FORTUNE SYSTEMS

INTRODUCTION TO FOR:PRO

SYSTEM TOOLS

INTRODUCTION TO FOR:PRO

COMMAND-LEVEL USE OF FORTUNE'S OPERATING SYSTEM

FORTUNE SYSTEMS
300 Harbor Boulevard
Belmont, CA 94002

Printed in U.S.A.
1 2 3 4 5 6 7 8 9 0

Ordering Introduction to FOR:PRO

Order Number: 1002268-01 October 1983 Software Release 1.7
Please do not order products from the address shown below. Consult an authorized Fortune Systems dealer for copies of manuals and technical information.

About This Book

FOR:PRO, Fortune's Professional Operating System, is an enhanced version of Bell Laboratories' Version 7 UNIX operating system, modified to run on the Fortune system. The Introduction to FOR:PRO is designed to address the needs of two different groups of Fortune system users:

- Those who have read Understand Your Fortune System, used the Fortune menu system, and need to know FOR:PRO at the command level.

- Those with prior computer experience who want to use FOR:PRO at the command level rather than at the menu level.

The main goals of this book are to familiarize you with the commands on the FOR:PRO single-user operating system so you can use them to perform everyday tasks, and to provide a complete set of MAN pages corresponding to the commands on the FOR:PRO set of flexible disks.

The first two chapters are designed to bridge the gap between the menu system interface and the command level interface to FOR:PRO. The material contained in these chapters assumes only minimal familiarity with computer terminology and concepts. The last two chapters are designed to meet the needs of users experienced with other computer systems, but not necessarily UNIX-based systems.

This book assumes you have already read Understand Your Fortune System, so the concepts here are simply summarized.

ORGANIZATION OF THIS BOOK

This document has two parts:

- **Part 1** introduces FOR:PRO and describes how to use simple commands to perform common operations.

- **Part 2** is the entire collection of the manual pages, called "MAN pages." They provide reference documentation for all commands and files that are part of the single-user FOR:PRO system.

Part 1 is divided as follows:

- **Chapter 1** reviews UNIX terms and concepts introduced in Understand Your Fortune System. It then explains how to access FOR:PRO at the command level and how to set up your working environment.

- **Chapter 2** compares the menu system with equivalent UNIX commands, and describes how to use some of these commands.

- **Chapter 3** deals with advanced commands, such as those enabling you to set up system security, change the way commands work, and put commands together to perform routine tasks.

- **Chapter 4** explains the organization of system directories provided with FOR:PRO and summarizes the commands available on the FOR:PRO single-user operating system.

- **Appendix A** describes the changes to the print spooler at this release.

CONVENTIONS USED IN PART 1

When communicating with FOR:PRO at command level, you type entire commands without going through the menu system. Commands must be typed using proper command format, or syntax. This document uses certain conventions to help you understand exactly how a command is typed. These are outlined below.

- **Commands** to FOR:PRO are almost always lowercase letters, as is with most UNIX commands. Occasionally, some command options may be uppercase. Be careful to type commands using the case shown in the syntax descriptions.

- In examples, any **input** you must type appears in **boldface**, while output such as system messages does not. Also, items shown in **boldface** must be typed exactly as they appear in the text.

- **Words to be replaced** with your own text are underlined. Such items are also referred to as command-line parameters.

- **Brackets** [ ] indicate one or more options you may select from.

- **Hyphens** (**-**) usually signal an option on a command line, as in the command

  ls -l

  Always remember to type the hyphen.

- **Ellipses** (**...**) mean that the preceding option may be repeated more than once.

- Commands are sent to FOR:PRO by pressing the **RETURN** key, shown as Return throughout this text. In examples and in command syntax, it is represented as <RETURN>.

Contents

USING FOR:PRO COMMANDS 1

1 The Shell Environment

This chapter reviews concepts covered in Understand Your Fortune System and introduces concepts that relate particularly to using FOR:PRO at command level. A few simple commands are introduced to familiarize you with using the system at command level. Topics covered include:

- Related terms and concepts
- The three FOR:PRO shells and how to use them
- Directories, files, and file systems
- The ed text editor
- Customizing your login environment

BASIC UNIX TERMS AND CONCEPTS

To make the best use of the FOR:PRO operating system, you should understand basic terms such as **file system**, **utility**, **shell**, and other concepts briefly discussed in this section. Most of these terms are discussed fully in Parts 1 and 2 of Understand Your Fortune System.

FOR:PRO OPERATING SYSTEM

An operating system is a large set of instructions that manage the activities of a computer. The FOR:PRO operating system is an enhanced form of Version 7 UNIX, an operating system developed by Bell Laboratories. FOR:PRO consists of a **kernel** (the "heart" of the operating system) and a set of utilities. Together, they manage all input and output, schedule processes, manage devices, and handle other crucial system tasks while carrying out requests from the system's users. Users do not communicate directly with an operating system like FOR:PRO; instead, they send their requests through interface programs called **shells**.

SHELLS

A shell is a command interpreter that reads information you type at the terminal and passes it to the operating system. FOR:PRO provides three shells as interfaces to the operating system: the **Bourne shell** (on the single-user operating system set of disks), the **C shell** (available with Development Utilities only), and the **menu system.** You have probably already used the menu system to perform various tasks. This chapter demonstrates how to use FOR:PRO at command level by using either the Bourne or C shells to issue commands directly to the kernel without menus as a go-between.

FILES

As is the case with UNIX, FOR:PRO organizes information in a series of directories and files. FOR:PRO uses three types of files:

- **Ordinary files** - groups of characters combined to form a computer program, text to be processed by a program, text created through the use of an application such as Multiplan, and so forth.

- **Special files** - used by the operating system for handling peripherals such as disks and printers.

- **Directory files** - contain the names of files and other directories, providing a way to find information on the disk.

DIRECTORY CONCEPTS

You should be familiar with the following terms related to directories:

- **Home directory** - the directory an account accesses upon login. Since the system automatically creates a home directory for new accounts in the directory /u, your home directory will be

  /u/your_login_name

- **Working directory** - the directory you are currently using; also known as the **current directory.**

- **Parent directory** - the directory in which a given directory or file is listed. Every file or directory on the system has a parent except the **root** directory (/). The root directory, or root, is the main directory for the entire FOR:PRO file system. (See Figure 1-1.)

- **Child directory** - a directory listed in a given directory. All the directories immediately under root are "children" of the root directory.

Use of the **manager/root** Account

If you log into the system as **manager** or **root**, you are automatically put in the root directory (/). If you then access Fortune:Word or Multiplan, or any other application, any new documents or worksheets you create will be in the / directory instead of in your home directory. This can cause problems when you back up your system. All of these files will be owned by root, so you will have to log in as manager or root in order to back them up, copy them, or change them.

Before backing up your system on a daily basis, and especially before doing a cold boot, make sure you check the contents of the root directory to see if there are any user files resident there.

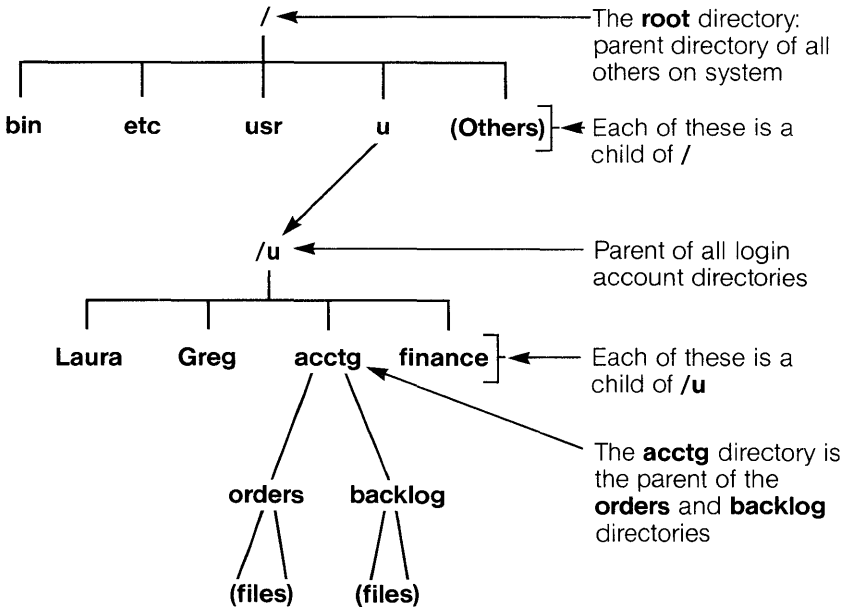
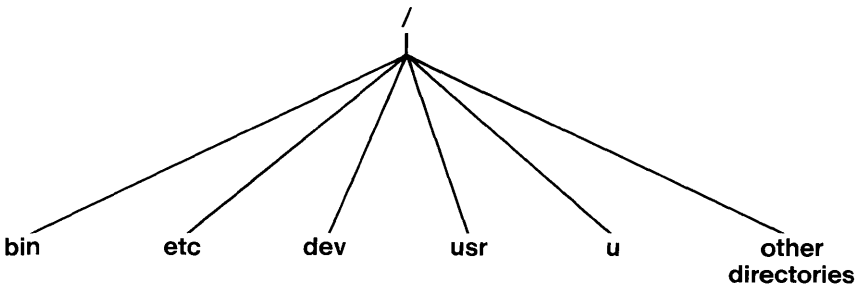


Figure 1-1. Parent and Child Directories

If so, you should consider moving them to an account in the **/u** directory. In any event, remember to back them up, or you will lose them if you do a cold boot.

FILE SYSTEMS AND TREE STRUCTURES

Files and directories on the FOR:PRO system are organized in a hierarchical manner called a **tree structure.** All the files and directories that exist under the root directory, which is the "top" of the upside down tree, are part of the FOR:PRO file system. All directories on the hard disk descend from the root. (See Figure 1-2.) Each flexible or hard disk contains its own complete file system. Normally when a flexible disk is used, it is mounted on **/f**, making **/f** the **root** directory for the flexible disk-based file system. Hierarchical tree structures are the operating system's way of looking at the data stored on the flexible or hard disk.



Directory		Contains
/	—	root
bin	—	system-level commands
etc	—	system-level commands and special files
dev	—	device files
usr	—	user-oriented commands and libraries
u	—	user account directories, user-created directories and files
other directories	—	other directories listed in root like /f, /m and Fortune software products

Figure 1-2. A Typical FOR:PRO File System

PATHNAME

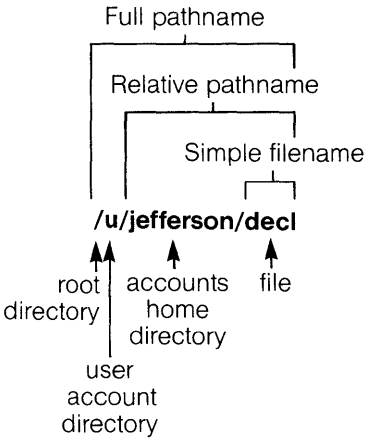
A **pathname** identifies where a file or directory exists within the tree-structured file system. There are two types of pathnames: full and relative. (See Figure 1-3.) A full pathname describes the exact location of a file within the hierarchical file system. For example, the full pathname of the file called **decl** created in the home directory of the user **jefferson** would be:

/u/jefferson/decl

A **pathname** differs from a **filename** in that a filename assumes the file is in the current directory. Use a file's pathname in a command if that file is not in your current directory.

A **relative pathname** includes any directories the system must go through to find a file, starting with your working directory. Using the previous example, if your current directory was **/u**, to access the file **decl**, you would specify

jefferson/decl



Full pathname — **(/u/jefferson/decl)** can be used while in any directory

Relative pathname — **(jefferson/decl)** used when current directory is **/u**

Simple filename — **(decl)** used only when file is in current directory, **(/u/jefferson)**

Figure 1-3. Filenames and Pathnames

From a current directory of **/u/jefferson** you could use ".." as a shorthand to indicate the parent directory **/u**. The ".." stands for "parent directory" and in this case, **/u**.

SHELLSCRIPT

A **shellscript** is a file containing one or more commands to be executed sequentially, as if typed directly from the terminal. More complex shellscripts use variables and simple program logic like "IF...THEN...ELSE". You can write your own shellscripts to perform routine actions, saving time and reducing the risk of mistakes. Table 1-1 shows a sample shellscript which mounts a flexible disk, lists its contents, then unmounts the disk.

Table 1-1. A Shellscript for Listing Files on a Flexible Disk

Command	Results
The shellscript **fd.look** contains:	
mount /dev/fd02 /f	Mounts a flexible disk.
ll /f	Displays a detailed list of files on the flexible disk.
umount /dev/fd02	Unmounts the flexible disk.

Execute these commands by running the shellscript fd.look and type

```
$sh fd.look
```

THE THREE SHELLS AND HOW TO ACCESS THEM

The three FOR:PRO shells, the **menu shell**, the **Bourne shell**, and the **C shell**, provide an easy way for users to communicate with the operating system.

The Menu Shell

The menu shell, also called the **global menu** system, consists of a set of menus descending from a single global menu you access when you log in.

The menus contain selections for running applications, accessing training materials, and performing basic system and file management tasks. For example, when you choose a selection from the system utilities or system management menus, this activates a FOR:PRO command or a shellscript which performs the selected activity.

The Bourne Shell

The **Bourne shell** is the standard user interface to UNIX. It was developed by Bell Laboratories and is offered on the FOR:PRO single-user operating system. Bourne shell prompt is a dollar sign ($) or a number sign (#), if you are logged in as root or manager. This book shows you how to issue FOR:PRO commands at the Bourne shell level. When you see a reference to the shell within this text, assume the Bourne shell is being discussed.

The C Shell

The **C shell**, also called the Berkeley shell, was developed at the University of California at Berkeley. It provides a user interface to the FOR:PRO operating system that is similar to the C programming language. The C shell prompt is the percent sign %. The C shell is not included on the FOR:PRO single-user operating system and is therefore not described here. If you want to use this shell, you should obtain the Development Utilities package, which contains the C shell software and appropriate documentation.

Accessing the Bourne Shell

Since all new accounts created with the newuser command automatically enter the menu system, you'll have to go to the global menu first before accessing the Bourne shell. Follow these steps to get from the menu to the Bourne shell:

1. From the global menu, type

   !

   then press Return. You can also type **!sh** to get to the Bourne shell.

2. The screen clears and a $ prompt (or # if you are logged in as root or manager appears in the upper left corner of your screen. The $ is the Bourne shell prompt.

Getting Back to the Menus

When you are finished using the shell, press the **Control** key and the **d** key simultaneously to get back to the global menu. This sequence is represented from here on in the text by **CTRL-D.** You can also type

exit

and press Return to get back to the menu system.

You can set up your account so that you access the Bourne shell automatically when you log in. It is recommended that you do if you intend to do most of your work at command level. See the section, Customizing Your Working Environment, for instructions on how to change your login shell. Then, you can access the menu shell by typing

menu

If your original login shell is the Bourne shell and you have not entered the global menu system since logging in, when you type exit or CTRL-D, you will be logged off the system.

To get out of the menu shell, you can always press the Cancel key. This logs you off completely, unless you were in another shell before accessing the menu system.

> NOTE
>
> It isn't a good practice to constantly move from one shell to another without occasionally logging off. For example, don't use **"menu"** to access the global menu, then use an application, and then use ! to go back to the Bourne shell several times without logging off. This will definitely slow down the system's response to your prompts.

Issuing Commands from the Shell

To get you started at the FOR:PRO command level, here are some essential commands that you'll soon be using on a regular basis. Remember that all commands must be typed in lowercase letters and must be followed by a carriage return. Here is a list of routine tasks and the commands you would use to perform them:

Routine Task	Command
Print or display (current) working directory	**pwd**
Find out what is in the directory	**ls, ll**
Change to another directory	**cd**
Find out who's on the system	**who**
Print (display) the current date and time	**date**
List the contents of an entire file on the screen	**cat**
List the contents of a file twenty-four lines at a time	**more, page**

Correcting Errors

You will probably want to know how to correct typing mistakes while entering commands to the shell. The default erase character is the Backspace key. It erases one character at a time. Simply press Backspace until you reach the place on the command line where you made the error and retype. Typing CTRL-W erases an entire word at a time. The default line delete, or **kill** character, is CTRL-X. After typing CTRL-X, press Return, and the line will be erased. If you want to change your default erase and kill characters, see Chapter 3.

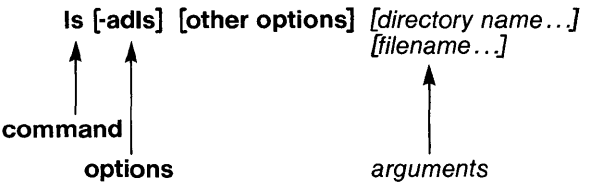
> NOTE
>
> You can always press the Cancel key to abort a command.

FOR:PRO Command Syntax

All FOR:PRO commands follow UNIX syntax rules. A command's proper format or syntax must be used to make it work properly.

Each command consists of a word or abbreviation representing the command itself, usually followed by one or more words called **arguments.** Arguments further define the action a command is to take. Finally, each FOR:PRO command must be terminated by pressing the Return key.

A typical FOR:PRO command, **ls,** can be broken down as follows:



An option is a single character that tells the command to perform a certain additional function. One or more of the options indicated in brackets can be specified in a single ls command line. The ellipses (...) mean that more than one file or directory name can be used with a single **ls** command.

In the command formats shown in this document, the text you must type exactly as shown appears in **bold.** Optional items are enclosed in square brackets. This includes arguments, which are typed as shown, and parameters, for which you must supply real values, like an actual filename or directory name. Parameters are underlined.

Print Working Directory (**pwd**)

One of the first things you are likely to ask once you've entered the shell is, "Where am I?" The **pwd** command prints the pathname of your current directory.

Follow these steps to find the location of your current working directory:

1. You should see the $ prompt on your screen. Type **pwd** and press the Return key.

2. Then system responds with a message like

   /u/yourname or /

   where yourname is the name of your account. If you access the shell from the global menu immediately after login, you should be in your home directory.

However, if you log in as manager, you will be in the **root** directory when you return to the shell. Use **pwd** to find out where you are.

To get to your home directory at any time, type

cd /u/yourname

where yourname is the name of your login account.

> NOTE
>
> Accounts should be created with the **newuser** command, which creates a directory for each user login account in the **/u** directory. When you log into the system using your account (login) name, you will be put into your home directory: /u/yourname.

List Current Directory (**ls**)

To see the contents of a directory except for files begining with ".", use the **ls** command. If you type

ls

the names of all directories and files in the current directory will be displayed on the screen. You can also request a list of the contents of a different directory by typing that directory's pathname with the **ls** command, as in this example:

ls /usr/bin

The **ls** command also has a large number of options for listing information about directories and files in addition to their names. To use the **ls** command with an option, the format would be

ls -option

Options are always preceded by a hyphen (-). Some commonly used options of **ls** are:

-l Lists the directory's contents in long format, giving status information about all files and subdirectories. This information includes the permission settings, size, and ownership of each item listed in the directory. (FOR:PRO provides a shortcut command, **ll**, which does the same thing.)

-a Lists all entries in the directory, including all the entries beginning with a period as in the . and .. entries. If you are not logged in as root or manager, you can only see the . entries if you use the **-a** option of **ls**. The . entry represents the directory listed. The .. entry represents the parent of the directory listed.

> NOTE
>
> If you are logged in as root or manager, simply typing **ls** will list the files whose names begin with "." (except for . and ..) as well as all others in the directory.

Listing Your Current Directory: To look at the entire contents of your current directory, type **ls -a**. The names of any directories and files you have created thus far including files beginning with . (like **.profile**), are displayed. (If this is your first time on the Fortune system, you may see only the dot (.) files, since your current directory may be empty.)

Listing Another Directory: You can also look at directories other than your current directory without changing your location. To look at a long listing of the / directory, for example, type

ls -l /

Notice the syntax you used for this command:

ls -option directory.name

The resulting display contains both files and directories and should resemble the following:

```
drwxrwxr-x  2 root    944 Jul 6 11:22  bin
drwxr-xr-x  1 root   1104 May 3 19:27  dev
drwxrwxr-x  3 root   1264 Jul 6 08:43  etc
drwxrwxr-x 10 bin     176 Jul 6 09:20  u
```

The directories above are four system directories that reside in /. They are discussed in more detail in Chapter 4. The following is a brief description of the status information:

d	Indicates the file is a directory. A hyphen (-) indicates an ordinary file.
rwxrwxr-x	Permissions assigned to the file or directory, which govern the way a file or directory can be used. The letter "r" means read, "w" means write/update, and "x" means execute. You'll learn more about this later.
2	Number of links to the file; usually 1. For a directory, it indicates the number of subdirectories in the directory.
root	The owner of the file.
944	Number of bytes (characters) the file or directory takes up.
Jul 6 11:22	Date the entry was last updated.
bin	The name of the file or directory.

Change Working Directory (**cd**)

The **cd** command moves you from one working directory to another. To use the command, type

cd directory.name

The way you specify the directory name depends on the location of the directory to which you want to move. For example, suppose you have a /u directory that contains the entries shown in Figure 1-5. To access the file **june.1**, you would use **cd** to get to the directory **june**, in which **june.1** resides. If your current directory is **/etc**, you'd have to give the full pathname to get to **june**.

```
cd /u/greg/memos/june
```

If you are already in the directory called **greg**, you'd use this relative pathname

```
cd memos/june
```

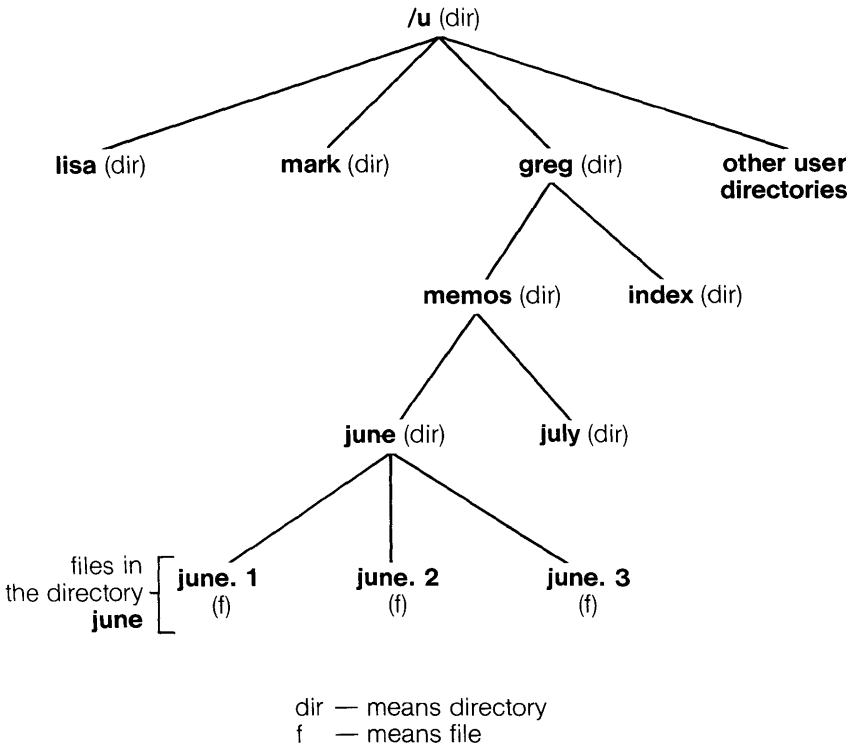


Figure 1-5. The User Account Directory (**/u**) and Typical Subdirectories

If the current directory is **memos**, type

cd june

Never begin a relative pathname with a /. The system will start its search with the / directory, rather than with your working directory.

If the directory you want is the parent of your working directory, type

cd ..

Referring to Figure 1-5 if your current directory is june, typing **cd ..** gets you to the directory **memos**. When you did an **ls -a**, you saw special characters (..) that stand for the parent directory. If you were in the **greg** directory and typed **cd ..**, you would go to **/u**, the parent directory of **greg**.

Finally, if the directory you want isn't located at a level directly above or below your current directory, you'll have to type **cd** and the full pathname of that directory. For example, suppose you are in the **memos** directory and want to access a directory called **lib** that you know isn't in a direct line above or below the current directory. You'd have to type that directory's full pathname: **/usr/lib**.

Using **cd**: The examples below show you how to move from your home directory to other directories on the system.

1. Type

   cd ..

   and your new location becomes the parent of your current directory.

2. Type **pwd**. You should get the response

   /u

3. Go to the **/etc** directory. Type its full pathname

   cd /etc

Display the Date and Time (**date**)

Date is a simple command that tells you the current system date and time. To use this command, type

date

Remember, if you don't update the date and time screen when you start up the system, the response you see may not agree with the actual date and time. To correct this, use the **/etc/setdt** command to display the same date and time screen shown when you turn on the Fortune system. Make your changes on this screen. (If you are logged in as root or manager, you can leave off the "/etc/" prefix.)

Who is Using the System? (**who**)

The **who** command tells you who is using your system. On a multiuser system this is very handy. On a single-user system, you might use it to find out what user ID you used when you logged into the system. To use **who**, type

who

The **who** display tells you which account is logged in and which terminal (**tty**) that person is using.

Concatenate and Print a File (**cat**)

The **cat** command, short for concatenate, which means "join," is a simple command that performs three functions:

1. **cat** displays an already existing text (ASCII) file on the screen. To use **cat** for this purpose, you'd type

   cat old.filename

2. **cat** combines or concatenates two files, so the second file specified is appended to the first. To concatenate **file.1** and **file.2**, creating a new file called **file.3**, you'd type

   cat file.1 file.2 > file.3

Then **file.3** is created from the results of concatenating **file.1** and **file.2**, leaving **file.1** and **file.2** in their original form. The > symbol tells the system to put the results of the **cat** process into **file.3**. Should you use this form where **file.3** already exists, **file.3** is overwritten by the results of the concatenation of **file.1** and **file.2**.

3. **cat** allows you to create a new file by simply pausing while you enter text. In this case you'd type

   cat > new.file

   to create a brand new file called new.file. You can also type

   cat >> old.file

   to append information you type at the terminal to an existing file, called old.file.

   In either case, the system waits while you type information you want sent to new.file or old.file. Just type in the lines you want. Signal the end of the file by pressing CTRL-D. Make sure new.file is the name of a new file and not an existing one; otherwise, it will be overwritten.

