud. Options:

-c compressed print
-e expanded print

prx Printronix P300 or P600 printer using XON/XOFF protocol at 1200 baud.

EXAMPLES

1. Assuming there is an existing Hewlett-Packard 2631A line printer named *hp2*, it will use the **hp** model interface after the command:

```
/usr/lib/lpadmin -php2 -mhp
```

2. To obtain compressed print on *hp2*, use the command:

```
lp -dhp2 -o-c files
```

3. A DIABLO 1640 printer called *st1* can be added to the LP configuration with the command:

```
/usr/lib/lpadmin -pst1 -v/dev/tty002 -m1640
```

4. An *nroff*(1) document may be printed on *lp* in any of the following ways:

```
nroff -T450 files | lp -dst1 -of
nroff -T450-12 files | lp -dst1 -of
nroff -T37 files | col | lp -dst1
```

5. The following command prints the password file on *st1* in 12-pitch:

```
lp -dst1 -o12 /etc/passwd
```

NOTE: the **-12** option to the **1640** model should never be used in conjunction with *nroff*(1).

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1).
S/Series CTIX Administrator's Guide.
CTIX Administration Tools Manual.

NAME

lpr - line printer spooler

SYNOPSIS

lpr [option ...] [name ...]

DESCRIPTION

The *lpr* command causes the named files to be queued for printing on a line printer. If no names appear, the standard input is assumed; thus *lpr* may be used as a filter.

Note that *lpr* is a simple alternative to the *lp*(1) system; one system should not use both.

The *lpr* command uses a CTIX demon to manage spooling.

The following *options* can be given (each as a separate argument and in any order) before any file name arguments:

-c Makes a copy of the file to be sent before returning to the user.

-r Removes the file after sending it.

FILES

/etc/passwd	user's identification and accounting data
/usr/lib/lpd	line printer daemon
/usr/spool/lpd/*	spool area
/etc/init.d/lp	initialization for *lp* or *lpr* spooling system
/etc/rcopts/LPR	presence of this zero-length file is required to start *lpr* when the system is booted.

SEE ALSO

S/Series CTIX Administrator's Guide.

NAME

lpsched, lpshut, lpmove - start/stop the LP scheduler and move requests

SYNOPSIS

/usr/lib/lpsched

/usr/lib/lpshut

/usr/lib/lpmove requests dest

/usr/lib/lpmove dest1 dest2

DESCRIPTION

lpsched schedules requests taken by *lp*(1) for printing on line printers (LP's).

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp (1)* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status [see *accept*(1M)] for the new destination when moving requests.

FILES

/usr/spool/lp/*	spool area
/etc/init.d/lp	initialization for *lp* or *lpr* spooling system (calls *lpsched*).
/etc/rcopts/LP	presence of this zero-length file is required to start *lpsched* when the system is booted.

SEE ALSO

accept(1M), enable (1), lp(1), lpadmin(1M), lpstat(1).
S/Series CTIX Administrator's Guide.
CTIX Administration Tools Manual.

NAME

lpset - set parallel line printer options

SYNOPSIS

lpset [control options] [mode options]

DESCRIPTION

The *lpset* command sets the translation options for the parallel printer interface. The following control options can be used; the interpretation of these options by the interface is described under *lp*(7).

-i*n*	Set the indent to *n*.
-c*n*	Set the number of columns to *n*.
-l*n*	Set the number of lines-per-page to *n*.
-p*printer_id*	Print on the specified printer; *printer_id* can be 0 or 1. If *printer_id* is not specified, **/dev/lp0** is used.

The following mode option choices can also be selected:

bs | nobs	Backspace/No backspace
raw | canon	Raw output mode/Canonical mode
cap | allcase	Translate lowercase to capitals/Both upper- and lowercase
cr | nocr	Carriage return/No carriage return
ff | noff	Formfeed/No formfeed
nl | nonl	New-line/No new-line

With no options, *lpset* reports the current values of device **/dev/lp0.** Initially, the values are as follows: an indent of 4, 132 columns, 66 lines per page. If **-c** is set to 0, the control values and modes are be set to their default values.

EXAMPLE

The following command specifies an indent of 4, 80 columns across the page, and no automatic formfeeds for line printer **/dev/lp0.**

```
lpset -i4 -c80 -l0 -p0 noff
```

FILES

/dev/lp
/dev/plp
/dev/plp1

SEE ALSO

lp(7).

NAME

lpstat - print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

The *lpstat* command prints information about the current status of the LP spooling system.