Using **cat**: To use **cat** for the first time, use the cd command to get to **/etc** and type

cat group

The system displays the contents of the **file group** on the screen. The file **/etc/group** is a system file containing the names of all user groups on the system.

> NOTE
>
> Do not use **cat** to display binary files such as system programs and executable user programs on the screen. If you do this by mistake, you may get a strange screen display. Use the **-v** option of **cat** (**cat -v**) to look at binary files.

Display a File a Screen at a Time (**more**)

The **more** command is similar to the display function of cat, because it displays an already existing text file on the screen. However, the **more** command only displays 24 lines of text at a time. Use **more** to display long files, because cat rapidly scrolls a long file over several screens without stopping until the end of the file.

To use **more**, type

more filename

The **more** command pauses at the bottom of the first screen of text until you press the Spacebar to display the next screen of text, or press Return to display the next line.

The following example demonstrates the use of **more**. A **pipe** (¦) will also be used. On many keyboards, the ¦ key is one of the three gray keys located to the left of the standard keys. Pipes allow you to use the output of one command or process as input to another.

1. Type the following:

   ls -l /etc ¦ more

   You've "piped" the results of an **ls -l** operation to the **more** command. If /etc is your current directory, you can omit **/etc** from the **ls** command line.

2. The system displays the first screen of the long directory listing, indicating that there is more to come.

3. Press the Spacebar to view the next 24 lines of output. If the $ prompt isn't at the bottom of the directory list, keep pressing the Spacebar until you see the $ prompt, signifying the end of the file. To get out of a display before it is finished, type **q** and press Return, or press the Cancel key.

INTRODUCTION TO TEXT EDITING AND THE ed COMMAND

Text editors provide tools for creating and editing text files. If you need to create reports and memos on the Fortune system, you should use the Fortune:Word application. However, if you need to create source programs, perform minor file updates, and create data files, you should use a text editor.

Types of Editors

There are two basic types of text editors available on the FOR:PRO system: line editors and screen editors. Line editors enable you to create text, then edit one line at a time. The line editor, **ed**, is the text editor available on the single-user operating system set of disks. It's easy to learn and adequate for creating simple ASCII text files. To do extensive text formatting on a file created with **ed**, use the commands associated with the text formatting program **nroff**, available with the Development Utilities software.

Screen editors enable you to edit text a screenful at a time, taking advantage of the characteristics of the video screen. If you have the Development Utilities set on your system, you can use **vi**, the screen editor provided with that software package.

The next couple of pages show you how to use **ed**. A brief example and table of ed commands are also provided.

Accessing **ed**

To use **ed**, you must be at the command level. Once there, use ed like any other FOR:PRO command. Use the syntax

ed filename

regardless of whether you are creating a new file or editing an existing one.

> NOTE
>
> The cursor control keys (grey keys with arrows) do not work in **ed**.

Rules for Creating Files

There are a few rules governing the creation of files. The first rule is that filenames can be a maximum of 14 characters, and should not begin with the following special characters:

, $ > < ? * & ; .

The second important rule is that FOR:PRO, like all UNIX systems, distinguishes between uppercase and lowercase letters, so type your filenames carefully.

Creating a New File with **ed**

There are two modes of operation in **ed:** **append** mode for creating text, and edit mode for issuing commands to modify text. To create a new file, type **ed** and the name of the file. Follow the file-naming rules discussed above. The system responds

```
?filename
```

indicating that this is a new file. The ? prompt indicates that you are in edit mode.

To enter append mode, type **a** and press the Return key. In append mode, you type the text you want in the file.

If you make a mistake while typing, you can correct the mistake provided it is on the current line. Use the Backspace key to erase the errors and then retype the text as you want it.

To leave append mode, start a new line with a period . and follow it by a Return.

At this point you can either edit the file or save it for modification at a later time. If you don't plan to edit the file right after creation, save its contents by issuing a **w** command. This command writes the current version of the file onto the hard disk. Type **q** to leave **ed** and return to the shell.

A Sample **ed** Session

Here's how to create a sample file using **ed.**

1. Type the following:

   ed softdrink

2. The system responds

   ?softdrink

3. Get into append mode by typing **a** and pressing the Return key. You won't get any more system responses until you leave append mode.

4. Type a few lines of text, such as these:

```
In New York, they call it soda.
Then it migrated to California
Where they also called it soda.
In Ohio and Indiana, they call it pop.
In Georgia, they call it coke,
even if it isn't cola.
Or so I'm told. But in Boston, they call it tonic
(that's tawnic!).
```

5. The single period on the last line of the file represents the end of text. Now type **1,$p** to display the whole file. (**1** refers to the first line, **$** to the last line of the file.)

6. Type **w** to write the text you just typed onto the hard disk. The system responds with a number, representing the number of characters (bytes) in the file.

7. Type **q** to leave **ed** and return to the shell. You should see the **$** prompt on your screen.

Editing Text Using ed

To modify existing files, use some of ed's single character commands to change text on a line-by-line basis. The **ed** commands you'll use most often are described in the table on the next page.

CUSTOMIZING YOUR WORKING ENVIRONMENT

If you want to access the shell directly every time you log in, you have to change the entry in the **/etc/passwd** file that describes your account. The **passwd** file contains important information about all the user accounts on the system. It tells the name of the account, the password for that account, the group to which the user

Table 1-2. Table of ed Commands

Command	Description
a	Appends text after the current line. The syntax of the **a** command is (current line) **a** <RETURN> (New text--as many lines as needed) . <RETURN>
d	Deletes the indicated line or lines. The syntax of **d** is [line number]**d** or [line number],[line number]**d**
i	Inserts text above the current line. You can insert as much text as necessary, indicating the end of text by typing a period on the line below the last line of text, followed by a Return. The syntax of **i** is **i** <RETURN> (New text--as many lines as needed) . <RETURN>
p	Prints on the screen the line or range of lines specified. The syntax of **p** is [line number]**p** [line number],[line number]**p** Use the **$** symbol to specify the last line of the file. You can also print the contents of a line by typing that line number, for example, 3 Return.
q	Quits **ed** and returns to the shell.

s Substitutes (replaces) a new string of text for an old string in the given line, in a range of lines, or globally, throughout the entire file.

Use this format to change the current line of text. Use the g option when you want to make the indicated change to every occurrence of oldtext on that line.

s/oldtext/newtext/[**g**]

Use this format to change text over a range of lines indicated by x,y.

x,y **s**/oldtext/newtext/

This format adds the newtext string to the beginning of the current line.

s/^/newtext/

To add something to the end of the line, use **$** instead of ^.

To change a string globally (throughout an entire file), use this format of the substitute (**s**) command:

1,$**s**/oldtext/newtext/**g**

u Restores the results of the last editing command issued at the current line. This only applies to editing commands that may have modified the current line. The contents of the line are restored to their state before the last command was issued. This command will **not** undo global changes -- it will only undo the change made to the last line of the file.

w Writes the contents of the file to the hard disk.

/text/ Searches for the characters between the slashes. (If the text contains special characters like /,\,[,;, or *, they must be preceded by a backslash \.)

	Return prints the current line.
.	In edit mode, typing a period and pressing Return prints the current line.

File Position Commands

$	Moves you to the last line of a file, printing its contents.
+[nn]	Moves you from the current line to the next line; with nn, moves you forward nn lines.
-[nn]	Moves you from the current line to the preceding line; with nn, moves you backwards nn lines (towards the top of the file).
nn	Moves you to the line numbered nn, and prints that line.

---

belongs, and the shell into which the user is automatically placed upon logging in. For most accounts, the listing will end with

/bin/menu

meaning that the account enters the menu system upon login. To access a shell other than the menu system, you must change **menu** to read sh or csh.

Changing the **passwd** File

To change the **/etc/passwd** file from the FOR:PRO level, you must log in as manager or root. When you log in as manager or root, your working directory is automatically set to /. (Get to the shell from the menu by typing **!** or **!sh.**) Then use cd to get to the **/etc** directory. It is recommended that a backup copy of the passwd file be made. To do this, simply use the **cp** command, as in

cp passwd passwd.backup

There is now a backup copy that you can reinstate if needed. Should you accidentally delete or mangle the passwd file, type

cp /etc/passwd.backup /etc/passwd

to restore the file from your backup copy.

Get into the passwd file by typing ed passwd. Find the line in the passwd file that pertains to your account. Use the /string/ command to find it. For example,

/laura/

will find the entry for the user **laura** and print out that line. Note that the last entry on the line describing your account reads **/bin/menu.** Now change the string **/bin/menu** to read

/bin/sh

Type this command line to change the string

s/menu/sh/p

The changed line is then displayed. Save your changes using the **w** and **q** commands. See the previous section on the editor Editing Text Using **ed** if you need help.

To see the effect of this change, you have to log in again. You can log off and log back in, or you can just type **login,** and you will be prompted for your account name and password. You should directly access the Bourne shell instead of the global menu. Once you've done this, you might want to set up your environment by modifying the **.profile** file.

Setting up **.profile**

The **.profile** file is listed in the / directory, and is merely a prototype file. It is meant for the **manager** or **root** account and does nothing for you as an ordinary user until you customize it. You must first make a copy of **.profile** in your home directory and then make the necessary modifications to the copy. These steps are described in the following paragraphs.

> NOTE
>
> Once you put a **.profile** file in your home directory, it will be executed upon login, even if your login shell is /bin/menu.

What's in **.profile**

Before you modify it, **.profile** looks like this:

```
PATH=':/bin:/usr/bin:/usr/ucb:/etc'; export PATH
TERM=FT; export TERM
HOME=/; export HOME
```

The first line of the file describes your command search PATH, which tells the system where to look for the commands you type. By default, the system first looks in the current directory, then the **/bin** directory, then in **/usr/bin**, and so on, as shown above. Since the shell environment gives you the ability to define your own commands, you may want to change the PATH so the system looks in your directory first, then searches the others.

The TERM line indicates the type of terminal you are using, by default FT, which stands for Fortune terminal.

The HOME line indicates the directory that FOR:PRO considers your home directory. The default for root and manager is /, which is why the PATH and HOME fields have the values they do. You should change this to be /u/yourname, where yourname is the name of your account. That way, whenever you type **cd** without specifying a name, you access your home directory.

Customizing **.profile**

First, copy **.profile** to your home directory, as in this example:

```
cp /.profile /u/yourname/.profile
```

Now you have your own copy to work with. If you type the command **ls -la**, you will notice that the permission rights for the **.profile** file are read only

```
r--r--r--
```

You have to change this so you have write access to the file. Use the **chmod** (change mode) command to give yourself write access. The **chmod** command is complex, as is the entire concept of permissions. **chmod** is described in Chapter 3. Enter this command line, so you can edit the file

```
chmod u+w .profile
```

The "u" stands for **user** and the "w" stands for **write.** Issue the command **ls -la**, and you see a write permission to the file. Use **ed** to change the first line. This is the PATH line that indicates the search path for your account. Change the string **:/bin** to

:/u/yourname**:/bin**

where yourname is the name of your account.

Use the following command string to change the PATH line, using a comma as the delimiter for the **s** command:

s,:/bin,:/u/yourname**:/bin,p**

This changes your PATH line and prints it out with the modifications you just made.

Change the HOME line so that it has your account name instead of the root directory. Again, use the comma as a delimiter in the substitution.

3s,/,/u/yourname**,p**

The "3" specifies that the change will take place on the third line of the file. You can omit it if "3" is already the current line.

Type **w** Return and **q** Return to get out of **ed** and to make these changes permanent. Having done this, you can go to your home directory by simply typing **cd.**

To put these changes into effect, you must log in again. First log out, using CTRL-D, then log in as you normally do.

When you log into the system from now on, your **.profile** file will automatically be executed.

MORE CUSTOMIZATIONS

You can customize your login environment by adding items to your **.profile** file. Here is a list of some simple things you can do:

- Display a message or "reminder" file
- Display the system time and date
- Change the default shell prompt
- List the contents of your directory
- Find out who is on the system

The first item in the list involves using either the echo command, followed by some text to be displayed, or using the cat command to list a file of reminders you've previously created. For example, if you have a file called **reminder** in your directory, you can list its contents by including the command

cat reminder

in your **.profile** file.

Similarly, the current time and date can be displayed by putting the **date** command in the **.profile** file.

Here is a sample **.profile** file that uses all the custom features listed above.

```
PATH=':/u/laura:/bin:/usr/bin:/usr/ucb:/etc'; export PATH
TERM=FT; export TERM
HOME=/u/laura; export HOME
PS1=LJD$
date
echo This system is running 1.7
cat reminder
ls -las
who
```

Setting Shell Variables

Changing the default Bourne shell prompt ($) involves setting the special **PS1** variable. In the example shown above, the default prompt is set to the user's initials, followed by a dollar sign, in this case, LJD$.

Like PS1, all shell variables are set by equating them to some value. The general format is

variable=value

with no space on either side of the equal (=) sign. For example,

PS1=PROMPT$

If you want a prompt that contains embedded spaces, surround the entire prompt string with quotes, as in

PS1="HI THERE$"

Once set, a shell variable is referenced by preceding it with the dollar sign. For example, to display the setting of the HOME variable, type: **echo $HOME.** With a **.profile** setup like the one shown in the example, typing **cd $HOME** should get you to the directory you defined as "home". (You can also simply type **cd** to get home.)

Displaying Variable Settings

To find out what shell variables are currently set for your environment, type

set

A complete list of variables and their settings is displayed.

/etc/profile in FOR:PRO

The file **/etc/profile** defines the default language for the system (LANGUAGE=EN), the **lpr** defaults for users of the menu system (LPRFLAGS), and the XON/XOFF mode for Fortune terminals. In XON/XOFF mode, when output to the screen is suspended using CTRL-S, it can only be resumed by typing CTRL-Q.

Currently, only the menu shell invokes **/etc/profile.** The Bourne shell (**/bin/sh**) does not read **/etc/profile.** If your login shell is **/bin/sh,** incorporate the settings in **/etc/profile** into your environment by copying the lines from **/etc/profile** into your **.profile** file. Note that **lpr** does not use the LPRFLAGS setting when used from the Bourne shell. (See Appendix A.)

2 Using FOR:PRO Commands

The most frequently used FOR:PRO commands included in the single-user operating system set of disks are described in this chapter. You'll also find a section describing the connection between selections on the menu system and the FOR:PRO commands which the selections activate.

HOW MENU FUNCTIONS RELATE TO FOR:PRO COMMANDS

If you've used the menu system prior to reading this document, you probably noticed a similarity between the FOR:PRO commands discussed so far and certain selections on the menus. In fact, most of the selections on the System Utilities and System Management menus merely activate FOR:PRO commands. Other selections run shellscripts consisting of a series of commands. For example, the selection "Create a Directory" on the System Utilities menu activates the FOR:PRO command **mkdir**.

Table 2-1 lists some commonly used menu system selections and the FOR:PRO commands that are activated by these selections.

To use the commands with the "/etc" prefix, you must either be in the **/etc** directory or you must type the full pathname for the command. If you are logged in as manager or root, you can use them without typing "/etc" before the command name. These commands are described more fully in Chapter 3.

USING FOR:PRO COMMANDS FOR ROUTINE TASKS

The next few pages describe some basic FOR:PRO commands. You'll use these commands to create and manipulate files and directories, print your work, and perform disk-related activities. You'll also learn how to receive and interpret system status information.

Table 2-1. Menu System Selections and FOR:PRO Commands

Menu Items	FOR:PRO Command Invoked
System Utilities	
Copy a file	**cp**
Delete a file	**rm**
Group ID change of a file	**/etc/chgrp**
List file contents on screen	**more**
Move or rename a file	**mv**
Owner change file ownership	**/etc/chown**
Print file content on printer	**lpr**
Permission change of a file	**chmod**
Copy a directory	**cp**
Create a directory	**mkdir**
Delete a directory	**rmdir**
Go to another directory	**cd**
Group ID change of a directory	**/etc/chgrp**
List directory information	**ls**
Move or rename a directory	**mv**
Name current directory	**pwd**
Owner change of a directory	**/etc/chown**
Permission change of a directory	**chmod**
System Management	
Shutdown computer	**shutdown**
Who is using the computer	**who**
Disk usage	**du**
Percent of disk used	**df**
Display current date and time	**date**
Set date and time	**setdt**
Write a message to a terminal	**write**
Send a message to all terminals	**wall**

METACHARACTERS AS SHELL "SHORTHAND"

The Bourne shell provides a notation method that enables you to use a single special character in a command to represent any number of

files and directories. Use these special characters, called **metacharacters**, in the command line in the following fashion:

*	To match a string of characters of any length, including a null (empty) string.
?	To match any single character except null.
[n-n]	To match any single character within the range indicated in n-n where n-n can be any two letters in the alphabet.

Metacharacters are most useful when copying, moving, deleting, and simply listing file names.

Using *

The asterisk character matches a character string of any length. Suppose you had the following files in a directory:

```
apples1     grapes1     grapes2
apples2     pears       apples.g
oranges     apples.a
```

If you issued the command

```
ls apples*
```

the result would be

```
apples1    apples2    apples.a     apples.g
```

The system lists all files that begin with **apples**. The * matches any characters in the filename that follow the string **apples**.

Using ?

Unlike the asterisk, the question mark metacharacter matches any single character in a file, not an unlimited string. Using the same directory as above, if you issued the command

```
ls apples?
```

the result would be

```
apples1    apples2
```

The system lists any files in the directory whose names begin with the string **apples** and include one other single character. Notice that the files **apples.a** and **apples.g** were not listed because in both cases **apples** was followed by more than one character. If you type

```
ls apples??
```

you receive the results

```
apples.a    apples.g
```

In this case, the two question marks **??** match any two characters (except null) following the **apples** string.

Using [n-n]

The brackets indicate that the match required by the command be any single character between the range n-n. Using the same directory as in the two previous examples, suppose you issued the command

```
ls apples.[a-f]
```

The system would respond only with the name **apples.a**. The file **apples.g** is outside the range enclosed in the brackets.

CREATING AND MODIFYING DIRECTORIES AND FILES

Among the most commonly used commands are those for creating and manipulating directories and files: **mkdir**, **mv**, **cp**, **rm** and **rmdir**. The next few pages describe the syntax and usage of these commands and their most commonly used options.

Creating Directories (**mkdir**)

To create a directory, use the **mkdir** command. You can use the **mkdir** command to create subdirectories in your home directory or in any other directory for which you have write permission.

The easiest way to use mkdir is to first go to the directory where you want to create the new directory. Then issue the mkdir command using the following syntax:

mkdir directory.name ...

You can supply as many directory names as you want, depending on how many directories you are creating. However, remember to use a pathname for any directories not in the current directory. The **mkdir** command automatically creates the . file, representing the directory itself, and the .. file representing the parent directory, in any new directory.

Moving and Renaming Directories and Files (**mv**)

The **mv** command's most commonly used function is to move files and directories to different places. You can also use **mv** to simply rename files and directories without moving them. This is because **mv** removes the file from its original location and puts it in the specified location. If you use **mv** on a file in the current directory, and specify a new filename instead of a pathname as destination, then the file remains in the original directory, but its name is changed.

The syntax for that use of **mv** is

mv { filename.1 / dir.name.1 } { filename.2 / dir.name.2 }

It's also possible to move a file or directory, changing its name in the process. Use the following syntax

mv { old.filename new.pathname / old.dir.name new.pathname }

new.pathname is a complete pathname which specifies where the directory or file is located.

To move a file from the current directory to another, use the syntax

mv [**options**] filename ... directory.name

Supply the name of the file you want moved and the name of the destination directory. Remember, once you move a file to another directory, it no longer exists in its former directory. The options are explained below.

Move one directory to another by using the syntax

mv [**options**] dir.name to.dir.name

When you move a directory, all its files and subdirectories are moved as well. Note that you cannot move a directory to one of its subdirectories.

mv Command Options: You can use one of three options: **-f**, **-i**, or **-**, when invoking **mv**. The **-i** option causes the system to pause if it finds an existing file in the directory with the same name as the file you want to move into that directory. If you type a "y" as the next character in response to the prompt, the system replaces the old file with the new file. If you type any other character in response, the system will not move that file. The **-f** option causes all files to be moved, regardless of any permissions set or the existence of duplicate filenames. The **-f** and **-i** options are mutually exclusive. If you do use them together, **-i** overrides **-f**.

The **-** option interprets all later arguments to **mv** as filenames. This means you can move a file whose name begins with **-**. The **-** option can be used with **-i** or **-f**.

Copying Files and Directories (cp)

The **cp** command offers facilities for copying both files and directories. When you use **cp** to copy a file or directory, the original is not destroyed. Instead, you have two copies of the same data. To make this clearer, in the following syntax descriptions for **cp**, the original file or directory is referred to as the "source," and the new copy or file or directory receiving the copy is called the "destination."

Making a Copy of an Original File: For basic file copying, use the following **cp** syntax:

cp [**options**][**-**] source.file destination.file

This form of **cp** makes a copy of a file in the same directory as the original or in a different directory. The options are described below. The destination file retains the same file permissions as the source file. If the destination file is in the same directory as the source, be sure to give the copy a different name than the source. If either the source or destination file isn't in your working directory, remember to use pathnames.

Here's a shortcut for copying a file from another directory into the current directory.

cp dirname/sourcefile .

Recall that . represents the current directory. The name of the file is not changed when it is copied.

Options of **cp**: On the Fortune system, the **cp** command has a great number of options. As with the **mv** command, you can use the option **-i** to have the system prompt you every time it finds an existing file in the destination directory with the same name as a source file to be copied into the directory. The **-f** forces copying of all source files to the destination directory even when the system encounters duplicate filenames. You can also use the - option to create files with names beginning with a hyphen.

Copying Directories: You can copy a directory into another by using the **-r** option.

cp -r source.dir destination.dir

The **-r** stands for recursive, meaning that the source directory and all of its subdirectories and files are copied into the destination directory. You cannot copy a directory into one of its subdirectories. You must use the **-r** option to completely copy a directory that contains subdirectories, otherwise the system ignores any subdirectories it encounters and only copies the files.

Removing Files and Directories (rm)

The **rm** command removes files and directories from the file system. Its syntax is

rm [-f][-i][-r][-] { filename ... / dir.name }

You can use **rm** to delete more than one file by typing individual filenames or by using metacharacters. Due to the serious nature of **rm**, you should be very careful when using this command. Unless you use the **-i** option, the only time **rm** prompts during file deletion is when it encounters a file for which you do not have write permission.

> NOTE
>
> You do not need write permission on a file to delete it, as long as you have write permission for the directory in which the file resides. If you try to delete a file for which you don't have write access, the system simply informs you of the file's permission mode and then asks again if you want to delete it. All you have to do is type **y** and the system deletes the file.

Options of **rm**: The safest way to use **rm** for deleting files is to use the command with the **-i** option, as in

rm -i *filename* ...

In this case, the system prompts, "rm:remove filename?" and waits for your response. Type **y** to remove the file. Any other response cancels the deletion.

To force **rm** to delete all files without displaying prompts or error messages (even on write-protected files), use the **-f** option, as in

rm -f

Once again, **-i** and **-f** are mutually exclusive, with **-i** taking precedence if you mistakenly use them in the same command.

Removing Directories with **rm**: To remove all files and subdirectories in a directory, use the recursive option of **rm**, as in

rm -r -i *dir.name*

In this case, **rm** searches for and deletes first all files, then all subdirectories beneath the named directory.

To give you an idea of the danger of **rm -r**, if you issued the command

rm -r *

the system would delete all files and directories from the current directory on down the file hierarchy. If you were to issue such a command from /, you could delete the entire contents of the hard disk! So be very cautious using **rm -r**, particularly with metacharacters. Use the **-i** option whenever you use **-r**, just to be safe.

Removing Directories with rmdir

You can use the **rmdir** command to delete directories, provided the directories do not contain any files. The syntax of **rmdir** is

rmdir dir.name

If you created directories and didn't put any files into them, you can use **rmdir** to delete them. However, if you use **rmdir** to delete a directory that does contain files, you'll receive an error message, and the system will ignore the command. You must delete the directory's contents before you can delete it with **rmdir**.

MANIPULATING FILES ON FLEXIBLE DISKS

FOR:PRO provides three commands for handling flexible disks: **mount**, **umount**, and **format**. These commands, along with **cp**, help you store and retrieve user files on flexible disks. Note that to install and backup any Fortune Systems' software, you must use the Product Maintenance menu.

How Information is Organized on a Flexible Disk

Each flexible disk contains a separate file system with the directory **/f** at the top of the hierarchy. When you copy or move files to the flexible disk, they are listed under **/f**.

The file system on the hard disk includes an empty directory also called **f** listed directly under /. When you mount a flexible disk, the system "logically" attaches the files on the flexible disk to the **/f** directory on the hard disk. When you unmount the flexible disk, the association between the flexible disk and **/f** on the hard disk is removed.

Formatting a Flexible Disk

Before using a flexible disk, you must format it. During the formatting process, the system arranges the surface of the disk, so that data can be stored on it. The surface is divided into areas called **blocks**. Individual blocks hold 1024 bytes or characters, on the FOR:PRO operating system.

> NOTE
>
> An attempt to access an unformatted disk will cause the system to hang. If this happens, reset the system by pushing the reset switch.

During formatting, the system also creates the **configuration block**, the first block on the disk. The configuration block describes the size of the disk, the date the disk was formatted, and other status information.

Using the **format** Command: Use the **format** command to prepare brand new disks or to reformat a used flexible disk. The **format** command erases any files that might exist on the disk. Therefore, do not format a disk that contains files you want to save. The syntax of **format** is

format [-k][other options] /dev/fd02

The **-k** option of the format command causes the system to save information in the configuration block about any bad blocks encountered in the formatting process. If you're reformatting disks, you may want to use **format -k**, because used disks are more likely to have bad blocks than new ones.

After you've formatted the disk, you should use the command **mkfs** to make a file system on the flexible disk. Type

mkfs -a /dev/fd02

This command creates a file system on the flexible disk. While running **mkfs**, the system will display some messages.

Using **mount** and **umount** with a Flexible Disk

The **mount** and **umount** commands tell the system that you want to load or unload a removable file system. Before issuing the mount command, place the flexible disk in the drive. Then use this syntax to mount the flexible disk

/etc/mount /dev/fd02 /f

Since **fd02** is a special device file, it is contained in the **/dev** directory. This is why you use the full pathname **/dev/fd02**. The system attaches the file system on the disk in drive **fd02** to the directory **/f** on the hard disk.

> NOTE
>
> If you followed the directions on setting up a **.profile** file in your home directory, you don't need to type the **/etc** prefix. Just type **mount** or **umount** plus the proper arguments.

To unmount a file system, use the following syntax:

/etc/umount /dev/fd02

Again, the **/etc** prefix is not necessary if your path has been changed to include **/etc**. This unmounts the files listed under **/f** on the hard disk. After typing the command, remove the disk from the drive.

> NOTE
>
> Do not remove the flexible disk from the drive before issuing the **umount** command. Such an action could damage the file system on the hard disk.

Using **cp** with the Flexible Disk

To copy files to the flexible disk, mount the disk and then use **cp** with the following syntax:

cp file1 [file2] ... /f

You can copy as many files as you want onto the flexible disk, provided that there is enough space on the disk. Use metacharacters with **cp** as shortcuts; don't forget to specify pathnames for files not in the current directory. To look at the contents of a mounted flexible disk, just type

ll /f

or

ls -a /f

The second format displays all special "." files as well as other files.

Copying Files to a Flexible Disk

To copy a directory and all its contents to the flexible disk, use the **-r** option:

cp -r dirname **/f**

For example, to copy the contents of the directory **/u/greg/memos**, type

cp -r /u/greg/memos /f

Copying Files from a Flexible Disk

To copy files from the flexible disk onto the hard disk, first mount the disk, then use this syntax:

cp /f/filename directoryname

The directoryname represents a relative pathname, and specifies the directory name where the file is to be copied. The directoryname argument should not contain the name of the file being copied. You can specify more than one filename to be copied from a flexible disk. Suppose you want to copy the file **may** from a flexible disk to the **/u/greg/memos** directory on the hard disk. You would use this format of **cp**:

cp /f/may /u/greg/memos

To copy all the files and any directories on a flexible disk to a directory **/u/greg/index** on the hard disk, use

cp -r /f/* /u/greg/index

All the files under **/f**, including any directories (except special files beginning with "."), are copied to the directory **/u/greg/index**. To copy the entire contents of the flexible disk to your current directory, type

cp -r /f/* .

To copy all the files and directories in the current directory to a flexible disk, use this form of **cp**:

cp -r * /f

The contents of your current directory, including any subdirectories and all their contents (except for special files, like **.profile**, which have to be copied individually) are copied to the flexible disk. Other useful options are **V**, **o**, **t**, **B**, and **R**. See the MAN page entry for **cp** in Part 2 of this book.

PRINTING YOUR FILES

FOR:PRO provides the following printer-related commands, which enable you to print files according to your specifications, and then manipulate these files once they are placed in the print queue:

Command	Description
lpr	Places the file you request in print queue for printing.
pr	Displays a paginated file on the screen, or, when used with **lpr**, causes the printing of a paginated file.
lpq	Displays the list of files waiting to be printed.
lprm	Removes files from the list of those waiting to be printed.
lpdun	Interrupts and resumes printing. Sets printer defaults and creates new printer directories.
lpmv	Changes the order of files waiting to be printed.

Printing Files with the **lpr** Command

The **lpr** command places requested files in a list called a **print queue** for printing. There are many options to the **lpr** command, ranging from those that serve the simplest printing needs to others used for more complex printing equipment. Only the basic options of **lpr** are explained here. Refer to Appendix A for information related to **lpr** and its associated utilities.

Simple Print Requests Using **lpr** and **pr**: If you simply want a printout of a file, without page breaks or page numbers, use **lpr** as shown below:

lpr [**options**] filename ...

If you want page numbers on your printout, use a combination of **pr** and **lpr**. The command **pr** organizes the file into pages with the file title and the page number displayed at the top of each page. The syntax of **pr** is

pr filename

If you issue **pr** in this form, your file is displayed on the screen in the paginated format. To print a paginated file, you must use **pr** and **lpr** connected by a **pipe**, as in

pr filename **|** **lpr**

Some Useful Options of **lpr**

Some particularly useful options of **lpr** are summarized below. They are described in more detail in the following paragraphs and in Appendix A.

Option	Description
-b	Adds a banner preceding the text of the printed file containing the name you specify in the center.
-d	Causes the printer to stop after printing a page so you can load single sheet paper (only for printers with friction feed or single sheet feeders).
-h	Removes the leading banner from a printed file.
-n	Requests more than one copy of a file.
-o	Changes the owner name printed on the banner page.
-p	Indicates the number of the printer you want to use on a multiple printer system.
+P	Specifies pitch of a printed file.
+L	Changes the print position of the left margin of a file.
+V	Determines the number of lines printed per inch.

NOTE

You can use more than one of these options on the command line. If you do, make sure you type all options beginning with **-** before typing those beginning with **+**. Otherwise, the **+** options override those beginning with a **-**.

Banner-related options: If your login shell is **/bin/sh** instead of **/bin/menu**, the first page of every printout created by **lpr** usually consists of a banner containing the name of the person who printed the file, the date of printing, and the name of the file. Users with menu login shells (**/bin/menu**) will not have banner pages on their printouts. Banners are the only method of identifying the owner of a printout. You can use the **-b** option to produce a banner containing both your account name and another specified name. The format for this version of **lpr** is

lpr -b user.name filename

This might be useful if several people use the same account, like ENG, and you want to identify a printout as belonging to you, for example, MARVIN. The printout would show both the account name you were using, ENG, plus the user name you specified, MARVIN. In this case, the command line would be

lpr -b MARVIN report

To eliminate the banner (header page) entirely, use the **-h** option:

lpr -h filename

> NOTE
>
> The default **lpr** setting for users with menu login shells is to suppress the banner on each print job. Use the **-b** option to override the default. (See Appendix A for details.)

Duplicate copies: You can print out more than one copy of a file by issuing the command:

lpr -n xx filename

xx is an integer specifying the number of copies to be printed.

Specifying printer numbers: If you have more than one printer on your system, each will have a printer number assigned. (See

FOR:PRO Installation Instructions for details on defining printers.) The default printer number is 1. If you want to print a file on a printer other than printer 1, type

lpr -p printer.no filename

printer.no represents the number of the printer you wish to use.

Printing one sheet at a time: The **-d** option of **lpr** causes the printer to pause after printing each page of a file. This enables you to feed single sheets of paper like letterhead stationery or other preprinted forms into the printer while it is inactive. After you've loaded the next sheet, continue printing by issuing the command **lpdun.**

Specifying Pitch: The **+P** option of **lpr** lets you specify the pitch (characters per inch) for the individual print job. Its syntax is

lpr +Pnn filename

nn is either 10, 12, or 15 characters per inch. The default is 10.

Changing the Left Margin: Use the **+L** option to specify the left margin of the printout. When the printer is inactive, its printhead rests in the leftmost print position which is position 0 on printers which have a margin ruler. Normally the left margin of a file begins at this print location.

If you specify **+L**, the left margin changes without having to reposition the paper. Its syntax is

lpr +Lnn filename

nn represents the number of characters to the right of the printhead where the left margin starts. If you use both the **+P** and **+L** options on the command line, be sure to specify **+P** before you specify **+L.**

Changing Lines per Inch: The **+V** option of **lpr** lets you specify the number of vertical lines per inch you want on your printout. Its syntax is

lpr +Vnn filename

nn represents lines per inch. The default is 6 lines per inch. Other choices are 8 or 10 lines per inch.

> NOTE
>
> The **+L**, **+P**, and **+V** options are effective on a per print job basis. To change specifications for all print jobs, you must change the defaults. To do this, see Appendix A.

The Print Queue Command (**lpq**)

When you use **lpr** to send a file to the printer, the file is placed in the print queue. The system maintains the queue for the purpose of determining print priorities. By default, the first file sent to the printer is the first printed. However, you can change the order of files in the queue by using **lpmv**.

Each printer attached to the Fortune system has its own print queue. You can view the contents of the queue by issuing the **lpq** command, using the following syntax:

lpq [-aqsp]

The table below explains the **lpq** options and their uses.

Option	Use
-p nn	The **-p** nn option lets you view the queue for the printer whose number is represented by nn. The default is printer 1. On a multiple printer system, use **lpq -p** nn to view the queue for a printer other than printer 1.
-a	To view the queues of all printers on the system, type **lpq -a**.
-s	The **-s** option displays a status table, in place of, or in addition to, the print queue. It is explained further in the following pages.
-q	When used with **-s** in the form **lpq -qs**, displays both the print queue and status table.

Simply typing **lpq** displays the queue for printer #1. That display should resemble the following:

```
Pr# Owner  QID Type  Size  Filename  Jobname    Comment  Copies
1   steff  419 File  5183  cab00419  bridge.2.3           1
     ruby  421 File 13295  cab00421  prog.15              1
```

The individual fields are defined below.

Field	Description
Pr#	Printer number - given only for first job in each queue.
Owner	Owner of the file in the queue.
QID	Queue identification number.
Type	Type of entry. The possible values for this field are:
	doc Word processing document
	file Ordinary file created by an editor
	pipe File created through a pipe
	shell File created through a shellscript
Size	Size of the file in bytes.
Filename	The name the system gives the file.
Jobname	The job name of file being printed, usually the same as the actual file name.
Comment	Any comment typed on the comment line of a word processing document summary.
Copies	Number of printed copies requested.

In addition to the print queue, you can also view the state of the printer by issuing the **lpq** command and the option **-s**. The result is a table that describes the printer's current mode of operation. The display you receive after issuing **lpq -s** should resemble the one on the next page.

```
Pr# State   Ribbon   Wheel   Forms  Line  Page  PID  Waiting on
  1 Active  Black    Round   paper  256   66    422
```

Here is a description of the individual fields.

Field	Description
Pr#	Printer number.
State	Current operational mode of the printer. Most common states are: active, suspended, and idle.
Ribbon	Type of ribbon currently loaded.
Wheel	Type of printwheel loaded on the printer.
Forms	Type of forms loaded.
Line	Maximum width of a line in characters.
Page	Maximum length of a page in lines.
PID	The process id number of the filter currently handling printing.
Waiting on	A message explaining why printing is suspended.

To display both the print queue and the printer state, type

lpq -ds

Removing Files from the Print Queue (**lprm**)

To remove a file from the print queue or to terminate a job being printed, use the **lprm** command. Before issuing **lprm**, display the print queue; make note of the QID of the file you wish to delete. Then issue the **lprm** command using the following syntax:

lprm [**-p** nn] QID.number

where QID.number represents the QID of the file to be deleted. The system then alerts you that it is deleting the file. If the

deleted file was the one currently printing, printing halts, and the message, "Printing Cancelled by Operator" appears as the last line on the printout.

Use the **-p** nn option to kill a file queued to a printer other than printer 1. The nn represents the number of the printer.

> NOTE
>
> If a file is almost done printing when you issue **lprm**, it may finish printing anyway. This is because many printers have a large buffer size (typically 2000 characters) and the buffer must empty before the **lprm** takes effect.

To remove all jobs queued by the same person, type

lprm owner.name

where owner.name is the login name of the person who sent the files to the printer.

Suspending and Resuming Printing (**lpdun**)

The command **lpdun** is used to suspend and restart a print job. You'll most often interrupt printing to fix buckling paper, change a ribbon, or change a printwheel on a character printer. Using the **lpdun** command, you can resume printing at the last page printed before the paper or ribbon problem occurred.

The proper syntax for suspending a print job is

lpdun -i [**-p** nn]

When you issue this command, the printer stops printing the current file. Use the **-p** nn option to interrupt printing on a printer other than printer 1.

Resume printing by issuing **lpdun** using the following syntax:

lpdun [**-n** pp]

You'll often use the **-n** option of **lpdun**, where pp represents the number of pages to be reprinted. For example, suppose paper started buckling on the fourteenth page of a file, mangling this page and the next, before you were able to suspend printing. Your last good page was the thirteenth page. In this case, you would type **lpdun -n2** to reprint the last 2 pages. However, you could reprint up to nine pages. If pp is more than the number of pages that have been printed, printing starts at the beginning of the job. To restart the printer at the point it stopped, simply type **lpdun**.

On jobs that are being printed one sheet at a time (started with **lpr -d**), issue **lpdun** after you've successfully fed and positioned the paper in the printer.

In addition, **lpdun** has other useful options explained in the MAN page entry for **lprm**.

SYSTEM STATUS COMMANDS

The FOR:PRO operating system provides a number of system status commands, among them are the following:

Command	Description
date	Presents the date and time.
who	Tells you who is currently logged into the system.
ps	Presents a table of programs on which your account is currently running.
du	Displays the amount of space in blocks a file takes up.
df	Presents a table indicating how much free space remains on a disk.

The **date** and **who** commands were described in the previous chapter. The **ps**, **du**, and **df** commands display tables concerning the operational state of the computer; they are especially useful if you are the manager of a multiuser system. The major aspects of these commands are discussed on the next pages.

Using **ps** to Check Process Status

The **ps** command displays a table describing the state of all processes on the system. You might think of a process as a currently executing program. The process table displayed by **ps** lists any system and user-created programs that are currently running.

The syntax of **ps** is

ps [-a][-l][other options][pid]

To see the short form of the process table, type **ps.** The resulting display shows all processes running under your current account name. (Note that if you are currently logged in at two terminals at the same time, the process table shows the processes running under the same account on both terminals.) The resulting short process table will resemble the following:

```
PID  TTY  TIME  CMD
29    03  0:04  -sh
```

Here's an explanation of the fields in the short process table.

Field	Definition
PID	The process ID number assigned by the system to the running process.
TTY	The terminal controlling the running process.
TIME	The amount of computer time used so far to execute the process.
CMD	The command line typed to start the process. Note that the information listed under CMD may not always look exactly like what you typed. If you run a shellscript or type a command that uses pipes, you'll see a process number for each pipe under CMD.

In the above example, the account that issued the ps command is currently running the Bourne shell (-sh) from terminal TTY 03. The system assigned the process ID 29 to the shell and has spent four seconds of CPU time running the shell.

> NOTE
>
> If you are logged in as manager or root, you will also see an entry for the ps process itself, even when you issue the "short form" (no options) of **ps**.

If you are the system manager, you'll probably want to view a table listing all processes initiated from every terminal on your system. The **-a** option lets you view all processes executing from all terminals. Type

ps -a

The resulting display resembles this one.

```
PID  TTY  TIME  CMD
28   co   0:07  /m/menu/bin/msh -p /m/menu/control
                -l user -s
29   03   0:04  -sh
59   co   0:00  sh -c TERM=FT^Jexport TERM^JCLEARHOME
                =20^L10^^^Jexport CLEARHOM
```

In this case, terminal 03 is running the Bourne shell, while the console is running shellscripts from the menu system.

Managing Disk Space with **df** and **du**

If you have a single-user system, or are the system manager, you'll want to monitor disk usage from time to time on your system to prevent the hard disk from running out of space. Two commands, **du** and **df**, provide information relevant to disk usage.

Displaying Used and Free Space: The **df** command displays a table depicting the number of free blocks available on the file system. You can use **df** to determine the free space not only on the hard disk, but also on a mounted flexible disk.

The syntax of **df** is

df [-f][other options] [filesystem ...][file...]

filesystem represents the device containing the particular file system, /dev/hd02 for the hard disk, or /dev/fd02 for a flexible disk. (The flexible disk doesn't have to be mounted, just inserted, to use **df**.) If you do not specify a filesystem, the **df** table is shown for the root file system on the hard disk.

The df Table Display: When you type **df**, the resulting table resembles this one.

Filesystem	Mounted on	kbytes	used	free	% used
/dev/hd02	/	7895	5342	2553	68%

The fields of the table are described below.

Field	Definition
Filesystem	Device containing the file system.
Mounted on	The top directory in the file system hierarchy on the specified device. In the example above, the entry is /, representing the root directory. For a mounted flexible disk, the entry **/f** appears in this field.
kbytes	The total number of kilobytes of storage space available on the disk. (A kilobyte is 1024 bytes. Also, one **block** is equal to 1024 bytes.)
used	The number of kilobytes used or taken up by files.
free	The number of kilobytes still unused.
% used	The percentage of disk space used.

The most significant field in the display is % used. You should check this field every day, perhaps more often if your system is heavily used. You shouldn't let the amount of used space on the hard disk exceed 90 percent, because hard disk efficiency is reduced when free space on the disk gets closer to zero. Should this happen, delete files you don't need and backup on a flexible

disk other files that you don't use routinely but want to save. The **-f** option lets you view just the % used field. If you type **df -f**, you'll receive a message like

"68% of the available space is in use"

Displaying Disk Usage (**du**)

The **du** command shows how much space a specified directory takes up on the disk in terms of blocks. (Remember that one block is one kilobyte, or 1024 bytes.) If you are the system manager or have your own single-user system, use **du** to determine the size of files. You may want to move any large files that are seldom used but still must be saved onto a flexible disk.

The syntax of **du** is

du [**-a**][**-s**][dir.name...]

If you type **du** dir.name, the system presents a display resembling the following

56 /u/ruby

where **/u/ruby** is the directory specified for dir.name.

If you simply type **du**, the resulting display may resemble

71

Since you haven't supplied a name, the system gives you the disk usage of the current directory (.). If the current directory contains other directories, **du** (with no options or arguments) displays the size and names of all directories within the requested directory, plus any of their subdirectories.

To view just a total of blocks used by a directory, use the **-s** option of **du**, as in

du -s dir.name

To see how many blocks the entire file system takes up, type

du -s /

Displaying File Size in Blocks: The **-a** option of **du** gives the disk usage for all files in the specified directory. For example, suppose you wanted to view the disk usage for all files in the directory **/u/ruby**. You would type

```
du -a /u/ruby
```

The system's response might be

```
4  /u/ruby/a.out
3  /u/ruby/hist.list
1  /u/ruby/hist.list.fr
1  /u/ruby/hist.list.dc
4  /u/ruby/myprog.p
```

If you want to view the disk usage of all files on the system, type

```
du -a / | more
```

since the list generated will appear on more than one screen.

USING THE SORT COMMAND

The **sort** command sorts information in existing files (typically lists) in alphabetic and numeric order. It is particularly useful for sorting files such as telephone lists, price lists, inventories, and statistics.

How **sort** Works

The **sort** command considers each line in a list to be composed of fields: strings of alphanumeric characters, separated from others on the same line by spaces, tabs, or other delimiters. Since sorting is done on a field basis, you don't have to set up the data in a file in columns.

The next paragraphs use the file, **bookfile**, as an example. You may want to use it for practice with **sort**. **bookfile** is the type of list a bookseller might use to keep an inventory. It lists anthologies of the works of famous authors, the price for each book, and current inventory. It has four fields on each line: first name, last name, price, and quantity.

```
William Shakespeare          24.98     14
Washington Irving            10.98     10
Emily Bronte                  5.98     85
Leo Tolstoy                  20.00      6
Walt Whitman                  3.35    103
Charlotte Bronte              5.98     71
Alexandre Dumas              12.00     28
```

You can have the system perform the sort on any or all of the four fields.

Suppose you wanted the list rearranged alphabetically by last names. When issuing the **sort** command, you'd designate that the last name field is the field to be sorted. The field used in the sort is called the key field. **sort** looks at the key field on each line, and reorganizes the lines in the file so that the key fields are in alphabetic or numeric order.

Invoking sort

The general syntax of **sort** is

sort [+pos1] [-pos2] [**-o** new.file] [orig.file] ...

where orig.file represents the file or files you want to sort. If you don't specify a filename for orig.file, sort expects you to type in the data you want sorted from your keyboard. When you're finished, type **CTRL-D**. To perform a simple sort on an existing file, type

sort filename

In this case, the system considers each line in filename to be a single field.

> NOTE
>
> Unless you specify otherwise, files are sorted according to the ASCII collating sequence. In this sequence, control characters come first, then special characters (!'#, and so forth), the

> numbers 0-9, uppercase letters, and finally lowercase letters. Tables containing the entire ASCII collating sequence can be found in most programming texts.

Sorting by Key Field

You are more likely to sort by specific fields, rather than by using an entire line. To do this, you must indicate the position of the field on the line. When defining key fields, use the syntax

sort [+pos1] [-pos2] ... orig.file

where +pos1 indicates the starting position and -pos2 the ending position of the field. Sort considers the starting position of the first field on a line to be position 0. The ending character of the first field is position -1. The starting character of the next field is position +1, and so on.

The chart below shows the position indicators for an entry in the bookfile illustrated earlier:

```
William  Shakespeare          24.98    14

+0    -1 +1        -2         +2  -3  +3 -4
```

To sort the file by the author's last name, issue the command

sort +1 -2 bookfile

Sorting is done on the second field, which begins with position +1 and ends with position -2.

You can indicate as many key fields as you want in a single **sort** command. The system sorts key fields in the order specified on the command line, not in the order they appear in the file.

Saving Sorted Lists in a File (-o and >)

The **-o** option of sort lets you save the sorted list in a specified file. When you issue the **sort** command without the **-o** option, the sorted list is by default displayed on your terminal, and exists only as long as it remains on the screen.

However, most of the time, you'll probably want to save the sorted file. To do so use the **-o** option

sort -o new.file [+pos1][-pos2] ... orig.file

orig.file represents the file to be sorted and new.file represents the resulting sorted file. Note that new.file is not displayed on the screen. You'll have to use **cat** or **more** to display new.file. The contents of orig.file are unchanged by the **sort** operation.

Another way to achieve the same results is to type

sort [options] [+pos1][-pos2] ... orig.file > new.file

The > symbol preceding new.file tells the system to direct the sorted list to a file called new.file, rather than display it on the screen. In general, the > symbol can be used after any FOR:PRO command to redirect the output that would otherwise appear on the screen to a specified file.

More **sort** Options

The **sort** command has many options. They are usually specified before the field positions. The general format of **sort**, including options, is

sort [options] [+pos1][-pos2] [**-o** new.file] orig.file

The most useful options for sorting single files are described below:

Option	Definition
-b	Ignores leading blanks in fields.
-f	Treats uppercase and lowercase letters equally.
-n	Performs a numeric sort.
-r	Sorts the list in reverse order.
-o	Saves the sorted list in the specified output file.
-tx	Lets you define field separators other than the default tabs or spaces.

Equal Treatment for Uppercase and Lowercase Letters (**-f**): The **-f** option tells the system to sort an uppercase letter in the same position as the lowercase version of the same letter. Its syntax is:

sort -f [options] [+pos1] [-pos2] ... orig.file

For example, suppose you were sorting a list in which the starting letters of the key field were mixed uppercase and lowercase. Naturally, you'd put the fields beginning with B and b after the a's and before the C's.

However, unless you specify the **-f** option, the system first lists all fields beginning with A-Z then all fields beginning with a-z. Remember, the system uses the ASCII collating sequence, where upppercase letters are higher in the collating order than lowercase letters. The **-f** option makes the system treat uppercase and lowercase letters equally.

Removing Leading Blanks (**-b**): A blank also has a value in the ASCII collating sequence. If you have more than one blank at the beginning of the field, sorting is performed first on the blanks, then on the remaining characters in the field, which may result in an incorrect sort. Use the **-b** option to remove any leading blanks from fields.

You can use **-b** in either of two ways. If you type

sort -b [options] [+pos1] [-pos2] ... orig.file

leading blanks are removed from all fields. To remove leading blanks from a specific field, type

sort [+pos1]**b** orig.file

Leading blanks are now removed from the field indicated in +pos1. In the syntax form above, a letter following a field position is called a **flag**, rather than an option. Note that when you use a flag, it overrides other options.

Numeric Sorts (**-n**): Numbers in fields can be regarded in two ways, as characters or as numeric values. A number regarded as a string of characters such as a phone number or serial number has no significant numeric value. Numbers such as prices or quantities obviously have significant numeric value.

To sort a field in which numbers have values, use the **-n** option to indicate you want to perform a numeric sort. If you use the following syntax:

sort -n [options] [+pos1] [-pos2] ... orig.file

all fields are treated as numeric fields. To limit numeric sorting to a specified field, use **-n** as a flag, as in

sort [+pos1]**-n** ... filename

where **+**pos1 represents the field to be sorted numerically.

> NOTE
>
> You can use the **-b** option to sort a field in which numbers are treated as characters, provided that the numbers are all equal in length.
>
> For example, zip codes occupy five character positions. If the numbers vary in length, you'll need to use **-n** to achieve the results you want.

Sorting in Reverse Order (-r): You can use the **-r** option to sort files in reverse alphabetic order (Z-A) or from highest numeric value to lowest. If you type

sort -r [options] [+pos1] [-pos2] orig.file [**-o** new.file]

all fields are sorted in reverse alphabetic order. To sort a specific field in reverse alphabetic order, use **r** as a flag following the position indicator of the field. If you wanted to sort a specific field in reverse numeric order, use both **n** and **r** as flags, as in

sort [+pos1]**nr** ... orig.file

Specifying Field Delimiters (-tx): The **-t** option of sort lets you specify a delimiter other than a space or tab to represent field boundaries. Use the following syntax:

sort -tx [options] [+pos1] [-pos2] ... orig.file

x represents the character you want to use as a delimiter for example, # or :). When choosing delimiters, be very careful that the delimiter is not one that has a special meaning to FOR:PRO, such as / or !. If you need to use a special character, precede it with the escape character /, as described in Chapter 2.

COMMUNICATING WITH OTHER USERS

The FOR:PRO single-user operating system includes two commands for sending messages, **write** and **wall**. The **write** command enables you to send a message to another terminal, even to carry on a telephone-like conversation. **wall** sends your message to all terminals.

Communication Between Users (**write**)

The **write** command is very simple to use. Its syntax is

write account.name [ttyname]

You can specify either a user's account name or the tty number of the terminal used by the person with whom you want to communicate.

> NOTE
>
> Normally you don't have to supply the tty number to use write. However, sometimes you may want to contact a person who is logged in on two or more terminals at the same time. You may have to send a message to both terminals in order to contact the individual.

Once you invoked **write**, the system waits for your input at the terminal. Type your message, ending each line with a Return. After you type the first line, the system sends your message to the specified user. To end the message, press **CTRL-D**.