If no *options* are given, *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
-u"user1, user2, user3"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat -o
```

prints the status of all output requests.

-a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

-c[*list*] Print class names and their members. *List* is a list of class names.

-d Print the system default destination for *lp*.

-o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

-p[*list*] Print the status of printers. *List* is a list of printer names.

-r Print the status of the LP request scheduler

-s Print a status summary, including the system default destination, a list of class names and their members, and a list of printers and their associated devices.

-t Print all status information.

-u[*list*] Print status of output requests for users. *List* is a list of login names.

-v[*list*] Print the names of printers and the path names of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls - list contents of directory

SYNOPSIS

ls [**-RadCxmlnogrtucpFbqisf**] [names]

DESCRIPTION

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

There are four listing formats:

Multicolumn format — This is the default when the standard output is a terminal. By default this format sorts names down the page; the **-x** option controls this. Choice of multicolumn format is controlled manually by the **-C** option.

Simple format (one entry/line) — This is the default when the standard output is not a terminal. Each line consists of a file name together with whatever additional information is requested by options.

Long format — See the description of the **-l** option.

Stream format. — See the description of the **-m** option.

The number of columns used in multicolumn and Stream format is taken from an environment variable **COLUMNS**. If this variable is not set, the *terminfo*(4) database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

The *ls* command has the following options:

-R Recursively list subdirectories encountered.

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-d If an argument is a directory, list only its name (not its contents); often used with **-l** to get the status of a directory.

-C If the output device is a terminal, simple (one entry per line) format. If the output device is not a terminal, multi-column output with entries sorted down the columns.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format; files are listed across the page, separated by commas.

-l List in long format, giving mode, number of links, owner, group, 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 rather than a size.

-n The same as **-l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

-o The same as **-l**, except that the group is not printed.

-g The same as **-l**, except that the owner is not printed.

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **-n** and **-c**.)

-u Use time of last access instead of last modification for sorting (with the **-t** option) or printing (with the **-l** option).

-c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**-t**) or printing (**-l**).

-p Put a slash (/) after each filename if that file is a directory.

-F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.

-b Force printing of non-printable characters to be in the octal **\ddd** notation.

-q Force printing of non-printable characters in file names as the character question mark (?).

-i For each file, print the i-number in the first column of the report.

-s Give size in blocks, including indirect blocks, for each entry.

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

The mode printed under the **-l** option consists of ten characters. The first character can be one of the following:

d The entry is a directory.
b The entry is a block special file.
c The entry is a character special file.
p The entry is a FIFO (named pipe) special file.
- The entry is an ordinary file.

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

ls -l (the long list) prints its output as shown below:

```
-rwxrwxrwx  1 smith   dev     10876  May 16 9:42 part2
```

This horizontal configuration provides a good deal of information. Reading from right to left, notice that:

- The current directory holds one file, named **part2.**
- The last time that file's contents were modified was 9:42 A.M. on May 16.
- The file is moderately sized, containing 10,876 characters, or bytes.
- The owner of the file, or the user, belongs to the group **dev**, and the user's login name is **smith.**
- The number, in this case **1**, indicates the number of links to the file **part2.**
- The row of dash and letters indicate that user, group, and others have permissions to read, write, and execute **part2.**

The permissions are indicated as follows:

r The file is readable.
w The file is writable.
x The file is executable.
- The indicated permission is *not* granted.
l Mandatory locking occurs during access (the set-group-ID bit is on and the group execution bit is off).
s The set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on.
S Undefined bit-state (the set-user-ID bit is on and the user execution bit is off).

t The 1000 (octal) bit, or sticky bit, is on [see *chmod*(1)], and execution is on.

T The 1000 bit is turned on, and execution is off (undefined bit-state).

The ability to assume the same ID as the user during execution is, for example, used during login when you begin as **root** but need to assume the identity of the user stated at login.

Mandatory record locking describes a file's ability to allow other files to lock its reading or writing permissions during access.

EXAMPLES

An example of a file's permissions is shown below:

```
-rwxr---r---
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions follows:

```
-rwsr-xr-x
```

The second example describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

Another example of a file's permissions follows:

```
-rw-rwl---
```

This example describes a file that is readable and writable only by the user and the group and can be locked during access.

The following use of the *ls* command displays the names of all files in the current directory, including those that begin with a dot (.), which are not normally included in the *ls* report:

```
ls -a
```

The following use of the *ls* command displays an informative report that includes all files, even non-printing ones (a); the i-number—the memory address of the i-node associated with the file—printed in the left column (i); the

size (in blocks) of the files, printed in the column right of the i-numbers (s); The report is displayed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

ls -alsn

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

FILES

/etc/passwd	user IDs for **ls -l** and **ls -o**
/etc/group	group IDs for **ls -l** and **ls -g**
/usr/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod(1), find(1).

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the section on mapping remote users in the *S/Series CTIX Administrator's Guide*.

BUGS

Unprintable characters in file names may confuse the columnar output options.