The person you indicated receives a message on the screen specifying your name and tty number plus a "beep" to get the recipient's attention. For example:

```
Message from ruby tty03

    Can you check the printer to see if it has enough paper?
    Thanks.
```

> NOTE
>
> If you send a message to a user who isn't currently logged in, that message will be lost. Therefore, use the **who** command before using **write**, not only to find out who is on the system, but also to see which terminals are in use.

Sending a Message to All Terminals (**wall**)

wall operates according to the same principle as **write**, except that the message you type is sent to every terminal on your Fortune system. To invoke **wall**, type

```
wall
```

As in **write**, the system pauses, waiting for your input. Type the message, indicating its end by pressing the **CTRL-D**. Unlike **write**, the system waits for the CTRL-D, regardless of the message's length, before transmitting it to everyone currently logged in.

A typical message from **wall** might look like

```
Message from manager console ...

    EMERGENCY!! We're 95% full.  Please delete any junk files
    or move files you don't need right now to a flexible disk.
```

Suggestions for Using **write** and **wall**

Use **write** and **wall** with a certain amount of discretion. Ideally, these commands should only be used to communicate vital information, as in the case of the system in danger of becoming

overloaded, or if a shared resource, such as a printer, must be taken offline for servicing. It is a good practice to restrict use of the **wall** command to the manager and root accounts.

Since your messages are transmitted as you type them, they are displayed in the middle of whatever the recipient is currently viewing on the screen. Although the messages are only temporary, it appears as though the message text has been written into the recipient's file, particularly if the recipient is running an application. However, these messages are nondestructive. The recipient need only clear the screen and return to the file to see that the message has not become part of the file and has not disturbed anything.

You may want to put the text of the message into a temporary file so you can edit it before sending it. To do this, use **ed** filename or **cat** filename to put the text into a file, then edit the text using **ed**. Send the message using this syntax:

write username < filename

or

wall < filename

The < symbol directs the text of filename to the indicated FOR:PRO command, for example, **wall** or **write**.

Using **write** Like a Telephone: You can carry on a two-way conversation with another user via **write**. For example, User 1 issues the **write** command and sends a message to User 2 without pressing **CTRL-D**. User 2 invokes **write** and answers User 1's message. This response is promptly displayed on User 1's screen. Both users can continue sending messages in this fashion, until they both press **CTRL-D**.

You should establish a protocol for carrying on two-way conversations, to ensure both users don't write to each other at the same time. When you finish writing to another user, wait for a response before writing again. End each message with a code you have both agreed upon. It is a common practice to end each message in the conversation with the letter "o" for "over." To indicate that you are going to sign off, end your final message with "oo" for "over and out."

3 Advanced Concepts

This chapter describes advanced operations you may want to perform at command level. It also explains in detail more involved UNIX concepts you may have previously encountered. Among them are:

- File and directory permissions
- Changing file and directory ownership
- More about groups
- Changing default erase and kill characters
- Redirection of input and output
- Creating simple shellscripts

PERMISSIONS/ACCESS RIGHTS

The methods used to keep files safe from unauthorized users have many names: file or directory permissions, access rights, user rights, and file protection rights. These terms all refer to the protection settings that can be placed on a file or directory. These settings specify what kind of access different users on the system have to that file or directory.

File Permissions

There are three kinds of access rights for files. Directories are a little different. (See Directory Permissions below.) As used on the FOR:PRO operating system, these are:

r Means the file can be examined or **read** but can't be modified or changed.

w Means you can modify or write to the file. The file can also be edited or changed.

x Means you can run or **execute** the file. Such a file contains instructions, as does a program. You can execute a shellscript by simply typing its name, as long as it has "x" rights.

> NOTE
>
> If a file is **read only**, you must override this protection setting in order to delete it. A message asks whether you want to override protection for the file when you try to remove such a file.

For example, look at this listing, obtained by using the **-l** option of the **ls** command:

```
-rw-rw-r-- 1 laura      604 Jun 14 17:09 change.stuff
-rw-rw-r-- 1 laura     1926 Jun 15 15:41 customs
-rw-rw-r-- 1 laura      210 Jun 15 15:08 ex1
-rwxrw-r-- 1 laura       42 Jun 13 13:58 fd.look
-rwxrw-r-- 1 laura        5 Jun 15 16:44 logout
-rw-rw-r-- 1 laura     1278 Jun 14 15:10 long2
-rw-rw-r-- 1 laura     1578 Jun 14 16:44 new.stuff
-rw-rw-rw- 1 laura      652 Jun 17 16:34 private
-rw-rw-r-- 1 laura      114 Jun 15 16:11 reminder
-rwxrw-r-x 1 laura       47 Jun 15 15:45 u
```

The read, write, and execute permissions are indicated in the left columns of the display. The first hyphen indicates an ordinary file rather than a directory. Nine possible permission slots follow, three for each class of user. Possible values for these permission slots are **r**, **w**, **x**, and **-**. They correspond to **read**, **write**, **execute**, and **no**, or **null** permission, respectively. (The hyphen indicates that a permission is denied.)

User Classes

The three classes of users on the FOR:PRO system are:

User Class	Description
user (u)	The user who created the file or now "owns" the file. The permissions for the owner are always shown first.
Group (g)	Users can be placed in different groups on a FOR:PRO system. A group can have rights which differ from those of the owner and everybody else.

Other (o) — Everybody else, or the general public, is assigned a third set of rights. This is useful when you want to protect certain files from modification, but you want everyone to be able to read those files.

Note that the three classes of users have single-letter abbreviations: **u** is **user**, **g** is **group**, and **o** is **other**. These will come in handy when you need to change a file's access rights.

Directory Permissions

Directory permissions are a bit different from file permissions. Here is what the **r**, **w**, and **x** permissions mean when applied to directories.

Permission	Meaning
r	Permits users to list or read a directory's contents.
w	Permits users to write or copy items to a directory. Thus, they can create files and directories in that directory, copy files and other directories into that directory, and remove files and directories within that directory.
x	Permits users to go to, or search the directory and to copy files from that directory as long as they also have read rights to that directory. Also allows users to execute programs from the directory, as long as they already know the program name.

The hyphen (-) indicates the absence or denial of a permission. That is, the permission slot is set to null.

Changing File/Directory Access Rights (**chmod**)

You'll find it necessary to change the access rights on a file or directory from time to time. For example, you may want to

make a file executable, as in the case of a shellscript, or you may need to add or take away certain access privileges on a particular file or directory. It is important to remember that you have to be the owner of a file or logged in as root or manager to change the access rights on a file or directory.

To change file or directory permissions, use the **chmod** command. This command has a variety of arguments and options. A quick way to learn **chmod** is to memorize the following six letters:

Letter	Means
u	**user** or **owner** (must be you!)
g	**group** (everyone else in your group)
o	**other** (everyone on the system besides owner and group)
r	**read** permission
w	**write** permission
x	**execute** permission

> NOTE
>
> The directory a file resides in must be writable in order to delete a file. If the file you are trying to delete is read only, but is in a writable directory, **rm** prints a message. If you answer **y** in response to it, the file is deleted. Also, if you use **rm -f**, it overrides the file's **read only** status and deletes it anyway.

The **chmod** command uses these six letters in combination with these common operators:

Operator	Meaning
=	Means "assign these rights to"
+	Means "add this right to"
-	Means "take this right away from"

The **chmod** command is used in this form:

chmod {**ugo**} {**-+=**} {**rwx**} filename ...

You can specify more than one letter, more than one permission, and more than one filename in a single **chmod** command.

For example, to assign all users except yourself read only rights to the file **badges** and **payroll**, you'd issue this command:

chmod go=r badges payroll

Of course, you would have to be the owner of both the **badge** and **payroll** files in order to change their access privileges.

Another **chmod** example: Assume you are the user called pubs, and you want to take away write privileges from the general public (the user class other) for a file called **private.** This file currently has **rw** permissions assigned to all user classes. Here's the command line you would use:

chmod o-w private

You are subtracting the write privilege (**w**) from the user class other (**o**).

If you issue the command

ls -l private

you'd see the following display:

```
-rw-rw-r-- 1 pubs       652 Jun 17 16:34 private
```

Changing File/Directory Ownership (**chown**)

When you copy a file from someone else's directory to your own, you become the owner of the copy in your directory. If another user copies a file or directory to your directory, that user will own the copy. If the permissions for the **other** class deny write access, you may need to use the **chown** command to change the ownership of that file or directory so you can modify it.

You must be logged in as manager or root to use **chown**. Use this format to change the ownership of a file or directory:

chown new.owner filename

The name of the new.owner must be a valid existing user account name on the system. The filename can be a pathname, if necessary.

MORE ABOUT GROUPS

Users on a Fortune system can be organized into groups for purposes of restricting access to certain files or directories. All users on the system belong to the group called **users** by default. Assuming you belong to the group called **users**, any file or directory created by you can be assigned a special set of access rights for the rest of the user accounts on the system. Thus, any other account belonging to the **users** group can have special access to your files and directories as discussed previously. Conversely, if you don't want any other user accounts to access your files and directories, simply use **chmod** to take all access rights away from the **group** class.

Default Groups on the System

To find out what groups exist on your system and which users belong to each group, type

cat /etc/group

You may want to create another group for yourself and other people in your department to use. This makes it easier to control access to information that you want only certain people to use. On a FOR:PRO system, the best way to assign special access rights to certain people is to put those users into a group. Remember however, that a user account can only belong to one group at a time.

The procedures for creating and modifying user groups are discussed in detail in Understand Your Fortune System. You can accomplish the same tasks from FOR:PRO level, if necessary, by modifying the **/etc/group** and **/etc/passwd** files.

Changing Group Ownership (**chgrp**)

Like the **chown** command, **chgrp** can only be accessed by the manager or root accounts. Its purpose is to change the group ownership of files and directories. You may want to do this when you need to turn over files to a different group of users.

For example, if the **Acctg** group decides to turn over all the files in the **back.order** directory to the **Audit** group, you'd have to change the ownership of the directory. Use this format of the command:

chgrp new.group { filename / directory.name }

CHANGING ERASE AND KILL CHARACTERS

The default erase and kill characters were discussed in Chapter 1. Should you need to change either or both of them, use the **stty** command, followed by the word **erase** or **kill**, and the new character sequence you want to use.

stty erase new.erase.char

stty kill new.kill.char

You can find out what the default settings are for the erase and kill characters by typing

stty everything

This displays the various settings for your particular terminal, most of which may not mean much at first glance. If you look at the bottom two lines, you'll see listings for the current erase and kill characters. The defaults are:

```
erase: ^H  (CTRL-H)
kill:  ^X  (CTRL-X)
```

In this case the "^" means CTRL.

Stopping/Restarting Screen Displays

Sometimes you may accidentally use **cat** on a long file instead of **more**. If you want to stop a screen display, type CTRL-S. To restart the display, type CTRL-Q. This is also known as XON/XOFF protocol, or flow control.

REDIRECTION OF INPUT AND OUTPUT

Command input and/or output can be redirected. You might redirect output so it can be used by another command. Input can come from a file, rather than the keyboard, as it normally does. This section describes how redirection can be done.

Redirection of Output

Output produced by FOR:PRO commands is normally sent to the screen, which is called the standard output. When you run the **who** command, for example, the resulting list of who is on the system appears on your screen. However, if you want to send the listing to a file, you can redirect the output with a "greater than" sign (>) followed by the destination filename. For example,

```
who > whofile
```

directs the output from the **who** command into a file named **whofile** in the current directory. The output is not displayed on the screen. If **whofile** already exists, its contents will be replaced by the output from the **who** command.

If you wish to append a command's output to an existing file rather than overwriting the current contents, use two greater than signs (>>). For example, you might start a list of thoughts for future reference with the following commands:

```
date > ideas
echo 'With the echo command and redirection
of output I can easily jot down
ideas in a file for future reference.' >> ideas
```

The **date > ideas** command line creates a file named **ideas** and places its output, the current date, within it. The **echo** command

appends the first idea to the file. (If you follow the **echo** command with multiple lines of text as shown here, you will receive a > prompt at the start of the second and third lines, indicating that the shell is waiting for the rest of the command to be entered.) Additional entries could be appended similarly. For example,

```
date >> ideas
echo 'I should keep a list of things I need
to do in a separate file' >> ideas
```

Redirection of Input

Programs often expect input from the keyboard, called the **standard input**. Just as output can be redirected, input can be redirected so that it comes from a file or another device rather than from the keyboard.

Suppose, for example, that you have written a novel, the chapters of which are stored in 34 files. You now decide to change the main characters' names from Paul, Mary, Jack, and Susan to Pierre, Marilyn, Jacques, and Victoria. Rather than entering the necessary editing commands 34 times while editing the 34 files, you can store the required commands in a file and use the file as input to the editor. This file, which might be called **edits**, could contain the following:

```
1,$s/Paul/Pierre/g
1,$s/Mary/Marilyn/g
1,$s/Jack/Jacques/g
1,$s/Susan/Victoria/g
w
q
```

The first four of these commands tell the **ed** editor to start at the first line of the file being edited and to continue to the bottom line ($), and for each line in that range to substitute the second name shown for the first name. The **g**, which stands for **global**, changes every occurrence of the indicated string throughout the line. Without the **g**, only the first occurrence on each line would be replaced.

Once the necessary editing commands have been stored in the file **edits**, you can change the names in Chapter 1 of the novel, which is contained in the file **ch.01**, by typing

```
ed ch.01 < edits
```

The result is exactly the same as if you had typed

```
ed ch.01
```

and then proceeded to type the lines contained in **edits** from your keyboard.

It would be tedious to make changes in all the files with the method just described; you would have to type 34 command lines. Bourne shell programs by default take input from the keyboard, but you can simplify the editing process by sending these 34 commands to the shell from a file instead. First, put the names of the files to be edited into a file named **shell.input** using the command

```
ls ch.* > shell.input
```

The **shell.input** file now contains 34 lines, each of which is the filename of one of the chapters. Edit **shell.input** using the following commands:

```
ed shell.input
1,$s/^/ed /
1,$s/$/ < edits/
w
q
```

The second of these commands tells the editor to insert the string "ed " at the start of each line. Similarly, the next command instructs the editor to place the string " < edits" at the end of each line. When surrounded by slashes, the $ represents the end of a line just as the ^ represents the start of a line. After editing with these commands, the lines of **shell.input** look like this

```
ed ch.01 < edits
ed ch.02 < edits
```

and so on.

To initiate a shell that will take its input from the file you just created, **shell.input**, use this command:

```
sh < shell.input
```

A new Bourne shell is started up by this command. When the shell finishes executing the commands in **shell.input** it will die, and the original shell will then give a prompt indicating it is ready for another instruction.

Creating a Shellscript

Instead of creating a new shell that takes its input from a file like **shell.input**, you can instruct the original shell to directly execute the commands in the file. To do so, you would first make the file executable by typing

```
chmod +x shell.input
```

Then, simply typing the filename

```
shell.input
```

would cause the commands in **shell.input** to be executed. Command files like **shell.input** are called **shellscripts**, and are useful when a sequence of commands must be executed numerous times. For example, FOR:PRO uses the shellscript named **rc** to perform part of the system start-up procedure every time you power up the Fortune system. See More On Shellscripts in this section.

PIPES

In Chapter 1, you saw how a **pipe** (|) could be used to send a long directory listing through the **more** command. A pipe connects one program's output channel to another program's input channel. The result is the same as redirecting the first program's output to a file, and then redirecting the second program's input from that file, but the pipe process is more efficient.

Suppose, for example, that you want to examine a long listing of all files on your system. The command

```
ls -lR /
```

produces the desired listing, but it flashes by too quickly to be read. To make the listing appear a screenful at a time, pipe it through **more**

```
ls -lR / | more
```

Similarly, the listing could be sent to the printer by piping it through lpr

```
ls -lR / | lpr
```

Multiple Pipes

Multiple pipes are quite useful. For example, suppose you have a list of names and phone numbers in a file named phone. The list is in no particular order because new names have been appended as they became available.

```
Thomas Decatur          400-846-2412
Barbara Danforth        200-347-9825
Paula Pearson           800-968-2135
Bill Brodsky            200-845-7623
```

To produce a printed listing with last names in alphabetical order, including the date of the printout, a suitable heading, and line numbers, you can use the following command:

```
sort +1 -2 phone | cat -n | pr -h "Phone Numbers" | lpr
```

The +1 -2 sequence in the **sort** indicates that the second field of each line in the **phone** file (the last name field) should be used for sorting (see Chapter 2 for more information on how **sort** works). The first **pipe** symbol sends the output from the sort command to **cat**, which adds line numbers when used with the **-n** option. The **pr** command divides its input into pages, putting a heading at the top of each page which includes the date and time, the input filename (if any), and a page number. In this case, there is no input filename, because the input comes from a pipe rather than a file, so the **-h** (for "header") option has been used to include the character string "Phone Numbers" in the heading. The output from **pr** is piped into **lpr**, which prints the text. The result is shown on the next page.

```
Sep  1 12:16 1983  Phone Numbers Page 1

    1   Bill Brodsky            200-845-7623
    2   Barbara Danforth        200-347-9825
    3   Thomas Decatur          400-846-2412
    4   Paula Pearson           800-968-2135
```

More on Shellscripts

Shellscripts have many uses and are great timesavers. Mounting and unmounting disks are common operations, and are particularly suited to shellscripts. Earlier in Chapter 1, you saw an example of a shellscript that performed those disk-handling operations. Suppose you used the **ed** editor or the **cat** command to put the following command lines in a file called **mu**:

```
mount /dev/fd02 /f
ll /f
cd /f
pwd
cd
umount /dev/fd02
```

You could then execute this series of commands by typing the command

```
sh mu
```

Alternatively, you could make the command file executable (**chmod +x mu**) and then simply type its name, **mu**, at shell level.

Now let's get fancy. Suppose you want to make a shellscript that will automatically back up all the files in your home directory to a flexible disk. This is probably a good idea because you want to back up your directory on a daily basis anyway.

```
cd $HOME
mount /dev/fd02 /f
cp -rtV * /f
ll /f
echo 'The Backup Is Now Complete.'
cd $HOME
umount /dev/fd02
echo 'Please Remove Your Disk.'
echo 'Bye!'
```

Using the **$HOME** variable is a precaution to make sure you are in your home directory. The **echo** command lines let you know what's going on by displaying the text enclosed in quotes.

Using Variables

Suppose you want to write a shellscript that prints files with certain printer attributes, like a different pitch or vertical spacing. It would be helpful to have a way of specifying the name of a file to be printed each time you run the shellscript. Using variables within a shellscript gives you just that flexibility. Variables are indicated by a dollar sign **$** followed by a number, or a string. For example, **$1**, **$2**, and **$ans** are typical shell variables. Here's an example of a shellscript (called **print**) that uses a variable:

```
lpr +P10 +V8 $1
```

To use this shellscript, type its name, followed by the name of the file to be printed, as in

```
print chapter1.o
```

(Remember that the file must be executable to be used this way. To make it executable, type **chmod +x print.**)

Using Conditionals

When writing shellscripts, you may want to give yourself or other users the option of doing some particular action. For example, when mounting a disk, you might ask if the user wants to see what's on the flexible disk. In this case, you would have to perform one action if they wanted a list, and another if

they did not. That calls for the use of a **conditional**, or **if...then...else**, statement. Often, the two alternate paths that can be taken in a conditional statement are called **branches**, indicating that the program can branch to one section or another, depending on some particular condition. For example, if you ask a question and the user answers "yes," the program takes the **then** branch but if the user answers "no," the **else** branch is taken instead.

Here is an example of using a conditional in a simple shellscript:

```
mount /dev/fd02 /f
echo Disk is Not Mounted
echo "Do you want to see what's on the flexible disk?  (yes
    or no)"
read ans
if test "$ans" = "yes"
then ls -las /f | more
else echo "Bye!"
fi
exit
```

The **test** line checks the value of the **$ans** variable. If it is **yes**, the list operation is performed, if not, the word "Bye!" is echoed. The test operation has many options which you can read about in many commercially available UNIX texts, some of which are recommended below.

For More Information

To learn more about shellscripts, consult one of the following books:

S.R. Bourne. The Unix System: Addison-Wesley Publishing Co., 1982.

McGilton, H., Morgan, R. Introducing the Unix System: McGraw-Hill Book Co., 1983.

4 Single-User Operating System

The three flexible disks labeled FOR:PRO Single-User Operating System are packaged with FOR:PRO Installation Instructions. They are distinguishable from the other command and utility sets available on the Fortune System: Development Utilities, and Language Development Tools. You may also have a multiuser operating system master disk which allows you to upgrade your system from one to multiple users.

This chapter summarizes the commands and system files and directories on the single-user operating system disks.

SYSTEM FILES AND DIRECTORIES

The / directory, the root of the hard disk file system, also contains some special files. This section briefly summarizes the purpose of the files and directories located in the root directory.

If you list the contents of the root directory using the command **ls -las**, you will see something like the following:

```
/:
total 126
      1 drwxrwxr-x14 root        304 Jul 13 16:19 .
      1 drwxrwxr-x14 root        304 Jul 13 16:19 ..
      1 -r--r--r-- 2 bin          90 May  9 20:30 .profile
      1 drwxrwxr-x 2 root        832 Jul  7 16:30 bin
      1 drwxr-xr-x 2 root        704 Jul 22 10:42 dev
      1 drwxrwxr-x 3 root       1024 Aug 11 09:25 etc
      1 drwxrwxrwx 2 root         64 Jul  8 17:08 f
      1 drwxrwxr-x 2 bin          32 Jun 27 11:06 h
      2 drwxrwxrwx 2 root       2048 Jun 27 12:19 lost+found
      1 drwxrwxrwx 9 root        144 Jul 11 13:27 m
      1 drwxr-xr-x 2 bin          64 Jun 27 11:10 sa
      1 drwxrwxrwx 2 root        112 Aug 11 11:04 tmp
      1 drwxrwxr-x 4 bin          80 Aug  4 14:45 u
    110 -r--r--r-- 2 bin      112580 Jun 27 12:22 unix
      1 drwxrwxr-x 9 root        144 Jul  7 16:28 usr
```

Remember that the contents of the root directory on your system may be somewhat different, especially if you have the multiuser operating system installed.

In this list of entries, the first number you see indicates the number of i-nodes taken up by the file or directory. An i-node describes a file and points to the actual physical disk blocks that make up the file. There is at least one i-node for each file and directory on the system.

A "**d**" in the column immediately to the right of the i-node number indicates that the entry is a **directory.** A hyphen (-) indicates that the entry is a **file.**

Special Modes

An **executable** file contains information that can be interpreted by the shell as command input. Such files are sometimes called "command files" on other systems, because they contain commands that can be executed by the operating system. You will notice that certain executable files and commands in **/bin** and **/etc** have a special mode known as **set user id.** Any program or file so designated has an "**s**" instead of an "**x**" in the owner rights slot. This means that when users execute these files, the user ID is set equal to that of the actual owner of the file. This allows users other than root to use commands that must create files in directories to which they normally would not have access rights. The **set user id** mode is important for such processes as **login, date, mail,** and **mkdir.**

Links

The concept of a **link** is really a system-level way of saving space by having several command names linked to a single copy of the actual binary source for that command. Commands that are linked use the same source but sometimes produce a slightly different result. Examples are **ll, lf, lr,** and **ls,** all of which list the contents of a directory, but in different ways.

You can see whether a command is linked by looking at the first number to the right of all the access and protection rights. For most entries, a 1 appears in this column. However, for files like **ll, lf, lr,** and **ls,** the number is greater than 1, indicating that there are several different versions of the same command.

Major System Directories

Here's what the major directories listed in the **root** (/) directory are used for.

Directory	Description
bin	Contains the binary source code for most of the FOR:PRO commands on the system. These commands are accessible to all users on the system.
dev	Contains all the special device files needed by the operating system. Each peripheral device attached to the system is defined as a file, and has an entry in the /dev directory. The contents of this directory vary depending on the devices you have attached to your particular system.
etc	Contains those FOR:PRO commands and special files which can only be accessed by manager or root. There are some exceptions to this which are discussed below. /etc also contains many crucial system files and tables which you can look at, but should not modify unless you know what you are doing.
f	The root node (directory) of the flexible disk file system. The /f directory is used by the system in order to transfer information between the file system on the flexible disk and a file system on the hard disk.
h	Top-level directory for the hard disk-based file system. Like /f, its function is to provide a way for the operating system to reference the contents of the hard disk by treaating it as a directory.
sa	Contains important files which are needed to reboot the system. These files are **/sa/boot** and **/sa/reconf**. The **"sa"** stands for stand alone because the programs in sa are meant to run without the aid of the operating system.
tmp	This is a very important directory used by many system processes to store temporary

information. It is a "scratch" area on the disk where information may be temporarily stored before it is written or copied to some other location. Don't ever delete this directory.

u This is the master directory for all user accounts. All the login accounts on the system which were created by the **newuser** process are listed in /u. (See Understand Your Fortune System for more information about the **newuser** process.)

unix The operating system itself -- don't touch this one!

usr Contains programs for users, including macros, user libraries, and other development utilities.

The /bin Directory

Many of the commands you will use on a routine basis are stored in the **/bin** directory. The word bin stands for binary. Most files in the **/bin** directory are binary files, so don't try to use **cat** to display them unless you use the **-v** option. In FOR:PRO, **/bin** contains the following commands on the single-user operating system set of disks:

cat	chmod	cp
date	dd	df
echo	ed	lf
ll	lr	ls
mkdir	mv	rmdir
pwd	rm	cmp
sh	sync	kill
du	expr	pr
login	passwd	stty
ps	sort	wall
test	true	
who	write	

The /etc Directory

Most of the entries in **/etc** are programs used by the system to do routine operations. For example, important system startup and shutdown programs are stored here. You must be logged in as manager or root to access some of these files and commands (for example, the **passwd** file). In this way, the files and commands that are crucial to the proper operation of the system are protected from unauthorized users.

The **chgrp** command, which changes the group ownership of directories or files, can only be used by the manager. You can't change the ownership of a file unless you are logged in as manager or root.

When You Need These Commands

You may find it necessary to use some of the commands in **/etc** and modify some of the files there if you need to reboot the system, or install a new operating system. For example, when starting up the system, you may experience an unsuccessful file check. You would then have to use the **/etc/fsck** program to manually check the file system. You might also want to know what's in the special files that reside in this directory in case you need to modify them or back them up.

The Files in /etc

The **/etc** directory contains the following special files:

File	Purpose
devtype	Lists all peripheral devices attached to the system. Specifies port number (tty number), baud rate, category and type of device, printer message port and optional location description.
disk	Contains configuration blocks for disk devices.
disktab	A list of all the disk drives available on potential Fortune systems.
fstab	The file system table which indicates the name of the device on which the file system resides.
group	Lists all the user groups defined for the system. By default, there are groups defined for **sys**, **bin**, **daemon**, and **users**. The first field is the group name, the second, the group password, the third, the group ID, and the last, a list of all the accounts that belong to that group.

motd Stands for message of the day. You can put in this file any message that you want all users to see when they log onto the system.

mtab A table of all currently mounted devices. Useful when you want to know whether a flexible disk is really mounted or not. If a disk is mounted, you might see something like this when you type **cat /etc/mtab**

```
/ffd02
```

This means a flexible disk is mounted on drive 02, with a root directory of /f.

passwd Contains a list of all the user accounts on the system, including default users like **root** and **manager.**

printcap Contains definitions for all supported printers on Fortune systems. These definitions are used by the printer control programs.

profile A general purpose profile file which is executed prior to your own **.profile** file, if you have one in your home directory. At this release, **/etc/profile** defines print spooler defaults for printing files via the global menu "Print a File" option. It also defines the default language as "English," LANGUAGE=EN, and sets Fortune terminals to XON/XOFF mode.

termcap The file used by FOR:PRO to interpret the information coming from your particular terminal. It contains data on nearly every terminal type supported by UNIX.

ttys Defines the baud rate, login setting, and tty number for each port available on the system.

ttytype A table of the device names and tty numbers that are connected to your system. There should be one entry here for each port listed in the **ttys** file.

The Commands in **etc**

The **/etc** directory also contains the following commands:

boot	chgrp
chlog	chown
config	disk
dskselect	format
fsck	getty
halt	init
login.help	mkconf
mkdevs	mkfs
mklost+found	mknod
mount	pstat
rc	rc.cold
rc.hd	rc.pass1
rc.pass1.0	rc.pass1.F1
rc.pass1.F2	rc.pass1.H
rc.pass2	rc.pass2.run
rdconf	reboot
setdt	setnswap
shutdown	surftest
uconf	umount
update	utmp

SINGLE-USER OPERATING SYSTEM CONTENTS

This section is an alphabetical summary of all the commands available on the single-user operating system. If you have either or both of the other command sets installed on your system, you will notice that this list doesn't show some of the commands you may be accustomed to using. The (**f**) indicates the item is a **text file**, a (**c**) indicates it is a **command**, (**sf**) indicates it is a **special system file**, (**sc**) indicates it is a **system administration** level command, and (**m**) indicates it is a **miscellaneous** routine or table.

basename (c)	boot (c)	bootcp (sc)
bootflop.conf (m)	capture (c)	cat (c)
cd (c)	chgrp (c)	chlog (c)
chmod (c)	chown (c)	cmp (c)

```
config (sf)          cp (c)               cp.AM (sf)
date (c)             dd (c)               devtype (f)
df (c)               disk (f)             diskconf (f)
disktab (f)          dskselect (sc)       dtinit (c)
du (c)               echo (c)             ed (c)
expr (c)             filsys (f)           find (c)
flop.conf (m)        format (sc)          fsck (sc)
fsck.AM (sf)         fstab (f)            getty (c)
genstring (c)        group (f)            halt (c)
init (sc)            kill (c)             lf (c)
ll (c)               login (c)            login.help (sf)
lpd (sc)             lpdun (sc)           lpf (sc)
lpdun (sc)           lpq (c)              lpr (c)
lprm (c)             lr (c)               ls (c)
menu (c)             mid (sc)             mkconf (sc)
mkdevs (c)           mkdir (c)            mkfs (c)
mklost+found (sc)    mknod (sc)           more (c)
mknod (sc)           more (c)             motd (f)
mount (sc)           mtab (f)             mv (c)
newgrp (c)           nice (c)             page (c)
passwd (c)           pr (c)               printstring (c)
ps (c)               profile (f)          pstat (sc)
pwac (c)             pwd (c)              rc (c)
rc.cold (sf)         rc.hd (sf)           rc.pass1 (sf)
rc.pass1.0 (sf)      rc.pass1.F1 (sf)     rc.pass1.F2 (sf)
rc.pass1.H (sf)      rc.pass2 (sf)        rc.real (sf)
rdconf (sc)          reboot (sc)          reconf (sc)
rm (c)               rmdir (c)            setdt (sc)
setnswap (sc)        sh (c)               shutdown (sc)
sleep (c)            sort (c)             stty (c)
surftest (c)         sync (sc)            tee (c)
termcap (f)          test (c)             trans (f)
true (c)             tty (c)              ttys (f)
ttytype (f)          uconf (sc)           umount (c)
update (sc)          utmp (f)             wall (c)
what (c)             whl (f)              who (c)
write (c)
```

Appendix A: The New Print Spooler

A new version of the print spooler is included with the 1.7 operating system. This print spooler resides in the directory **/usr/bin**, instead of in **/usr/ucb** as in the previous release. Several new utilities are associated with the print spooler. Whether you print from the menus or use **lpr** from FOR:PRO command level, you should read this information.

WHAT'S IN THIS APPENDIX

This appendix describes the following:

- The new version of **lpr** and its associated utilities
- The changes to **lpr** that affect print formatting options
- The impact of the new spooler on various software applications
- Related programs and files made obsolete by the release of FOR:PRO

THE NEW lpr

The new **lpr** command brings increased functions and some changes from the previous version, making it necessary for some users to change the way they use their printers. For example, software features are now available to do the following from the FOR:PRO command level, not just from the Fortune:Word menus:

- Temporarily stop and restart a print job so you can change the paper or ribbon.
- Change the pitch, lines per inch, and page size on a per print job basis, using **lpr** options.
- Run more than one printer on your system. In addition, you can logically number them so you can direct jobs to specific printers.

- Change the order in which print jobs will be printed by manipulating print queues.

- Control individual print job formats by writing customized shellscripts, by setting variables in user **.profile** files. Alternatively, the system manager which can set up printer defaults for everyone on the system. The latter is most useful when people do their printing from the System Utilities menu.

lpr and Related Commands

The components of the print spooler interface include:

- **lpr** Sends files to a designated printer with optional formatting.

- **lprm** Removes one or more files from a designated print queue.

- **lpq** Displays the status of one or more print queues.

- **lpmv** Changes the order in which jobs for a given printer will be printed.

- **lpdun** Stops and restarts a print job for printer servicing, or adding/changing paper or ribbon. Also sets new defaults for print page format.

- **pr** Displays a file on the screen, placing page numbers at the top of each unit of 66 lines. Output from **pr** can be piped to **lpr**, if page numbers are desired on the printout.

Per Print Session Formatting

New features in the current **lpr** enable you to control the format of a print job via software on a per print job basis. This makes it possible for individual users to print documents according to their own specifications without interfering with one another. It is no longer possible to set print format parameters via hardware switches and have the **lpr** software recognize them. Users who

previously controlled settings like pitch, lines per inch, page size, and baud rate by physically setting switches on the printer should note that the FOR:PRO spooler will not look at these hardware settings.

Instead, such changes in pitch, lines per inch, and so forth, must be done by using options of the **lpr** command, or by modifying the system-supplied defaults for print jobs. For users printing from the System Utilities menu, system-supplied defaults can be found in the file **/etc/profile**, in a variable called LPRFLAGS. This is explained below.

Users of **lpr** from command level will find that print job defaults are set by **lpdun** and **lpr**, and can be changed on a per user and per print job basis by issuing the appropriate options on the command line.

Print Job Defaults in LPRFLAGS

A file called **/etc/profile** is supplied with the FOR:PRO operating system. Currently, this file is only executed by the menu shell and is therefore effective only for menu system users. However, users can include its settings in their **.profile** files. It can be used to set up defaults for banners, pitch, lines per inch, page size, and line width for all print jobs queued through the print option of the System Utilities menu. When at the Bourne shell level, users who log into the menu shell can still override these settings on a per print job basis by using the proper **lpr** and **lpdun** options.

As delivered, LPRFLAGS in **/etc/profile** contains the following default printer setting for global menu users:

```
-h
```

This means that all printouts will have no header (banner) page, unless otherwise specified. This is a change from the previous system in which banners were the default. The change was made specifically to prevent paper waste in single-user systems where there is little need to identify printouts by user name. To change this default, you must edit the file and remove **-h** from the LPRFLAGS line. Use **ed** (or **vi**, if you have Development Utilities) to do this. You can also delete the LPRFLAGS line entirely if you like.

Users of the "Print" option on the System Utilities menu will get their print job defaults from the LPRFLAGS variable in **/etc/profile.** This is true except for users who have **.profile** files and who also use the menus. LPRFLAGS is ignored by **lpr** when used from the Bourne shell level. Only the menu shell uses LPRFLAGS with **lpr.** However, any other **lpr**-related setting in a user's **.profile** file will override the settings in **/etc/profile,** since the user's **.profile** file is executed after **/etc/profile.**

Defaults and How to Override Them

The following table shows how printer settings correspond to options of **lpr** and **lpdun.** To set these options for people using the global menus, include the proper option in LPRFLAGS in **/etc/profile.**

Setting	lpdun/lpr Option	Default Value
Line Length	**-tL**nn **(lpr)** **-c**nn **(lpdun)**	132
Lines Per Page (Form Length)	**-F**nn **(lpr)** **-l**nn **(lpdun)**	66
Left Margin	**+L**nn **(lpr)**	0
Pitch	**+P**nn **(lpr)**	10
Lines Per Inch (Vertical Pitch)	**+V**nn **(lpr)**	6

> NOTE
>
> Note that the - options must appear before the + options on a command line, and in LPRFLAGS. This is because any - options that appear after + options are treated as filenames.

You can check the **lpdun** defaults by typing

lpdun -z

Changing the Vertical Pitch

If you use the **+V** option to change vertical pitch, it will slightly adjust the size of a **page.** You will have to adjust the top-of-form after the job with the changed pitch size is completed. Note also that 10 pitch is really 9.6, so there are really 9.6 lines per inch instead of 10. This means the number of lines per page is really 105.6, not 105 or 106. You will notice a slight increase in page size on long print jobs when printing with +V10.

Changing Line Length

The default line length is 132 characters. To change it on a per print job basis, use

lpr -tLnn filename

The **-t** option guarantees the line will be truncated at nn characters. To change the default line length on a global basis, use the **-c** option of **lpdun**

lpdun -c nn

Individual Settings for Print Jobs

Print job settings can be altered on a per print job basis by each individual user. System defaults can be overriden by shellscripts that print files, or by customizing user **.profile** files.

For example, the following one-line shellscript prints files with a horizontal pitch of 12 characters per inch, and a vertical pitch of 8 lines per inch:

lpr +P12 +V8 $1

By making this shellscript executable (**chmod 755** filename), you could print files according to these settings by simply typing the name of the shellscript followed by the name of the file to be printed. For example, if the shellscript containing the above **lpr** line were named **mpr**, you could print files with it using this command line:

mpr filename

You could also put a line similar to this one in your **.profile** file:

```
mpr='lpr -b Greg +L5 $1'
```

Every time you type

mpr filename

from the Bourne shell, your printouts will have a banner name, like Greg in this case, and an adjusted left margin. Notice that the variable name used here is **mpr** and not **lpr**. It is advised that you do not use **lpr** as a name for shellscripts or aliases, as this can only cause confusion. By using other names you can make multiple **lpr** command lines that perform different functions.

> NOTE
>
> For users with login shells set to **/bin/menu**, your **.profile** file (if you have one) is executed after the **/etc/profile** file is read. Therefore, settings in your **.profile** file will override those in **/etc/profile**. Since **/etc/profile** is only read by the menu shell, you can incorporate its contents in your login environment with this command
>
> **. /etc/profile**
>
> as the first line of your **.profile** file.

Multiple Printers and Print Queues

The FOR:PRO spooler and Device Connection Menus allow you to define and make use of multiple printers per system. The **lpq**, **lpmv**, and **lprm** utilities allow you to look at, manipulate entries in, and remove entries from multiple print queues.

For Fortune:Word Users

In order to use Fortune:Word 1.0 at Release 1.7, it must be upgraded using the 1.7 Upgrade disk. Users who are familiar with version 1.0 of Fortune:Word will recognize that much of the printer

functionality available to them in Fortune:Word is now available at the FOR:PRO level via the new **lpr**. In Fortune:Word, the printer control menu can be used to set defaults on a per print job basis by changing the prototype file used for a particular print job.

For Multiplan Users

In order to use Multiplan on 1.7, it must be upgraded using the 1.7 Upgrade disk. The program **mp.printer**, which was formerly used to set printer defaults for Multiplan, has been replaced by a program called **mpinit**. This program defines the page size, pitch, number of lines per inch, and so forth for Multiplan print jobs. It is run automatically when you upgrade the Multiplan master disk using the 1.7 Upgrade disk, or when you use the Upgrade disk to restore your system backup. **mpinit** can be run by the manager account whenever the print job characteristics for Multiplan print jobs need to be changed. It is located in the directory **usr/bin**. In future releases of Multiplan, this program will be replaced by a printer menu similar to that of Fortune:Word.

For BAS Applications Users

Users of BAS applications should note that **/dev/lp** is linked by default to **/dev/tty01**. If you have multiple printers on your system and want to print BAS application files on a printer attached to a port other than tty01, make changes to the IPL files as discussed in Technical Tip 1.1, Multiple Printers within BASIC. (See also Appendix E of the Business BASIC manual.)

> NOTE
>
> Since BAS does not use **lpr**, if you use the same printer for both BAS and FOR:PRO-level printing, output from both may appear intermixed on the same printer. In this case, use **lpdun** to stop the **lpr**-level print job, then reprint the pages after the BAS job is finished. The BAS job will have to be restarted if clean output is desired. The same problem may occur if two or more people are printing BAS jobs at the same time.

Related Print Spooler Files

The following files are also related to the new print spooler on the single-user operating system:

File	Use
/etc/profile	For users of global menu system.
/etc/devtype	Defines all devices, including printers.
/dev/lp	Used by BAS applications only.
/etc/*.whl	* stands for any supported printer type. Files of this nature are used for alternate character sets.
/etc/printcap	Defines printer control characteristics for all supported printers.

Controlling Print Spooler Priority

As delivered at 1.7, the print spooler is assigned the standard priority (0). The default number of pages saved is 2. To change either of these default settings, use these options of **lpdun:**

-mnn	Change number of pages to save
-Pnn	Change priority of print spooler

In the **-m** option, the nn argument specifies how many pages are to be kept in memory for reprinting if it is necessary to restart a print job several pages back from the one at which printing halted. The **-P** option takes an nn argument that can have a value of -20 to +20. The highest priority setting is -20, which devotes maximum system resources to the print spooler. The lowest priority setting is +20, which will improve interactive response, but slow printing down. (On very heavily loaded systems, low priority jobs may never run, so use caution when lowering priorities.)

Obsolete_Items

If you used Fortune:Word, Multiplan, or other applications, the following programs are now obsolete:

```
/etc/ttype
/usr/bin/def.printer
/usr/bin/mp.printer
/usr/bin/lpint
/usr/lib/lpfc
/usr/bin/on.printer
/usr/bin/off.printer
/usr/ucb/lpr  (*)
/usr/ucb/lprm  (*)
/usr/ucb/lpq  (*)
```

Items marked with an asterisk (*) have been moved to **/usr/bin** as of this FOR:PRO release.

SINGLE-USER OPERATING SYSTEM MAN PAGES 2

Part 2 contains reference documentation for the FOR:PRO single-user operating system. It is provided in the form of MAN pages (manual pages) in the same format as the Bell Laboratories' UNIX Programmer's Manual. The MAN pages provide Fortune-specific reference material for all commands and system level files available on the single-user operating system set of disks.

ORGANIZATION OF THE MAN PAGES

UNIX MAN pages are traditionally divided into eight sections. This subset of the FOR:PRO MAN pages contains entries from Sections 1, 4, 5, and 8 and are described below. All sections appear in the FOR:PRO Programmer's Manual, which is included with the Development Utilities software. Refer to the header on the page for correlation to UNIX MAN pages.

Section		Description
1	**Commands** (Chapter 1 in this manual.)	References commands invoked directly from the shell, including the most commonly used commands.
4	**Special Files** (Chapter 2 in this manual.)	Discusses the special device files contained in the directory /dev. These files describe peripheral devices connected to the Fortune system.
5	**File Formats and Conventions** (Chapter 3 in this manual.)	Describes important system files used by a number of system programs.
8	**System Administration Commands** (Chapter 4 in this manual.)	Explains commands used for system administration.

THE MAN PAGE FORMAT

All MAN pages follow a specific format. Most MAN page entries have at least these three sections:

NAME Gives the command name and a description of its use.

SYNOPSIS Presents the syntax of the particular command, all its options, and other possible arguments.

DESCRIPTION Gives a detailed description of the command and how it interprets its arguments.

> NOTE
>
> In the **SYNOPSIS** section of certain commands, more than one command option may be listed. For example, the **ls** command has many options, as this syntax indicates:
>
> **ls [-abcCdFgilmgrRstuxl]**
>
> Each character within the brackets is an option of the **ls** command. You may use one, many, or none of the options on the same command line. However, make sure the first option you type is immediately preceded by a hyphen, as in
>
> **ls -l**

Some entries have additional sections that include the following:

OPTIONS Command options are discussed in detail here.

EXAMPLES Gives one or more examples of the command.

FILES Names any files built into the command.

SEE ALSO Refers you to commands related to the one described.

DIAGNOSTICS Discusses diagnostics you may receive when using the command.

WARNINGS Indicates problem areas and gives precautions.

LIMITATIONS Lists known bugs and design limitations.

NAME

basename - strip filename affixes

SYNOPSIS

basename *string* [suffix]

DESCRIPTION

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

This shell procedure invoked with the argument */usr/src/cmd/cat.c* compiles the named file and moves the output to *cat* in the current directory: cc $1 mv a.out `basename $1 .c`

SEE ALSO

sh(1)

NAME

capture - allows automatic capture of certain key system information

SYNOPSIS

capture [fps]

DESCRIPTION

Capture is a shell-script which allows automatic capture of certain key system information to a diskette, a listing, or interactively to the screen. This information is intended to assist in problem reporting and trapping for software products.

Capture gathers 'what' strings, directory listings, the contents of certain files, and other system information. It then dumps this data to a diskette or a listing which may be included with a problem report to provide information regarding the environment in which the problem occurred. Alternatively, *capture* has an interactive mode in which the user can select items of information items of information to be displayed on the screen, one at a time. Progress and information messages are sent to the screen.

OPTIONS

f sends the information to a flexible diskette. The user is asked whether to continue with the process ('y' to continue the process, 'n' to end the process). The diskette is formatted and the information from *capture* is written. The user is notified upon completion.

p sends the information to a printer.

s displays the information on the screen. Build a two digit code of (col)(row) from the displayed chart.

Example:

"2g". Reply <space> to "<<STOPPED>>".
Use "q" to quit.

FILES

/usr/bin/capture
shell script

/usr/lib/capture_menu
contains the menu for the screen option

/usr/lib/capture_inst
displays instructions

DIAGNOSTICS

If no option is specified when capture is invoked, the following is displayed:

```
Usage: capture floppy   or
       capture printer  or
       capture screen
```

NAME

cat - concatenate and print

SYNOPSIS

cat [-n] [-s] [-u] [-v] *file ...*

DESCRIPTION

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

If no input file is given, or if the argument '-' is encountered, *cat* reads from the standard input file. Output is buffered in 1024-byte blocks unless the standard output is a terminal, in which case it is line buffered.

OPTIONS

-n Causes the output lines to be numbered sequentially from 1. Giving **-b** with **-n** causes numbers to be omitted from blank lines.

-s Causes the output to be single spaced.

-u Causes the output to be completely unbuffered.

-v Causes non-printing characters to be printed in a visible way. Control characters print like ^X for control-x; the delete character (octal 0177) prints as ^?. Non-ascii characters (with the high bit set) are printed as M- (for meta) followed by the character of the low 7 bits. A **-e** option can be given with **-v** and causes the ends of lines to be followed by the character '$'; the **-t** option with **-v** causes tabs to be printed as ^I.

EXAMPLES

cat file

prints the file

cat file1 file2 > file3

concatenates the first two files and places the result in the third

SEE ALSO

cp(1), ex(1), more(1), pr(1), tail(1).

LIMITATIONS

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

NAME

cd - change working directory

SYNOPSIS

cd *directory*

DESCRIPTION

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

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command. It is therefore recognized and executed by the shells. In *csh*(1) you can specify a list of directories in which *directory* is to be sought as a subdirectory if it is not a subdirectory of the current directory; see the description of the *cdpath* variable in *csh*(1).

SEE ALSO

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

NAME

chlog - change UNIX error logging format

SYNOPSIS

chlog [unix] [default] [debug *hexnumber*] [*decimal decimal*]

DESCRIPTION

Chlog changes the style in which UNIX prints internal error messages. *Chlog unix* switches to the standard unix style of error message, while *chlog default* prints them in the default style of error numbers. *chlog debug hexnumber* controls how hardware fault messages are printed. Chlog uses the *elog*(2) system call. *Chlog decimal decimal* does a direct *elog*() system call.

EXAMPLES

Chlog unix performs elog(LOGTYPE, LUNIX). *Chlog default* performs elog(LOGTYPE, LDEFLT), *Chlog debug 50f3* performs elog(DBGLVL, 0x50f3). *Chlog 4 40* performs elog(kupdate,40), which instructs the kernel to do a sync(2) every 40 seconds.

```
/*
 * Used in the elog() system call to determine error logging type
 */
#define LOGTYPE     0       /* set the 'logtype' variable          */
#define DBGLVL      1       /* set the 'debug' variable            */
#define KUPDATE     4       /* enable kernel update()s             */
#define KNOUPDATE 5         /* disable update()s from kernel       */

#define LDEFLT      0       /* default error logging               */
#define LUNIX       1       /* make it more like normal
                               UNIX logging                        */
```

The DBGLVL bits are defined as follows (from sys/debug.h) :

```
#define D_MAIN      0x0001 /* starting icode & schduler msgs */
#define D_TRAP      0x0002 /* system calls, trap messages    */
#define D_UREG      0x0004 /* remapping calls to ureg, estabur*/
#define D_SYSCALL 0x0400 /* system call info                 */
#define D_STACK     0x0800 /* Stack trace before death       */
#define D_FLOPPY    0x1000 /* Floppy disk mfg debugs         */
#define D_FDPRINT 0x2000 /* Floppy disk verbose printf()s    */
#define D_WDISK     0x4000 /* Winchester disk printfs()s     */
```

FILES

/usr/include/sys/err.h

/usr/include/sys/types.h

SEE ALSO

elog(2)

NAME

chmod - change mode

SYNOPSIS

chmod *mode file ...*

DESCRIPTION

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

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, 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

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*] ...

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

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

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

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

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

EXAMPLES

To deny write permission to others,

```
chmod o-w file
```

To make a file executable:

```
chmod +x file
```

SEE ALSO

ls(1), chmod(2), stat(2), umask(2), chown(8).

NAME

cmp - compare two files

SYNOPSIS

cmp [-l] [-s] *file1 file2*

DESCRIPTION

The two files are compared. (If *file1* is '-', the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

OPTIONS

-l Print the byte number (decimal) and the differing bytes (octal) for each difference. If this option is not given, only the first difference is listed.

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

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

cp - copy or link files and directory trees

SYNOPSIS

cp [-abBdilnNoprRsStTuvVxX] [backup options] [-] *file1 file2*
cp [-abBdilnNoprRsStTuvVxX] [-] *file ... directory*

DESCRIPTION

File1 is copied onto *file2*. In the second form, one or more *files* are copied into *directory*, which must exist, and the last element of the file pathname is used as the name of the new file in *directory*. Links between source files will be preserved in the destination wherever possible.

If there is a single source argument and it is a special file, it is opened and read as if it were a regular file. In the multiple-source-argument or recursive case, special files are ignored.

If a destination file exists, its owner and mode are unchanged, and the source is copied onto it. Otherwise it is created with the ownership of the invoker and the mode of the source. The setuid and setgid bits will each be set only if the new file is owned by the same user id and group id as the source, respectively. If the source is not a regular file and the destination is new, then *cp* creates the new file with permissions rw-rw-rw- subject to the *umask*.

Cp attempts to read 20K bytes at a time, and writes out the same amount it reads each time. Thus it is usable with tapes or other devices with large or irregular block sizes.

Cp will not attempt to copy a file onto itself.

OPTIONS

-a Ask on all files before copying. If **-r** is selected, sets **-d**. Answer is interpreted as in **-i**.

-b Copy a file only when the destination doesn't already exist or its inode modification time (st_ctime) is older than the source.

-B Backup. Automatically makes file systems, handles error recovery and works with any other *cp* option. **-B** requires two backup options:

1) special file to use (mount)
2) size of file system to make

There is no destination file specified on the command line as it is assumed the same name is used on the backup media. Backup calculates the sizes of every file and tells you how many volumes are needed to store the files specified. Examples:

cp -B /dev/fd02 770 /bin/cat /bin/ls

This will back up only the files /bin/cat and /bin/ls.

```
cp -Br /dev/fd02 770 /bin /usr
```

This will backup all directories and files under /bin and /usr.

```
cp -Broust /dev/fd02 770 /
```

This will backup the entire filesystem.

-d Ask on all but regular files before copying. If *cp* is copying recursively, and you decline to examine a directory, that directory and recursively all its contents are skipped. Answer is interpreted as in **-i**.

-i Prompts the user with the name of the file whenever the copy causes an existing file to be overwritten. An answer of 'y' or 'Y' will cause *cp* to continue. Any other answer prevents it from overwriting the file.

-l Link instead of copying if source and destination are on the same file system. If a link can be made and the destination already exists, *cp* unlinks it before making the new link. Note that this is different from the behavior of ln(1). If you want a link to fail when the destination exists, as it would if you were using ln(1), then use **-n** along with **-l**.

-n Copy a file only when the destination does not already exist.

-N No copy operation. This allows you to do a **-o** and/or **-t**, to destination file(s) which already exist. No file copies take place, and no directories are made when this option is in effect. Cannot be used with **-l**, **-n**, or **-u**.

-o Set the group and user id ownership of the destination to that of the source. Only super-user can do this.

-p Use full pathnames. Form each destination name by appending the entire source path to the destination path. This is especially useful for copying selected files from one tree to another. Example:

```
cp -p d1/d2/f1 x
```

will make x/d1 and x/d1/d2 if necessary and copy d1/d2/f1 to x/d1/d2/f1.

-r Recursive. Copy each source directory and its entire subtree. Make new subdirectories in the destination directory as necessary. Without this option, directories are skipped.

-R Restore. Asks the user to insert diskettes and ensures they are all from the same backup. **-R** requires one backup option which is the special file to mount. Restore first discovers where each file is and asks you to insert each volume as needed. Examples:

```
cp -R /dev/fd02 bin/cat bin/ls /bin
```

This restores the files "cat" and "ls" to the directory bin.

```
cp -rpR /dev/fd02 bin usr /
```

This restores everything in /bin and /usr.

```
cp -Rroust /dev/fd02 /. /
```

This restores an entire backed-up filesystem.

-s Do special files. If **-l** is not set, or it is set and *cp* can't link, and you are super-user, do a mknod(2) so that the destination file is a device just like the source.

-S Replace an argument of "=" with a list of newline-separated files from the standard input. If this option is in effect, it is assumed that you are copying multiple files to a directory even if there is only one file in the list from the standard input, and the destination argument must be on the command line. This option is incompatible with -B, -R or -i.

-t Set the file modification and access times of the destination to that of the source. This only works if you are super-user or you own the destination file. To be sure that you will own destination files, use the **-u** option.

-T Same as **-t** plus the access time of the source is unchanged by the copy.

-u If the destination is not a directory and already exists, unlink it before copying. This is useful when the destination file is multiply-linked and you don't want to affect all the links. It is also useful when you want all the destination files to be owned by you and to have the same mode as the source.

-v Verbose, partial. Print a line on the standard output noting each copy operation which is skipped because of conditions specified by the **-n** or **-b** options.

-V Verbose, full. Print a line on the standard output noting each copy operation, whether it is to be done or skipped, each source directory that is examined, and each destination directory that is created.

-x Copy a file only when the destination already exists. Do not make any directories.

-X When used with the backup option, **-B**, verifies that the backup media is readable after each disk is written. Does not check file system integrity or do any data verification, however. This option is ignored when not used with **-B**.

The null option - indicates that all the arguments following it are to be treated as file names. This allows *cp* to work with files whose names start with a minus.

EXAMPLES

For these examples, assume the following directories and files:

d1/f1, d1/d2/f2, and d3

where d1, etc. are directories, and f1, etc. are files. Then

cp d1/f1 d3

would make d3/f1, and

cp -r d1 d3

would make d3/d1/f1 and d3/d1/d2/f2, and

cp -r d1/. d3

would make d3/f1 and d3/d2/f2.

You can effect a recursive move across file systems by doing a 'cp -r' followed by a 'rm -r'. The mv(1) command is best for moving a subtree within a file system.

SEE ALSO

cat(1), ln(1), mv(1), rm(1).

DIAGNOSTICS

Copy is normally silent about its operation. If an error occurs, a message will be printed to the standard error output. If a file is not copied for some other reason, this will be noted on the standard output unless this was because it did not meet a condition specified by **-n** or **-b**. For more verbosity, use **-v** or **-V**.

Cp does not stop prematurely on errors. Exit status is '0' if all files are copied OK. If any file or directory was skipped for some reason, the exit status is '1' unless any file or directory was supposed to be copied, but couldn't because of an error, in which case the exit status is '2'. Errors may result in partially-written destination files. Other exit statuses are possible for "fatal" errors.

LIMITATIONS

The ownership and times of directories created under the **-p** option cannot be set to match the source directories. In this case, the owner is set to the current uid, the time to the current times, and the mode is set to 'rwxrwxrwx'.

Links may be 'preserved' in the destination to a file which did not copy successfully.

NAME

date - print and set the date

SYNOPSIS

date [*yymmddhhmm* [*.ss*]]

DESCRIPTION

If no argument is given, the current date and time are printed. If an argument is given, the current date is set. *yy* is the last two digits of the year; 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; *.ss* is optional and is the seconds. The year, month and day may be omitted, the current values being the defaults. (Only the super-user can change the default.) The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time. *Uconf*(2) can be used to set the timezone.

EXAMPLES

date 8610080045

sets the date to Oct 8,1986 12:45 AM

date 1345

sets time to 1:45 PM and leaves the date unchanged.

FILES

/usr/adm/wtmp
to record time-setting

SEE ALSO

uconf(8), utmp(5).

DIAGNOSTICS

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

NAME

dd - convert and copy a file

SYNOPSIS

dd [*option=value*] ...

DESCRIPTION

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

OPTIONS

option	*values*
bs=*n*	set both input and output block size, superseding *ibs* and *obs;* also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
block	convert variable length records to fixed length
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lowercase
noerror	do not stop processing on an error
swab	swap every pair of bytes
sync	pad every input record to *ibs*
ucase	map alphabetics to uppercase
unblock	convert fixed length records to variable length
... , ...	several comma-separated conversions
files=*n*	skip *n* input files before starting copy
ibs=*n*	input block size *n* bytes (default 512)
if=	input file name; standard input is default
iseek=*n*	seek over n input records before copying
iskip=*n*	same as skip
nocreat	don't create the output file before writing to it
obs=*n*	output block size (default 512)
of=	output file name; standard output is default
oseek=*n*	same as seek.
oskip=*n*	skip (read over) n output records
quiet	Don't print summary of input and output records when done
seek=*n*	seek *n* records from beginning of output file before copying
skip=*n*	skip *n* input records before starting copy

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

product.

Cbs is used only if *ascii, unblock, ebcdic, ibm,* or *block* conversion is specified. In the first two cases, *cbs* characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and new-line added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size *cbs*.

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

EXAMPLES

To copy one 6-head (96 512byte blocks/cyl) hard disk to another,

```
dd if=/dev/rhd02 of=/dev/rhd12 bs=48k
```

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

To back up a (small) directory to a floppy (10 1K blocks/cyl):

```
tar cfv - dir | dd of=/dev/rfd02 obs=10k
```

SEE ALSO

cp(1), tr(1).

DIAGNOSTICS

f+p records in(out): numbers of full and partial records read(written)

LIMITATIONS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The 'ibm' conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions.

NAME

df - disk free

SYNOPSIS

df [-f] [-i] [-l] [*filesystem...*] [*file...*]

DESCRIPTION

Df prints out the number of free blocks available on the specified *filesystem*, e.g. "/dev/hd0z", or on the filesystem in which the specified *file*, e.g. "$HOME", is contained. If no file system is specified, the free space on all of the normally mounted file systems is printed.

The reported numbers are in file system block units; currently each filesystem block is 1024 bytes long. *du*(1) or *ls*(1) with the **-s** option.

OPTIONS

-f gives the report in Fortune friendly format. Initial **-f** should be used exclusively, as it overides all other options.

-i Report also the number of inodes which are used and free.

-l Also examines the free list, double checking that the summary number in the filesystem superblock is correct.

EXAMPLES

```
% df
Filesystem  Mounted on  kbytes   used   free  % used
/dev/hd02   /            5835   2706   3129     46%
% df -l
Filesystem  Mounted on  kbytes   used   free  hardway  % used
/dev/hd02   /            5835   2706   3129    3129      46%
```

FILES

/etc/fstab
list of normally mounted filesystems

SEE ALSO

fstab(5), icheck(8).

LIMITATIONS

It should print some warning message when the '-l' option discovers a messed-up file system.

NAME

dtinit - initialize /etc/devtype

SYNOPSIS

/m/sysman/dtinit [**-b** *baud*] [**-c** *class*] [**-m** #] [**-n** *device*] [**-e**] [**-d**] [**-s** *device*] [**-t** *type*] [**-x** *name*] [-T] [**-M**]

DESCRIPTION

Dtinit is used to update the device type file /etc/devtype. If an entry exists for the device, any new information specified in the command is added to the entry. If no entry exists, one is created.

OPTIONS

-b *baud*
baud rate for the entry. *baud* can be any legal baud rate or the one character specified as defined in Berkeley getty.

-c *class*
class of the device (e.g. 'P' for printer, 'C' for comm line).

-m #
number of the device in its class (e.g. P2 for the second printer).

-n *device*
UNIX special file name (i.e. the name in the device table /dev). This is used for transferring data. (see -s option).

-d disable this line. This sets a status bit to 0 indicating that the line is disabled.

-e enable this line. This sets a status bit to 1 indicating that the line is enabled.

-s *device*
service notification port. UNIX special file name (i.e. the name in the device table /dev). Bells (control g) are sent to this port when operator intervention is needed.

-t *type*
type of device. This is most likely a string which identifies an entry in /etc/printcap or /etc/termcap.

-x *name*
external name. This is any string to identify the device (e.g. Dot matrix printer near the water cooler).

-T Prints out a table of the entries in /etc/devtype.

-M Updates the /etc/ttys and /etc/ttytype files.

There is a one-to-one correspondence between options and fields in a devtype entry. The minimum arguments necessary to uniquely identify an entry is either a name or a class and number.

FILES

/etc/devtype
device configuration file

SEE ALSO
devtype(5), ttys(5), ttytype(5).

NAME

du - summarize disk usage

SYNOPSIS

du [-a] [-s] [*name...*]

DESCRIPTION

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

A file having two links to it is only counted once.

OPTIONS

-a causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

-s causes only the grand total to be given.

SEE ALSO

df(1), filsys(5).

LIMITATIONS

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

If there are too many distinct linked files, *du* counts the excess files multiply.

du does not count indirect blocks (see *filsys*(5)).

NAME

echo - echo arguments

SYNOPSIS

echo [-n] [*arg*] ...

DESCRIPTION

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

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, when running the Bourne shell, do 'echo ... 1>&2'.

OPTIONS

-n No newline is added to the output.

NAME

ed - text editor

SYNOPSIS

ed [-x] [-] [*name***]**

DESCRIPTION

Ed is the standard text editor.

If a *name* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed's* buffer so that it can be edited.

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

Commands to *ed* have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

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

Ed supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus \[. and sometimes ^*$.
2. A . matches any character.
3. A \ followed by any character except a digit or (\) matches that character.
4. A nonempty string *s* bracketed [*s*] (or [^*s*]) matches any character in (or not in) *s*. In *s*, \ has no special meaning, and] may only appear as the first letter. A substring *a-b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of form 1-4 followed by * matches a sequence of 0 or more matches of the regular expression.

6. A regular expression, *x*, of form 1-8, bracketed \(*x*\) matches what *x* matches.

7. A \ followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *n*th \(matched.

8. A regular expression of form 1-8, *x*, followed by a regular expression of form 1-7, *y* matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.

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

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

11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

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

1. The character '.' addresses the current line.

2. The character '$' addresses the last line of the buffer.

3. A decimal number *n* addresses the *n*-th line of the buffer.

4. ''*x*' addresses the line marked with the name *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.

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

6. A regular expression enclosed in queries '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.

7. An address followed by a plus sign '+' or a minus sign '-' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign can be omitted.

8. If an address begins with '+' or '-' the addition or subtraction is taken with respect to the current line; e.g. '-5' is understood to mean '.-5'.

9. If an address ends with '+' or '-', then 1 is added (resp. subtracted). As a consequence of this rule and rule 8, the address '-' refers to the line before the current line. Moreover, trailing '+' and '-' characters have cumulative effect, so '--' refers to the current line less 2.

10. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is equivalent to '-'.

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

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

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

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

(.)a
<text>

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

(.,.)c
<text>

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(.,.)d

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

e filename

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

E filename

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

f filename

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

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

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

(.)i
<text>

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

(.,.+1)j

This command joins the addressed lines into a single line; intermediate newlines simply disappear. '.' is left at the resulting line.

(.)k*x*

The mark command marks the addressed line with name *x*, which must be a lowercase letter. The address form ''*x*' then addresses this line.

(.,.)l

The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The *l* command can be placed on the same line after any non-i/o command.

(.,.)m*a*

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

(.,.)p

The print command prints the addressed lines. '.' is left at the last line printed. The *p* command can be placed on the same line after any non-i/o command.

(.,.)P

This command is a synonym for *p*.

q

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

Q

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

($)r filename

The read command reads in the given file after the addressed line. If no filename is given, the remembered filename, if any, is used (see *e* and *f* commands). The filename is remembered if there was no remembered filename already. Address '0' is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '.' is left at the last line read in from the file.

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

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

An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of '&' in this context can be suppressed by preceding it by '\'. The characters '*n*' where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

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

(.,.)t*a*

This command acts just like the *m* command, except that a copy of the addressed lines is placed after address *a* (which can be 0). '.' is left on the last line of the copy.

u

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

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

This command is the same as the global command *g* except that the command list is executed *g* with '.' initially set to every line *except* those matching the regular expression.

(1, $)w filename

The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writable by everyone). The filename is remembered if there was no remembered filename already. If no filename is given, the remembered filename, if any, is used (see *e* and *f* commands). '.' is unchanged. If the command is successful, the number of characters written is printed.

(1,$)W filename

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

x

A key string is demanded from the standard input. Later *r*, *e* and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption. This option does not work unless you have the Development Utilities Set installed on the Fortune 32:16.

(.)=

The line number of the addressed line is typed. '.' is unchanged by this command.

!<shell command>

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

(.+1)<newline>

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

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

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

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

OPTIONS

-x An *x* command is simulated first to handle an encrypted file.

- Suppresses the printing of character counts by *e*, *r*, and *w* commands.

FILES

/tmp/e*

ed.hup
work is saved here if terminal hangs up

SEE ALSO

sed(1), crypt(1).

B. W. Kernighan, *A Tutorial Introduction to the ED Text Editor*

B. W. Kernighan, *Advanced editing on UNIX*

DIAGNOSTICS

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

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

LIMITATIONS

The *l* command mishandles DEL.
A *!* command cannot be subject to a *g* command.

Because 0 is an illegal address for a *w* command, it is not possible to create an empty file with *ed*.

NAME

expr - evaluate arguments as an expression

SYNOPSIS

expr *arg ...*

DESCRIPTION

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

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

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

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

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

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

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

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

(*expr*)
: parentheses for grouping.

EXAMPLES

To add 1 to the Shell variable *a*:

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

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

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

Note the quoted Shell metacharacters.

SEE ALSO

sh(1), test(1).

DIAGNOSTICS

Expr returns the following exit codes:

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

NAME

find - find files

SYNOPSIS

find *pathname-list expression*

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) 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*.

-atime n
: True if the file has been accessed in *n* days.

-exec command
: True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument '{}' is replaced by the current pathname.

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

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

-links n
: True if the file has *n* links.

-mtime n
: True if the file has been modified in *n* days.

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

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

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

-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, more flag bits (017777, see *stat*(2)) become significant and the flags are compared: *(flags&onum)===onum*.

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

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

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

-user uname
True if the file belongs to the user *uname* (login name or numeric user ID).

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

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).
2) The negation of a primary ('!' is the unary *not* operator).
3) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
4) Alternation of primaries ('**-o**' is the *or* operator).

EXAMPLES

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

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

FILES

/etc/passwd

/etc/group

SEE ALSO

sh(1), test(1), filsys(5).

NAME

kill - terminate a process with extreme prejudice

SYNOPSIS

kill [*-signo*] processid ...

DESCRIPTION

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

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

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

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

SEE ALSO

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

NAME

login - sign on

SYNOPSIS

login [*username*]

DESCRIPTION

The *login* command is used when a user initially signs on, or it can be used at any time to change from one user to another. The latter case is the one summarized above and described here. See *Understand Your Fortune System* for initial login.

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

After a successful login, accounting files are updated and the user is informed of the existence of mail, and the message of the day is printed. The user is also informed of the time he last logged in (unless he has a *.hushlogin* file in his home directory - this is mostly used to make life easier for non-human users, such as *uucp*).

Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(1)) according to specifications found in a password file. Argument 0 of the command interpreter is "-sh", or more generally the name of the command interpreter with a leading dash ("-") prepended.

Login also initializes the environment *environ*(5) with information specifying home directory, command interpreter, terminal type (if available) and user name.

If the file /etc/nologin exists *login* prints its contents on the user's terminal and exits.

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

FILES

/etc/utmp
: accounting

/usr/adm/wtmp
: accounting

/usr/spool/mail/*
: mail

/etc/motd
: message-of-the-day

/etc/passwd
password file

/etc/nologin
stops logins

.hushlogin
makes login quieter

/etc/securetty
lists ttys that root may log in on

SEE ALSO

mail(1), newgrp(1), passwd(1), environ(5), passwd(5), getty(8), init(8), shutdown(8).

DIAGNOSTICS

"Login incorrect," if the name or the password is bad.

"No Shell", "cannot open password file", "no directory": consult your dealer.

lpr, print - line printer spooler

SYNOPSIS

print file ...

lpr -CdhclLmriTn -F*n* **-n** *n* **-p** *n* **-z** *n* **-f** *filter* **-j** *jobname* **-u** *file* **-o** *name* **-q** *string* **-s** *cmd* **-S***n,n,n,A* **-A***n* **-P***name* **-L***n* **+P***n* **+D** **+W***n* **+V***n* **+:xx=#:** [*file* ...]

DESCRIPTION

Print runs the shell command 'pr $*|lpr', which prints a paginated copy of each file on the printer. Note that normal *pr*(1) options can also be used (but NOT lpr options).

Lpr causes the named files to be queued for printing. If no files are named, the standard input is read.

OPTIONS

Options which do not need an additional argument can be concatenated. For example:

"lpr -mhTcp 3" is the same as "lpr -m -h -T -c -p 3"

There are a number of options which are ignored if the **-C** flag is given, these are preceded by an *. The following options are accepted by lpr (any '+' options MUST follow any '-' options that are given):

-A*n*

Number of times to retry an unsuccessful print job. (The default is 3).

-b *name*

banner name to be used on the header page at the beginning of the print job. The default is the username of the owner of the print job. Overrides -h option. The -b option forces output of a header page, regardless of the default set by *lpdun*.

-c *n*

copy flag causes the print job to be copied into the print queue, instead of linked. This is usually slower than linking, but guards against changes that may occur in the file before it is printed.

-C Input is expected to be in a special format, which is currently used only by Fortune:Word. A number of options are ignored if this option is given (see note above).

-d Single sheet mode. Causes the printer to pause after printing each page and wait for a signal from *lpdun*(1).

+D

draft mode printing (faster, and usually lower quality) is used, if supported by the printer.

-f *name*
filter "name" will be used to do the actual printing instead of the normal filter. A full pathname MUST be used. The filter must accept (even if it ignores) the same flags as the normal filter. (see the man page for *lpf*(8).

* **-F** *n*
form length is set to n lines for this job. The default form length is set with *lpdun*(1). This option is ignored if -S is given. If the value given does not match the length of the paper on the printer, top of form will be incorrect for subsequent jobs, unless manually reset.

-h header page is NOT printed at the beginning of the print job. Generally used only if just one user is on the system, since this is the only way to identify the printout. (See also the -b option of *lpdun*(1).) This option is ignored if -b is given.

-j *name*
jobname for this print job. The default is the source filename of the print job if a file is being printed, shell commands default to the command, and standard input defaults to "Standard input."

-i input from a pipe (or standard input) is not copied to a temporary file, as is normally done. This is mainly used for jobs so large that copying them would use up all or most of the free space on the filesystem. Input from a terminal is not allowed, since there would then be two processes reading from the terminal at the same time. Requests for multiple copies force a copy file to be created.

* **-L***n*
line length is set to *n* characters for this job. Output lines longer than this are truncated, if the *-t* option is also given. This option is ignored if *-S* is given. The default line length is set with *lpdun*(1).

* **+L***n*
Changes the left margin to position *n*. (Must follow the +P option if the +P is given.) For example, if *n* is 10, printing starts in column 11.

-m *mail*(1) is sent to the owner of the job, when the job completes. If the owner was set with the *-o* option, and the owner given is not a valid username, mail is sent to the person who submitted the job, unless the owner name contains an @ or an !, in which case it is assumed the owner name is a valid user on another system.

-n *n*
number of copies to be printed, the default is one.

-o *name*
owner of the print job. The default is the username. This is NOT used to determine permission to remove jobs from the queue, but appears on the header page and in the queue listings as the owner of the job. It is also used to determine who mail should be sent to with the *-m* option.

-p *n*
printer the job is sent to. The default is printer 1.

-P*name*
printer type is set to *name* for this job, rather than the default, (which is set by *dtinit*(1).

*** +P***n*
pitch for this job is set to *n*. Some printers (such as the IDS) only support fixed pitches. For this reason only 10, 12, and 15 pitch are supported.

-q *string*
Any comment the user wishes can be placed here. At least the first 15 characters are displayed by *lpq*(1).

-r remove the files (if any) that were submitted after they are queued. This is useful for printing files which are tmp files. Also see the *-u* option.

-s *cmd*
cmd will be executed when the job is ready to be printed, and its standard output will be sent to the printer. Except for the time "cmd" is run, this is almost identical to:

```
cmd | lpr -i
```

No file arguments are allowed with this option. "cmd" is parsed to determine if it contains any Bourne shell metacharacters. If it does, it is run in a subshell, otherwise an attempt is made to run the command directly. If the exec fails, an attempt is then made to run the cmd in a subshell. In both cases, the environment is set to that of the user when lpr was invoked. This is currently used by Fortune:Word to do formatting at print time.

-S*n,n,n,A*
Use the sheet feeder on this printer, if supported. No checks are made to ensure the sheet feeder is actually on the printer. At least 1, and up to 4 comma separated fields follow the 'S'. The first 3 must be numbers from 1 to 3. The first number is the bin for the first page, the second is for the second page, and the third is for the third page. If the fourth field is present and is the letter 'A', the last two bins given will be alternated for the

rest of the job; otherwise the last field given will be used as the bin for the rest of the job. Bin 1 is the 'letterhead' bin, bin 2 is the 'plain paper' bin, and bin 3 is the 'envelope' bin. (For single bin feeders, you are considered to have only a 'letterhead' bin.) Several examples are given below:

lpr -S3,1,2,A

First page is an envelope, second page is letterhead, third page is plain paper; following this, even pages will be letterhead and odd pages will be plain paper.

lpr -S1,1,2

First page is letterhead, second page is letterhead, third page is plain paper; following this, all pages will be plain paper.

The sizes of the paper in each of the bins must be set correctly for some of the supported sheet feeders to work properly. Use the *-S* option of *lpdun*(1) to set and display the sizes.

If more bins are specified than are available for the printer type given (either via command line or from /etc/devtype). Those bins will be converted to 1, the 'plain paper' bin, and a diagnostic will be issued. Similarly, if the -S option is given, and the printer type does not support sheet feeders at all, the option will be ignored and a diagnostic issued.

-t truncation of overly long lines will be done. (line length is set by the *-L* option or by the default set by *lpdun*(1).) The default is to do no truncation; overly long lines will print on the platen.

-T Transparent mode. All characters in the job are sent to the printer exactly as they are. This is mainly so the user can include printer control sequences in their files, such as super/subscripts (this is particularly useful for *nroff*(1) output). This option does not allow reprinting of pages for print jobs, since the filter has no way of knowing where pages end. (See *lpdun*(1) for a description of this). Tabs are not expanded with this option, use *expand*(1) to expand them, if necessary. In addition, LF's are NOT turned into <LF><CR> sequences, as this would alter some printer control sequences.
Unfortunately, only the low 7 bits are used, because using 8 bits requires that the printer accept AND generate 8 bit characters. Some printers, like the NEC35XX series, can accept 8 bit characters, but always generate MARK parity on output. This has the unfortunate side effect of losing flow control, which is unacceptable.

-u *file*
Causes the following file to be removed after printing. This is useful with shell scripts that create or use temporary files. The FULL pathname MUST be given. This option may be given up to ten times to remove multiple files. The file will be removed even if job is not successfully printed.

* **+V***n*
Vertical pitch is set to *n* lines per inch. Currently only 6, 8 and 10 lines per inch are supported, although others may work for some printers. (10 lines per inch is actually 9.6 lines per inch for all currently supported printers, so page length may not come out the way one would expect.)

* **+W***n*
Change to font *n*. This is only supported for printers (such as IDS) which can change their fonts under software control.

-z *n*
The size the user wishes reported by lpq for this job can be placed here. This is useful when no files are given to lpr, since a size would otherwise not be reported.

+:xx=#:
Can be used to pass any capability from the printcap file. xx is any 2 letter sequence, the the printcap entry where the character 'V' occurs, and the :'s are REQUIRED.

FILES

/usr/spool/lpd/pr#
spooler directories (# is the printer number)

/usr/spool/lpd/pr#/[cdelt]*
Control files for use by the spooling system.

/usr/lib/lpd
printer daemon

/usr/lib/lpf
printer filter

/etc/printcap
For printer specific information

/etc/devtype
For correlation between printer #, printer port, and printer type, etc.

/usr/adm/lperr
If this file exists and is writable by all, all messages generated by the spooling system (including those suppressed by silent options) are written here. In addition, messages which are for troubleshooting only, (and hence are never output to a

terminal), are written here. Note that this file grows rapidly and is generally created only for troubleshooting.

SEE ALSO

dtinit(1), *lprm*(1), *lpmv*(1), *lpq*(1), *lpdun*(1), *lpf*(8), *lpd*(8), *pr*(1), *printcap*(5).

NOTES

Each page starts at the right margin (whatever it is). Thus, a line which has a formfeed in it will become TWO lines on output, the second of which will start at the right margin. Note also that a formfeed followed by a newline will result in a blank line at the top of the new page.

Error messages are written by a routine which attempts to decide whether it can output a newline, or if the user is using a screen oriented program, and therefore cannot do a newline (because the screen might be scrolled). The determination is made by checking to see if the terminal from which the job was started is set in raw and/or cbreak mode. If so, a message is written (in a terminal independent manner) to the bottom line of the screen; otherwise it simply outputs the message followed by a newline.

An attempt is made to send error messages to the terminal that started the job (the stderr, stdout, and stdin of lpr are checked in that order until one is found to correspond to a terminal; if one does, it is passed thru to lpd and lpf).

If a spooler program does not inherit the terminal name as in the program above (or if it is not writable), /dev/tty is tried, then stderr, which may, in some cases (e.g. a daemon running jobs started from more than one terminal), be the wrong terminal. The terminal type is set from the TERM shell variable, or if TERM is not set, it is set to FT.

Some messages which are of informational nature only (such as specifying a sheetfeeder when the printer does not support it) are not written to the terminal if a screen oriented program is running (they are still written to the error logging file, if it exists).

LIMITATIONS

For normal printing (without the -T or -C options) the spooling system will not handle videotext characters. They will be printed as '^Yx' where x is the escape characters. Future versions may handle videotext in the same manner as Fortune:Word, by looking up wheel files and mapping videotext sequences to the appropriate output sequences.

NAME

lprm, lpq, lpdun, lpmv - line printer auxillary programs

SYNOPSIS

lprm [-sp *n*] [*QID*] [*filename*] [*owner*] [*jobname*] ...

lpq [**-aqsp** *n*]

lpdun -b *on|off* **-c** *n* **-C -f** *form* **-i -l** *n* **-m***n* **-n** *n* **-N -p** *n* **-P***n* **-F -r** *ribbon* **-s -S[+-][***w1,l1,...w3,l3***] -w** *printwheel* **-z**

lpmv [-sp *n***] printid newid**

DESCRIPTION

These commands all default to printer #1, unless the **-p** *n* option is given (where *n* is the printer number).

IMPORTANT NOTE

Because of the communications between these programs and the filter program(s) (see *lpf*(8)), the action requested may not occur until the printer is ready to accept more data. The printer may not accept data if the printer cover is open, if a ribbon breaks, if the paper runs out, or simply if the data buffer of the printer is full. If for some reason the printer doesn't request more data, while the filter sends data to the printer, nothing will happen. For all supported printers this condition can be cleared by turning the printer off and on. In this case, some data will probably be lost.

lprm

This command allows users to remove jobs from the print queue. Users may only remove their own jobs; the superuser may, however, remove any job. Jobs may be identified by qid, filename, owner, or jobname. These are all reported by *lpq*(1).

-s silent option; normal results are not reported, but errors are.

It is important to note that spaces or special characters in the jobname must be given exactly. It is usually easiest to give the QID. *ALL* jobs matching ANY of the arguments will be removed. Thus, "lprm olson" will remove all of the jobs in the queue submitted by "olson" (also any jobs named "olson").

NOTE: Giving one (or more) filenames for a job which has multiple files (such as 'lpr a.c b.c d.c e.c') results in ALL files for that job being removed. This is always the case, no matter what type of argument is given; if more than one file is associated with a job, they will ALL be removed.

Multiple arguments may be given, as:

lprm 456 789 lpr.c root

Each job removed will be reported by QID, unless the **-s** option was given. Jobs that are printing at the time they are removed are stopped, and a message is printed on the last printed page.

Lpq

This command lists the line printer queues and state. The following options are accepted:

-a all printer queues are displayed. Otherwise, only the information for the printer specified is given. The printer number is listed in the first column for the first job in each queue.

-s display the state information; the default is to list only the queue(s).

-q when given with -s, queue information is displayed, followed by the state information.

Each entry in the queue is printed showing the owner of the queue entry, the queue identification number, the type of entry (doc (WP), file, pipe, or shell), the size (in characters), the file to be printed, the jobname of the entry, any comments attached to the entry, and the number of copies to be printed. Jobs are listed in the order they will be printed. For shell jobs, the shell command is listed under jobname, unless the **-j** *name* option was used with lpr. The jobname and comment fields are used entirely for the jobname/comment, if there is nothing in the other field.

With **-s**, the state, what (if anything) the printer is waiting on, the current ribbon, wheel and form, the line width in characters, the page length in lines, and the process id of the filter currently running are displayed (if the PID is not shown, nothing is printing).

lpdun

Lpdun is used to restart a suspended print job or notify the spooler of a change in the printer's condition. It is also used to set up default paper sizes and other default attributes of the printers. With any of the options that set defaults (e.g. -S, -c, and -P options) or with -z, options that interact with the filter (i.e., -i, -n, and -N), will be ignored. With no options, if a job was waiting on a new sheet, form, etc., it is restarted.

lpdun depends heavily on the normal filter *lpf*(8) being the printing filter. If another filter is used, and it does not work in the same manner, lpdun may not work properly.

-b [*on*|*off*]

If "on" is given, changes printer so it defaults to printing the banner page, if "off", changes default to no banner page. Either way, this can be overridden by options to *lpr*(1).

-c *n*

changes the default line width (in characters) for printer. (See also the **-L***n* option of *lpr*(1).)

-C When given with the -S option, indicates that the sizes given are in centimeters instead of inches.

-f *form*
The form type is set to *form*. This, like *ribbon* is not used by the spooling system, but may be used by Fortune:Word in the future.

-i interrupt the printing of a job, if a job is active. Note: many printers have a large (typically 2000 character) buffer; therefore printing may continue for a page or more after printing is interrupted.

-l *n* changes the default formlength in lines per page for printer. (See also the -F*n* option of *lpr*(1).)

-m*n*
n is the number of pages to be saved in memory by the filter so that they can be reprinted if necessary. The range is 0 to 9, with the default 2. If *n* is 0, nothing is saved, and no reprinting is possible. This option is provided to allow the user some control over how much memory is used by the filter; systems that have little memory or many users will want to keep *n* low.

-n *n*
restarts a suspended print job and gives the number of pages to backup (1 means restart at top of current page, unless a new page was just fed, in which case it means reprint from the previous page). The header page, if any, is never reprinted.

If printing is suspended while reprinting, and reprinting is again requested, reprinting starts *n* pages back from the highest numbered page that has actually been printed, NOT *n* pages back from the page that is currently printing. That is, if pages 1-21 have been printed, then a request is made to reprint 4 pages (18-21), and printing is then stopped at page 19, a request to reprint 2 pages will cause pages 20 and 21 to be reprinted, NOT pages 18 and 19.

If this option is omitted, or *n* is 0, printing resumes at the current position (this is the only exception to the highest numbered page rule). "lpdun" is thus equivalent to "lpdun -n 0".

-N This option is used to initialize or reinitialize a printer with a sheet feeder on it, since some sheetfeeders do not have manual controls to do this. If a job is running or suspended, the filter is told to output the appropriate strings, and then suspend itself. If there is no job running, a special job is submitted to the spooling system requesting that the printer be initialized.

-P*n* *n* is the priority level for the spooler to run at. The range may be from -20 to 20; 0 is the default for most Unix programs. A negative *n* will cause the spooler to run at a higher priority, but will slow other programs down. If *n* is 20 (the default) the spooler will run with a very low priority, putting as little load on the system as possible.

-r *name*
Changes the default ribbon for the printer to *name*.

-s This option is used with single sheet jobs to inform the spooler that a new sheet has been inserted, and printing can resume.

-S If followed by '-', no sheet feeder is in use. If followed by '+", a sheet feeder is in use, and *up* to 6 ',' separated numbers can follow. These are the width and length (in inches by default) of the paper in each of up to 3 bins. See the -S option of *lpr*(1) for the correspondence between bin numbers and their contents. Fields that are left blank (two commas in a row, or a comma immediately after the '+') are left unchanged.

lpdun -S+3.3,4.5,,,8.5,11

sets bin 1 to 3.3 inches wide and 4.5 inches long. The width and length of bin 2 is left unchanged, and bin 3 is set to 8.5 inches wide by 11 inches long.

-w *name*
changes the default wheel for the printer. (See also the +W*n* option of *lpr*(1).)

-z reports the sizes of the paper in the various bins, the width in characters, and the length in lines for single sheet and tractor feed jobs.

Lpdun is also used to set up new printers (in combination with *dtinit*(1)). If the printer spooling directories and/or control files are not found, lpdun will try to create them. (The directory "/usr/spool/lpd" must already exist.)

lpmv

moves a print job to a new position in the queue. *printid* and *newid* are the numbers displayed by *lpq*. in the QID column.

-s silent option; normal results are not reported, but errors are.

Printid is the id of the print job to be moved.

Newid is the print job *printid* will be printed *before*.

Note that moving a job in front of a printing job does NOT interrupt the printing job to print the moved job, however, the moved job will be the next job printed.

FILES

/usr/spool/lpd/pr#
spooler directories (# is the printer number)

The file (in each spooling directory)
that records current state of a printer.

/etc/devtype
The master configuration file for serial devices. Checked by lpq to see what printers are available.

SEE ALSO

lpr(1), *lpf*(8), *lpd*(8).

NAME

ls - list contents of directory

SYNOPSIS

ls [**-abcCdfFgilmqrRstux1**] *name...*
ll [*ls options*] *name...*
lf [*ls options*] *name...*
lr [*ls options*] *name...*

DESCRIPTION

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

There are three major listing formats. The format chosen depends on whether the output is going to a teletype, and can also be controlled by option flags. The default format for a teletype is to list the contents of directories in multi-column format, with the entries sorted down the columns. (Files which are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. This is because the individual file names may be arbitrarily long.) If the standard output is not a teletype, the default format is to list one entry per line. Finally, there is a stream output format in which files are listed across the page, separated by ',' characters. The **-m** flag enables this format; when invoked as *l* this format is also used.

OPTIONS

-a List all entries; usually all files beginning with a '.' are suppressed.

-b Force printing of non-graphic characters to be in octal notation.

-c Use time of file creation for sorting or printing.

-C Force multi-column output, e.g. to a file or a pipe.

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

-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.

-F Cause directories to be marked with a trailing '/' and executable files to be marked with a trailing '*'; this is the default if the last character of the name the program is invoked with is a 'f'.

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

-i Print i-number in first column of the report for each file listed.

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

The mode printed under this option contains 11 characters which are interpreted as follows:

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

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

r	if the file is readable;
w	if the file is writable;
x	if the file is executable;
-	if the indicated permission is not granted.

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

The last character of the mode (normally 'x' or '-') is **t** if the 1000 bit of the mode is on. See *chmod*(1) for the meaning of this mode.

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

-m Force stream output format.

-n Displays the user or group id number instead of user or group id name in a long listing. This option makes a long listing run much more quickly, as the names do not have to be searched for in /etc/passwd or /etc/group, respectively.

-q Force printing of non-graphic characters in file names as the character '?'; this normally happens only if the output device is a teletype.

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-R Recursively list subdirectories encountered.

-s Give size in blocks, including indirect blocks, for each entry.

-t Sort by time modified (latest first) instead of by name, as is normal.

-u Use time of last access instead of last modification for sorting (**-t**) or printing (**-l**).

-x Force columnar printing to be sorted across rather than down the page; this is the default if the last character of the name the program is invoked with is an 'x'.

-1 force one entry per line output format, e.g. to a teletype.

FILES

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

/etc/group
to get group ID's for 'ls -g'.

LIMITATIONS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as "ls -s" is much different than "ls -s | lpr". On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

Column width choices are poor for terminals which can tab.

NAME

man - print out the manual

SYNOPSIS

man [*section*] *title* ...

DESCRIPTION

Man is a shell program which provides on-line access to sections of the printed manual. If a section specifier is given, *man* looks in that section of the manual for the given *titles*. *Section* is an Arabic section number, i.e. 3. If *section* is omitted, *man* searches all sections of the manual and prints all the sections it finds, if any.

Man pipes its output through *more*(1) to stop after each page on the screen. Hit a space to continue, a control-D to scroll 11 more lines when the output stops. *More*(1) is smart enough not to ask for terminal input if the output of *man* has been redirected to a file.

FILES

/usr/man/man?/*

SEE ALSO

more(1).

NAME

menu - execute the Schmidt shell
.unix - provide unix interface for users of the Schmidt shell

SYNOPSIS

menu
.unix

DESCRIPTION

Menu puts you in the "user-friendly" command interpreter.

.unix allows the user to access the unix system directly from the Schmidt shell. The *unixdoor* file contains a list of commands run by *sh*(1). Currently, it contains a single command which calls *sh*(1) so the user can interact with the regular unix shell. By editing this file, a different command processor such as *csh*(1) can be used instead of the regular shell.

NAME

mkdir - make a directory

SYNOPSIS

mkdir *dirname...*

DESCRIPTION

Mkdir creates the specified directories. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

Mkdir requires write permission in the parent directory. The permission bits of the directories will be 777, as modified by the user's umask (see *sh(1)*).

SEE ALSO

chmod(1), rm(1).

DIAGNOSTICS

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

NAME

more, page - file perusal filter for crt viewing

SYNOPSIS

more [**-cdflsu**] [*-n*] [*+linenumber*] [*+pattern*] [*name...*]

page *more options*

DESCRIPTION

More is a filter which allows examination of a continuous text, one screenful at a time on a soft copy terminal. It normally pauses after each screenful, printing **--More--** at the bottom of the screen. A return displays just one more line and a space displays another screenful. Other possibilities are enumerated later.

OPTIONS

-c *More* draws each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

-d *More* prompts the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

-f This causes *more* to count logical, rather than screen lines, meaning long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but do not print when sent to the terminal as part of an escape sequence. Thus *more* thinks that lines are longer than they actually are, and fold lines erroneously.

-l Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

-n An integer which is the size (in lines) of the window which *more* will use instead of the default.

-s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

-u Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a standout mode, *more* will output appropriate escape sequences to enable underlining

or standout mode for underlined information in the source file. The *-u* option suppresses this processing.

+*linenumber*
Start up at *linenumber*.

+/*pattern*
Start up two lines before the line containing the regular expression *pattern*.

If the program is invoked as *page*, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and *k* - 1 rather than *k* - 2 lines are printed in each screenful, where *k* is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to preset any flags desired. For example, if you prefer to view files using the *-c* mode of operation, the *csh* command *setenv MORE -c* or the *sh* command sequence *MORE='-c' ; export MORE* would cause all invocations of *more* , including invocations by programs such as *man* and *msgs* , to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
display *i* more lines, (or another screenful if no argument is given).

^D display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d same as ^D (control-D).

iz same as typing a space except that *i*, if present, becomes the new window size.

is skip *i* lines and print a screenful of lines.

if skip *i* screenfuls and print a screenful of lines.

"q or Q"
Exit from *more*.

= Display the current line number.

v Start up the editor *vi* at the current line.

i/expr
search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current filename and the previous shell command respectively. If there is no current filename, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense).

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current filename and line number.

":q or :Q"
exit from *more* (same as q or Q).

(dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may press the line kill character to cancel the numerical argument being formed. In addition, the user may press the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can press the cancel key. *More* will stop sending output, and will display

the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

FILES

/etc/termcap
Terminal data base

/usr/lib/more.help
Help file

SEE ALSO

csh(1), man(1), msgs(1), script(1), sh(1), environ(5).

NAME

mv - move or rename files

SYNOPSIS

mv [-f] [-i] [-] *file1 file2*

mv [-f] [-i] [-] *file...directory*

mv [-f] [-i] [-] *directory1...directory2*

DESCRIPTION

Mv moves (renames) *file1* to *file2*.

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

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

Mv refuses to move a file onto itself.

Mv also moves (or renames) *directory1* to *directory2* along with all of its files and subdirectories.

OPTIONS

-f Stands for force, overrides any mode restrictions or the -i switch.

-i Stands for interactive mode. Whenever a move is to supercede an existing file, the user is prompted by the name of the file followed by a question mark. If answered with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.

- means interpret all the following arguments to *mv* as filenames. This allows filenames starting with minus.

SEE ALSO

cp(1), ln(1).

LIMITATIONS

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

Directories can only be moved within the same parent directory.

NAME

newgrp - log in to a new group

SYNOPSIS

newgrp [*group*]

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to *login*(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

Newgrp without an argument changes the group identification to the group in the password file; in effect it changes the group identification back to the caller's original group.

A password is demanded if the group has a password and the user himself does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

When most users log in, they are members of the group named **users**.

FILES

/etc/group

/etc/passwd

SEE ALSO

login(1), group(5).

LIMITATIONS

There is no convenient way to enter a password into **/etc/group**.
Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

passwd - change login password

SYNOPSIS

passwd [*name*]

DESCRIPTION

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

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

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the manager can change a password; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), crypt(3), passwd(5).
Robert Morris and Ken Thompson, *UNIX password security*

LIMITATIONS

The password file information should be kept in a different data structure allowing indexed access.

NAME

pr - print file

SYNOPSIS

pr [*option*] ... [*file*] ...

DESCRIPTION

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

OPTIONS

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

-f Use form feeds instead of newlines to separate pages. A form feed is assumed to use up two blank lines at the top of a page. (This option does not affect the effective page length.)

-h Take the next argument as a page header.

-l*n* Take the length of the page to be *n* lines instead of the default 66.

-m Print all *files* simultaneously, each in one column.

-F Fold, rather than truncate, lines in multi-column output.

-n Produce *n*-column output.

+n
Begin printing with page *n*.

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

-t Do not print the 5-line header or the 5-line trailer normally supplied for each page.

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

SEE ALSO

cat(1).

NAME

printstring - prints arguments with possible translation

SYNOPSIS

printstring translation-table [*arg*] ...

DESCRIPTION

Printstring writes its arguments or a translated version of its arguments terminated by a newline 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
- **** backslash
- *****n* the 8-bit character whose ASCII code is the 1, 2 or 3-digit octal number *n*, which must start with a zero. This last escape convention is not yet implemented.

Each argument is looked up in a translation table to determine if a corresponding value exists. If there is a corresponding value it is printed. If no value is found, the argument is printed. This does not apply if the argument begins with a blank or a backslash. If the argument begins with a blank or a backslash the argument is printed untranslated.

Printstring is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

echo(1), sh(1), trans(5).

NAME

ps - process status

SYNOPSIS

ps [akltUvx] [*pid*]

DESCRIPTION

Ps prints process status information. First, *ps* prints a heading line, then the information for each process. The default short listing prints only the PID, TTY, TIME, and CMD fields described below, whereas the long listing prints all fields, as in the following example:

```
FLAGS S UID   PID  PPID CPU PRI NICE SZ WCHAN     TTY TIME CMD
   KC S   0     0     0  70   0   20  2 runout    ?   8:08 swapper
    C S   0     1     0   0  30   20 15 (wait)    ?   0:02 /etc/init
    C S 101    32     1   0  28   20 39 tty6,0    02  0:24 -csh
    C R 101   877   876  42  51   20 37           co  0:01 ps lax
```

The fields are as follows:

F Flags associated with the process:

D: Detached process, parent has died.
U: Process has asked that it be locked in core.
W: Traced process, parent process is waiting for it.
T: Traced process.
S: Process is swapped out.
L: Temporarily locked in core (e.g. for raw I/O).
K: System process, not normal program.
C: Loaded in core.

S The state of the process:

0: nonexistent;
S: sleeping;
W: waiting;
R: running;
I: intermediate;
Z: terminated;
T: stopped.

UID

The user ID of the process owner.

PID

The process ID of the process, which can be used to kill it. The listing is sorted in PID order.

PPID

The process ID of the parent process.

CPU

Amount of processor utilization, used for scheduling.

PRI The priority of the process, ranging from highest priority, -128, to lowest, 127. The process cannot be killed while this number is negative, which should only happen during certain types of I/O.

NICE
Used in priority computation. A lower nice value begets higher priority. Normal value is 20.

SZ The size in 1K core increments of the data and stack portions of the process, i.e. exclusive of the code segment if it is sharable.

WCHAN
The event for which the process is waiting or sleeping; if blank, the process is running. This field is either the name of an internal kernel variable (like runout, lbolt, buf[10], inode[3], etc.), a system call name such as wait or pause (in parentheses), a tty major/minor dev like tty0,0, any of the above with an offset, as in tty1,1+56, or a hex address in rare cases.

TTY
The controlling tty for the process. Examples are "co" for console, "02" for tty02, or "?" if none.

TIME
The cumulative execution time for the process.

CMD
The command and its arguments. A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>. *Ps* makes an educated guess as to the command name and arguments given when a process was created by examining the argument list in the memory image of the process in memory or the swap area, wherever it is. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

OPTIONS

If the *pid* argument is specified, *ps* prints information about only that process.

a gives information about all processes with terminals (ordinarily only one's own processes are displayed).

k uses the file */usr/sys/core* in place of */dev/mem*. This is used only by kernel developers for debugging after a system crash.

l gives a long listing.

t*tty* gives information only about processes associated with terminal *tty*.

U Prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes; followed by the *ps* information.

v prints the accumulated child user and child system time for each process in addition to the accumulated time for the process itself. The l option overrides this option.

x gives information even about processes with no terminal.

FILES

/dev/hd01
this or another disk is the swap device

/dev/mem
core memory

/usr/sys/core
alternate core file

SEE ALSO

kill(1), pstat(1).

LIMITATIONS

Things can change while *ps* is running; thus, the picture it gives is only a close approximation to reality. In rare cases, *ps* can die from a segmentation fault because the kernel tables change too quickly. Some data printed for defunct processes is irrelevant.

NAME

pwac - maintain account and group information files

SYNOPSIS

/m/sysman/pwac [*option*] ... [*name*] ...

/m/sysman/pwac -a [*option*] ... [*name*] ...

/m/sysman/pwac -d [*option*] ... [*name*] ...

/m/sysman/pwac -t [*option*] ...

DESCRIPTION

Pwac is used to add, modify, delete and list the information maintained in the files */etc/passwd* and */etc/group*.

The simple version of the command lists the information for the specified accounts or groups and makes any desired changes.

OPTIONS

Names and options can be given in any order. Options can also be combined. The following options are recognized by *pwac:*

-a adds new accounts or groups. The information is normally taken from a template account or group called *proto* . Normally names must start with a lower case letter and contain only lower case letters and digits. The **-l** option allows any names to be added.

-d deletes accounts or groups.

-t lists all accounts or groups in a table format. Most items are printed as they are stored. A few items are not printed but are instead labeled flags. Currently they are **'y'** if the account or group is on, **'n'** if the account or group is off, and **'p'** if the account or group has a password. A group that has been turned off can still be used by its members. Other people, however, are unable to *newgrp*(1) to this group even if they know the password.

In addition, the following options can be used:

-F *format*

Use *format* to print only specific information about an account or group. Most characters are printed without expansion. Many letters, however, expand to values. For example, the character 'S' expands to the shell, the character 'H' expands to the home and so forth. A back-slash ('\') causes the next character to be printed without expansion.

-g Group operations are performed.

-l Values are taken literally.

-q (quiet) prints only error messages.

-T *name*
Use the *name* as a template instead of the default account "proto".

The following options can be used for adding or modifying:

-A *group*
Adds the accounts specified as members of *group*. This option can be repeated several times to add accounts to several groups.

-C *comment*
Use *comment* as the comment to be associated with the specified accounts. On some computers this comment is used to store the real name of the person who uses this account. On other systems, it is used to store job control information for sending requests to other systems.

-D *group*
Deletes the accounts specified as members of *group* This option can be repeated several times to delete accounts from several groups. It is not an error to attempt to delete an account from a group in which it is not a member.

-G *group or number*
Use *group* as the default group for the specified accounts. When used with a group operation use *number* as the group-id-number in the same manner as the user-id-number described above.

-H *directory*
Use *directory* as the new home directory. The character *'&'* is replaced with the account or group name. The **-l** option prevents this expansion. This is useful when working with accounts which are to be used as the templates for making other accounts.

-n Turn the specified accounts or groups off. This has the same effect as if the account or group has a password which is unknown.

-N *name*
Use *name* as the new name of the specified accounts or groups. This should normally not be done with more than one account or group. The rules for a valid name are the same as those used when adding accounts or groups.

-P *password*
Use *password* as the password for the specified accounts or groups. This password is normally encrypted before being stored. Using the **-l** option this can be prevented. In either case

an empty password indicates no password.

-S *shell*
Use *shell* as the new shell.

-U *number*
Use *number* as the user-id-number. If more than one account is listed the number is used for the first account, the number plus one is used for the second account and so forth.

-y Turn the specified accounts or groups on.

SEE ALSO

newgrp(1), passwd(1), su(1), passwd(5), group(5).

FILES

/etc/passwd
account information file

/etc/group
group information file

/etc/ptmp
lock file

/etc/passwd.temp
account information temporary

/etc/group.temp
group information temporary

/etc/ptmp.link
lock file temporary

LIMITATIONS

There is no way to specify that only some items are to be taken literally.

The rules affecting group membership are somewhat confusing.

NAME

pwd - working directory name

SYNOPSIS

pwd

DESCRIPTION

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

SEE ALSO

cd(1), csh(1).

NAME

rm, rmdir - remove (unlink) files

SYNOPSIS

rm [**-f**] [**-i**] [**-r**] [**-**] *file* ...

rmdir *dir*...

DESCRIPTION

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

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used.

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

OPTIONS

-f No questions are asked and no errors are reported.

-i Asks whether to delete each file (interactive option) and, under **-r**, whether to examine each directory.

-r No errors messages are given and *rm* recursively deletes the entire contents of the specified directory and the directory itself.

- The null option indicates that all the arguments following it are to be treated as filenames. This allows the specification of filenames starting with a minus.

SEE ALSO

unlink(2).

DIAGNOSTICS

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

NAME

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

SYNOPSIS

sh [-ceiknrstuvx] [*arg*]...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See **invocation** for the meaning of arguments to the shell.

Commands.

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

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

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

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

for *name* [**in** *word* ...] **do** *list* **done**

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

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**

A **case** command executes the *list* associated with the first pattern that matches *word*. The form of the patterns is the same as that used for filename generation.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

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

while *list* [**do** *list*] **done**

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

(*list* **)**

Execute *list* in a subshell.

{ *list* **}**

list is simply executed.

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

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

Command substitution.

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

Parameter substitution.

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

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

${*parameter***}**

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

${*parameter-word*}
If *parameter* is set then substitute its value; otherwise substitute *word*.

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

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

${*parameter+word*}
If *parameter* is set then substitute *word;* otherwise substitute nothing.

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

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

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

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

HOME
The default argument (home directory) for the **cd** command.

PATH
The search path for commands (see **execution**).

MAIL
If this variable is set to the name of a mail file then the shell informs the user of the arrival of mail in the specified file.

PS1
Primary prompt string, by default '$ '.

PS2
Secondary prompt string, by default '> '.

IFS Internal field separators, normally **space**, **tab**, and **newline**.

Blank interpretation.
After parameter and command substitution, any results of substitution are scanned for internal field separator characters (those found in **$IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null

arguments (those resulting from *parameters* that have no values) are removed.

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

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

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

; & ( ) | < > newline space tab

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

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

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

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

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

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

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

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

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

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

It should be noted that the command *'program < file > file'* will probably destroy *file* before calling *program* and should be avoided.

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

... 2>&1

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

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

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

any modifications or additions, all of which must be noted in **export** commands.

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

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

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following prints 'a=b c' and 'c':

```
echo a=b c
set -k
echo a=b c
```

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

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

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

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

: No effect; the command does nothing.

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

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

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

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

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

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

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

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

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

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

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

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

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

-x Print commands and their arguments as they are executed.

- Turn off the **-x** and **-v** options.

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

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

shift

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

times

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

trap [*arg*] [*n*] ...

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

umask [*nnn*]

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

wait [*n*]

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

Invocation.

If the first character of argument zero is -, commands are read from **$HOME/.profile**, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

-c *string*

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

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

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

The remaining flags and arguments are described under the **set** command.

FILES

$HOME/.profile

/tmp/sh*

/dev/null

SEE ALSO

csh(1), exec(2), test(1).

DIAGNOSTICS

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

LIMITATIONS

If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document. A garbage file /tmp/sh* is created, and the shell complains about not being able to find the file by another name.

NAME

sleep - suspend execution for an interval

SYNOPSIS

sleep *time*

DESCRIPTION

Sleep suspends execution for *time* seconds.

EXAMPLES

To execute a command after a certain amount of time:

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

To execute a command every so often from the Bourne shell:

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

SEE ALSO

alarm(2), sleep(3).

LIMITATIONS

Time must be less than 2147483647 seconds.

NAME

sort - sort or merge files

SYNOPSIS

sort [**-bcdfimnor**] [**-t***x*] [+*pos1* [*-pos2*]] ... [**-o***name*] [**-T***directory*] [*name*] ...

DESCRIPTION

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

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the options, one or more of which can appear.

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

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

OPTIONS

b Ignore leading blanks, spaces, and tabs in field comparisons.

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

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

f Fold uppercase letters onto lowercase.

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

m Merge only, the input files are already sorted.

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

o The next argument is the name of an output file to use instead of the standard output. This file can be the same as one of the inputs.

r Reverse the sense of comparisons.

t*x* 'Tab character' separating fields is *x*.

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

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

EXAMPLES

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

```
sort -u +0f +0 list
```

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

```
sort -t: +2n /etc/passwd
```

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

```
sort -um +0 -1 dates
```

FILES

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

SEE ALSO

comm(1), join(1), uniq(1).

DIAGNOSTICS

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

LIMITATIONS

Very long lines are silently truncated.

NAME

stty - set terminal options

SYNOPSIS

stty [*option...*]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, placing its output on the diagnostic output. With no argument, it reports the speed of the terminal and the settings of the options which are different from their defaults. With the argument "all", all normally used option settings are reported. With the argument "everything", everything *stty* knows about is printed.

OPTIONS

cbreak	make each character available to *read*(2) as received; no erase and kill processing, but all other processing (interrupt, quit, ...) is performed
-cbreak	make characters available to *read* only when newline is received
cooked	same as '-raw'
echo	echo back every character typed
-echo	do not echo characters
ek	set erase and kill characters to # and @
even	allow even parity input
-even	disallow even parity input
lcase	map upper case to lower case
-lcase	do not map case
-nl	allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl	accept only new-line to end lines
odd	allow odd parity input
-odd	disallow odd parity input
raw	raw mode input (**no** input processing (erase, kill, interrupt, ...); parity bit passed back)
-raw	negate raw mode
tabs	preserve tabs
-tabs	replace tabs by spaces when printing
tandem	enable flow control,(the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
-tandem	disable flow control

For the following commands which take a character argument *c*, you may also specify *c* as the "u" or "undef", to set the value to be undefined. A value of "^x", a 2 character sequence, is also interpreted as a control character, with "^?" representing delete.

brk *c* set break character to *c* (default undefined.) This character is an extra wakeup causing character.

bs0 bs1 select style of delay for backspace

cr0 cr1 cr2 cr3
select style of delay for carriage return (see *ioctl*(2))

dec set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and "newcrt".)

eof *c* set end of file character to *c* (default control D.)

erase *c* set erase character to *c* (default '#', but often reset to ^H.)

ff0 ff1 select style of delay for form feed

intr *c* set interrupt character to *c* (default DEL or ^? (delete), but often reset to ^C.)

kill *c* set kill character to *c* (default '@', but often reset to ^U.)

nl0 nl1 nl2 nl3
select style of delay for linefeed

quit *c* set quit character to *c* (default control \.)

start *c* set start character to *c* (default control Q.)

stop *c* set stop character to *c* (default control S.)

tab0 tab1 tab2 tab3
select style of delay for tab

tek set all modes suitable for Tektronix 4014 terminal

ti700 set all modes suitable for Texas Instruments 700 series terminal

tn300 set all modes suitable for a General Electric TermiNet 300

tty33 set all modes suitable for the Teletype Corporation Model 33 terminal.

tty37 set all modes suitable for the Teletype Corporation Model 37 terminal.

vt05 set all modes suitable for Digital Equipment Corp. VT05 terminal

0 hang up phone line immediately

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 (exta) extb
Set terminal baud rate to the number given, if possible.

A teletype driver which supports more functionality that the basic driver is introduced in *newtty*(4) and fully described in *tty*(4). The following options apply only to it.

crt Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)

-crt Turn off options set by **crt**.

crtbs Echo backspaces on erase characters.

crterase Wipe out erased characters with "backspace-space-backspace."

-crterase Leave erased characters visible; just backspace.
crtkill Wipe out input on like kill as in **crterase.**
-crtkill Just echo line kill character and a newline on line kill.
ctlecho Echo control characters as "^*x*" (and delete as "^?".) Print two backspaces following the EOT character (control D).
-ctlecho Control characters echo as themselves; in cooked mode EOT (control-D) is not echoed.
decctlq After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.
-decctlq After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)
etxack Diablo style etx/ack handshaking (not implemented).
flusho Output is being discarded usually because the user typed control O (internal state bit).
-flusho Output is not being discarded.
litout Send output characters without any processing.
-litout Do normal output processing, inserting delays, etc.
mdmbuf Start/stop output on carrier transitions (not implemented).
-mdmbuf Return error if write attempted after carrier drops.
new Use new driver (switching flushes typeahead).
nohang Don't send hangup signal if carrier drops.
-nohang Send hangup signal to control process group when carrier drops.
pendin Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).
-pendin Input is not pending.
prterase For printing terminal echo erased characters backwards within "\" and "/".
tilde Convert "~" to "`" on output (for Hazeltine terminals).
-tilde Leave poor "~" alone.

The following special characters are applicable only to the new teletype driver and are not normally changed.

flush *c* set flush output character to *c* (default control O.)
lnext *c* set literal next character to *c* (default control V.)
rprnt *c* set reprint line character to *c* (default control R.)
werase *c* set word erase character to *c* (default control W.)

The following options control *page mode*, a Fortune enhancement. Page mode may be used only after *pageen*, and then comes into effect with a *pageon* command or when one of the page mode control characters is pressed.

pageen Enable page mode.
pagelen *n* Set page length to n (default 25).
pageon Turn on page mode, if enabled.

Page mode prints out <<STOPPED>> at the bottom of the screen and waits for input, whenever *pagelen* lines has been printed, and interesting stuff would scroll off the top of the screen. (See *more(1)*). When the <<STOPPED>> prompt is printed, one of the following characters will print the next page, the next line, the next half-page, or clear the screen and print a page. Any other character will continue printing and turn off page mode.

clear *c* set clear character to *c* (default control N.)
nxhalf *c* set next half page character to *c* (default control B.)
nxpage *c* set next page character to *c* (default control F.)
nxline *c* set next line character to *c* (default control E.)

SEE ALSO

tabs(1), tset(1), ioctl(2), newtty(4), tty(4).

NAME

tee - pipe fitting

SYNOPSIS

tee [-a] [-i] [*file*] ...

DESCRIPTION

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

OPTIONS

-a Causes the output to be appended to the *files* rather than overwriting them.

-i Ignores interrupts.

NAME

test - condition command

SYNOPSIS

test *expr*

DESCRIPTION

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

The following primitives are used to construct *expr*.

-d file
: true if the file exists and is a directory.

-f file
: true if the file exists and is not a directory.

-n s1
: true if the length of the string *s1* is nonzero.

n1 **-eq** n2
: true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, or **-le** can be used in place of **-eq**.

-r file
: true if the file exists and is readable.

-s file
: true if the file exists and has a size greater than zero.

s1 true if *s1* is not the null string.

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

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

-t [fildes]
: true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

-w file
: true if the file exists and is writable.

-z s1
: true if the length of string *s1* is zero.

These primaries can be combined with the following operators:

-a binary *and* operator

-o binary *or* operator

(expr)
parentheses for grouping.

! unary negation operator

-a has higher precedence than **-o**. Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO

find(1), sh(1).

NAME

true, false - provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully.

EXAMPLE

True and **false** are typically used in input to *sh*(1) such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tty - get terminal name

SYNOPSIS

tty

DESCRIPTION

Tty prints the pathname of the user's terminal.

DIAGNOSTICS

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

NAME

wait - await completion of process

SYNOPSIS

wait

DESCRIPTION

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

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

SEE ALSO

sh(1).

LIMITATIONS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for. (This does not apply to *csh*(1).) *Csh* prints out the names and process numbers of all background processes when *wait* is interrupted, while *sh* does not.

NAME

wall - write to all users

SYNOPSIS

wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be manager to override any protections the users may have invoked.

FILES

/dev/tty?

/etc/utmp

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

'Cannot send to ...' when the open on a user's tty file fails.

NAME

what - identify SCCS files

SYNOPSIS

what *files*

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for @(#) (this is **@(#)** at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

prints

```
f.c:  identification information
f.o:  identification information
a.out:
      identification information
```

What is intended to be used in conjunction with the SCCS command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

LIMITATIONS

It's possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

who - who is on the system

SYNOPSIS

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

DESCRIPTION

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

Without an argument, *who* examines the /etc/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file is /usr/adm/wtmp, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with '/dev/' suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in 'who am I' (and also 'who are you'), *who* tells who you are logged in as.

OPTIONS

-f prints in Fortune friendly format.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5).

NAME

write - write to another user

SYNOPSIS

write *user* [*ttyname*]

DESCRIPTION

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

Message from yourname yourttyname...

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

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

Permission to write can be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

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

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

FILES

/etc/utmp
to find user

/bin/sh
to execute '!'

SEE ALSO

mail(1), mesg(1), who(1).

NAME

intro - introduction to special files

DESCRIPTION

Each type of device is accessed through a 'driver' module in the kernel. These drivers can be accessed through 'special files', which can be opened, closed, read, and written, just like ordinary disk files. Some of the special files support lseek(2), and many of them support ioctl(2). By heavily entrenched convention, unix special files are found in the /dev directory.

Special files are created by calling the mknod(2) system call. The mknod(1) program can be used to make a special file from the shell, but that is never necessary on the Fortune Systems 32:16 because when the system comes up, the program mkdevs(8) is run, which correctly makes the files in /dev automatically. Manual pages in Section 4 describe the device drivers on the Fortune Systems 32:16.

These special files in /dev are found on all configurations:

tty00	Built-in keyboard
tty01	Rear serial port
console	System console
conf	Configuration information.
mem	Memory.
kmem	Kernel data memory
null	Empty file on input, black hole on output.
tty	The controlling terminal of the opening process.
fd0[0-7]	Built-in floppy disk buffered (block) interface, partitions 0-7
rfd0[0-7]	Built-in floppy disk raw (character) interface, partitions 0-7
fd[1-3][0-7]	Other optional floppy disks, buffered interface.
rfd[1-3][0-7]	Other optional floppy disks, raw interface.

Some files for accessing optional plug-in device controller cards:

ttyxx	Terminal special files, from tty02 on up. For accessing Comm-A, cards.
hd[0-3][0-7]	Optional hard disk(s) buffered (block) interface, partitions 0-7
rhd[0-3][0-7]	Optional hard disk(s) raw (character) interface, partitions 0-7

Some of these devices can be accessed through other names in /dev for the convenience of certain application programs.

lp	Line printer.
tar	Default device used by tar(1), same as rfd02.
cul0, cua0	tty ports for cu(1) and uucp(1).

SEE ALSO

mkdevs(1), mknod(1), mknod(2), ttys(8),

All of Section 4.

NAME

console - Fortune 32:16 system console

DESCRIPTION

/dev/console is a character terminal. The built-in keyboard port, also known as */dev/tty00*, operates at 2400 baud. The keyboard is a standard ASCII device. The front port is not a standard RS-232 port, it only drives 5 volts, instead of the standard 12. The built-in 25x80 monitor operates at a maximum effective rate of 45000 characters/second. It operates according to AT&T Presentation Level Protocol, as defined in */usr/include/videotex.h*.

FILES

/dev/console

/dev/tty00

/usr/include/videotex.h

SEE ALSO

tty(4), reconf(8).

DIAGNOSTICS

All UNIX internal problems are reported on the built-in monitor.

NAME

floppy - built-in floppy disk drive

DESCRIPTION

The on-board floppy disk controller supplied with the Fortune 32:16 is based on the NEC 765A. The software currently supports MFM, 80-track, double-sided, double-density (MFM) disks with 5 1K blocks per track. This adds up to 800K bytes per disk. There can be up to 3 additional drives attached. See diskconf(5) to find out how floppy disks are arranged.

FILES

/dev/fd[0-4][0-7]

/dev/rfd[0-4][0-7]

SEE ALSO

disk(5)

NAME

Hard Disk - Winchester disks

DESCRIPTION

The hard disks currently offered for the Fortune 32:16 are 5-1/4" Winchester disks, based on the Seagate-506 electrical interface. The controller is a version of the Western Digital 1000 Hard Disk Controller. Each controller can support 4 disk drives. The maximum effective throughput (by dumping a large disk file into /dev/null) is around 80,000 bytes/second. The raw device (/dev/rhd??) provides faster throughput in large sequential transfers. The disks are laid out in from 0 to 7 partitions, described by a table at the beginning of the disk. See diskconf(5).

FILES

/dev/hd[0-3][0-7]

/dev/rhd[0-3][0-7]

SEE ALSO

disk(5), diskconf(5).

NAME

mem, kmem, conf - main memory

DESCRIPTION

Mem is a special file that is an image of the main memory of the computer. It can be used, for example, to examine the system.

Byte addresses in *mem* are interpreted as physical memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device addresses is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel data memory, rather than physical memory, is accessed.

/dev/conf is a window onto a small area in low core which contains a copy of the data in the EAROM. It is the only memory file that is accessible to user-written programs.

FILES

/dev/mem

/dev/kmem

NAME

newtty - summary of the "new" tty driver

SYNOPSIS

stty new

stty new crt

DESCRIPTION

This is a summary of the new tty driver, described completely, with the old terminal driver, in *tty*(4). The new driver is largely compatible with the old but provides additional functionality for page control.

CRTs and printing terminals.

The new terminal driver acts differently on CRTs and printing terminals. On CRTs at speeds of 1200 baud or greater it normally erases input characters physically with backspace-space-backspace when they are erased logically; at speed under 1200 baud this is often unreasonably slow, so the cursor is normally moved to the left. This is the behavior when you say stty new crt; to have the tty driver always erase the characters say stty new crt crterase crtkill, to have the characters remain even at 1200 baud or greater say stty new crt -crterase -crtkill.

On printing terminals the command stty new prterase should be given. Logically erased characters are then echoed printed backwards between a '\' and an '/' character.

Other terminal modes are possible, but less commonly used; see *tty*(4) and *stty*(1) for details.

Input editing and output control.

When preparing input the character # (normally changed to ^H using *stty*(1)) erases the last input character, ^W the last input word, and the character @ (often changed to ^U) erases the entire current input line. A ^R character causes the pending input to be retyped. Lines are terminated by a return or a newline; a ^D at the beginning of a line generates an end-of-file.

Control characters echo as ^x when typed, for some x; the delete character is represented as ^?.

The character ^V can be typed before *any* character so it can be entered without its special effect. For backwards compatibility with the old tty driver the character '\' prevents the special meaning of the character and line erase characters, much as ^V does.

Output is suspended when a ^S character is typed and resumed when a ^Q character is typed. Output is discarded after a ^O character is typed until another ^O is typed, more input arrives, or the condition is cleared by a program (such as the shell just before it

prints a prompt.)

Page Mode.

When page mode is set, printing more than a screenful of characters stops the display, the legend "<<STOPPED>>" prints at the bottom of the screen, and the user must type a character before the printout continues. The *nxpage* character (default ^F) shows another page. The *nxhalf* character (default ^B) shows another half-page. The *nxline* character (default ^E) shows another line, and the *nxpage* character (default ^N) clears the screen and shows a page.

Stty pageen enables the mode. Enabled means that is possible to turn on page mode. If the mode is not enabled, the above four special characters are not interpreted. Page mode can then actually be turned on when cbreak and raw modes are off, and echo mode is on. When the mode is enabled, any of the above four characters, or the command *stty pageon,* turns page mode on. The printer restart (default ^Q) restarts the printout and turn page mode off. Typing any other character to the "<<STOPPED>>" prompt is equivalent to ^F. *Stty pagelen 25* sets the length of the page to be used, the default is 25 lines, the screen length of the Fortune Intelligent Workstation. **Signals.**

A non-interactive program is interrupted by a ^? (delete); this character is often reset to ^C using *stty*(1). A quit ^\ character causes programs to terminate like ^? does, but also causes a core image file to be created which can be examined with a debugger. This is often used to stop runaway processes. Interactive programs often catch interrupts and return to their command loop; only the most well debugged programs catch quits.

See *tty*(4) for a more complete description of the new terminal driver.

SEE ALSO

csh(1), newcsh(1), stty(1), tty(4).

LIMITATIONS

There is always room for more complexity in this area. The page mode scheme is wired into Fortune terminal control codes. To operate well with cursor-control programs, it should be sensitive to cursor controls, this might downgrade efficiency.

NAME

null - data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

sio, comm-a - Standard and optional serial I/O ports

DESCRIPTION

The Fortune 32:16 has two asynchronous serial ports as standard equipment. One is used for the keyboard input and its beeper output. The other is available for RS-232C connection at the back of the main unit. Each optional Communications-A board has either two or four asynchronous serial ports.

All of these devices are accessed through the 'tty' interface described in tty(4).

The functionality of these serial I/O ports is provided by Zilog Z-80 SIO (motherboard ports) and DART (Comm-A ports) chips. The Zilog DART is a functional subset of the Z-80 SIO, lacking only the synchronous capabilities. These ports can operate at the standard UNIX speeds, 0-19200 baud, with the exception that Comm-A ports cannot operate at 200 baud. The setting for the input baud rate determines both the input and output baud rates. The output baud rate setting is explicitly ignored. The default baud rates for tty00 and tty01 are determined by values stored in the EAROM. See reconf(8).

The DART interrupts the CPU for each character input or output; however, for efficiency, input characters are batched up and sent to the reading process with a delay of no more than 1/20 of a second.

The device file names for the standard SIO ports are /dev/tty00 (built-in keyboard), and /dev/tty01, (RS-232C port at rear of main unit). The device file names for the Comm-A ports are /dev/tty02, /dev/tty03, etc. Later, other communications devices may also be known by ttyxx device names, the actual name assignment depending on the placement of the boards in the I/O option slots.

Electrically, the inputs and outputs of the serial ports described here appear at the back of the main unit on RS-232C connectors. (RS-232C is the Electronic Industries Association standard for serial I/O.) The world is divided into two kinds of equipment, as far as RS-232C connectors are concerned: Data Communication Equipment (DCE), e.g. modems, and Data Terminal Equipment (DTE), e.g. CRT terminals. You can connect DCE to DTE with a simple, straight-through cable. A special adaptor cable known as a "null modem" must be used to connect DTE to DTE, and a "null terminal" adaptor cable must be used to connect DCE to DCE. A port on a host computer might be connected to either a DCE or DTE depending on the application, requiring a special cable in one of the two cases.

The serial ports on the computers used for the initial development of UNIX were wired as DTE, and the modem control signals are described in the software and the manual pages as if this were the

case (Fortune retains this nomenclature for standardization with UNIX practice). However, the Fortune 32:16 RS-232C ports are wired as DCE so that terminals can be connected with a simple, straight-through cable. Therefore, modems must be attached with special adaptor cables. All of this can be a little confusing, so be careful to note that when the tty(4) manual page talks about DCD, you can think of this as if it were describing the DCD signal at the modem end of a modem adaptor cable. (In fact, on the host port, it corresponds to the DTR pin.) In general, therefore, the tty(4) man page nomenclature and the software names for the modem control bits follow the names at the modem end of a modem adaptor cable as shown below. The tables below show the correspondence between the DART signals, RS-232C connector pinouts, and modem cable connections. Note that the Zilog Z-80 DART is designed to be used as DTE.

The pinouts on the RS-232C ports look like this:

DART		PIN	NAME
RxD	<---	2	TxDTransmitted Data
TxD	--->	3	RxDReceived Data
CTS	<---	4	RTSRequest to Send
DTR	--->	5	CTSClear to Send
RTS	--->	6	DSRData Set Ready
GND	----	7	GNDGround
DTR	--->	8	DCDData Carrier Detect
DCD	<---	20	DTRData Terminal Ready
RI	<---	25	TRBTrouble

Note that the use of pin 25 is not officially defined in the RS-232C specification, but many printers and modems use it to indicate some problem (out of paper, taken off line, etc.).

When a port is to be connected to a modem, an adaptor cable must be used. (Note that this adaptor cable is not the same as a 'null modem' adaptor.) This adaptor cable has the following connections:

DART	HOST			MODEM		
RxD	TxD	2	<---	3	RxD	Received Data
TxD	RxD	3	--->	2	TxD	Transmitted Data
CTS	RTS	4	<---	5	CTS	Clear to Send
DTR	CTS	5				
RTS	DSR	6	--->	4	RTS	Request to Send
GND	GND	7	----	7	GND	Ground
DTR	DCD	8	--->	20	DTR	Data Terminal Ready
DCD	DTR	20	<---	8	DCD	Data Carrier Detect
RI	BUSY	25	<---	6	DSR	Data Set Ready

The end of the cable that connects to the Fortune 32:16 must terminate in a male connector.

NOTE: Noise on the input control lines on pins 4, 20, and 25 can put a heavy interrupt load on the CPU, crippling or even stopping system activity. Therefore, these pins are all pulled to -12 volts through a 15k ohm resistor (inactive logic level), so that if left unconnected, they do not fluctuate from noise. A connected peripheral device must actively pull them up to the high level to change their state. As per industry standard, pin 9 is connected to +12 volts, and pin 10 is connected to -12 volts, for the benefit of short-haul modems. These voltages will current-limit at 100 milliamperes. (In the official EIA RS-232C spec, these pins are defined as "Reserved for dataset testing".)

The keyboard jack on the front of the unit is intended only for connection of the standard Fortune keyboard. The keyboard input and output signals are connected directly to the Z-80 DART chip, which does *not* use RS-232C voltage levels. Permanent damage is done to that chip if you connect an RS-232C device to this jack. For the unusual case that needs to use the front keyboard port for something other than the normal keyboard, the pinouts look like this:

1	----	Shield
3	----	GND
4	----	+5 V
5	----	+5 V
6	--->	Data to keyboard.
7	<---	Data from keyboard.
8	----	gnd

FILES

/dev/tty[01][0-9]

/dev/console

SEE ALSO

Zilog Z-80 SIO Technical Manual
tty(4)

LIMITATIONS

Because of a limitation of the baud rate generation hardware, the 200 baud speed is not supported on the Comm-A. The serial port on the back of the motherboard handles input slightly more efficiently than the Comm-A ports, its use is preferred for high-speed input from another computer.

NAME

tty - general terminal interface

SYNOPSIS

#include <sgtty.h>
#include <sys/tty.h>
#include <sys/ioctl.h>

DESCRIPTION

The serial communications lines, **/dev/tty***, and the console keyboard-screen combination, **/dev/console**, are interfaced to the system through this driver. These devices are called tty devices, or just tty's.

What happens when you open, close, read, and write a tty depends heavily on the many options you can set with calls to ioctl(2). We will approach the matter by describing all the basic mechanisms and how the options affect them. Some of the options actually affect the hardware, such as setting the speed of a serial device, but most of the options affect the software behavior of the device.

Unfortunately, you won't find a lot of cleanliness, modularity, or regularity in anything having to do with tty devices. The whole subject is an incredible morass of compatability with past versions of the system, Berkeley extensions, Fortune extensions, and the painful truth that dealing with serial terminals in the real world is like unto the problems faced at the Tower of Babel. We refer the reader to the documentation for the standard Bell Version 7 release or the Berkeley release, if he wants to know which features of this driver are standard Version 7, which came from Berkeley, and which are Fortune extensions.

For our discussion of control characters, we will often use the official ASCII names as found in ascii(7). We will use all caps for these names, as in CR for carriage return.

Line disciplines.

Tty devices do all their input and output through a kind of driver called a *line discipline*. In the case where there is more than one line discipline available, line discipline switching is accomplished with the **TIOCSETD** ioctl:

int ldisc = LDISC; ioctl(filedes, TIOCSETD, &ldisc);

where **LDISC** would be the manifest constant from <sys/ioctl.h> for the desired line discipline. The active line discipline can be obtained with the **TIOCGETD** ioctl. Pending input is discarded when the line discipline is changed.

In this version of the system there is only one discipline available, the new tty line discipline from the 4.1 Berkeley Software Distribution, with further extensions added at Fortune Systems. The

manifest constant for this line discipline, found in <sys/ioctl.h>, is NTTYDISC.

This section describes both the characteristics of this particular line discipline and characteristics that are common to all line disciplines; unfortunately, the distinction is somewhat blurred.

First open and last close.

Like most UNIX devices, a tty is in a dormant state when it is not open by any process. It makes the transition from this dormant state to an active state when a process opens it. This kiss of life is referred to as 'first open'. Many processes can have the device open at the same time, and it will remain active until the last of those processes closes it, an event known as 'last close'.

The control terminal and process groups.

Each process has a pointer to a tty known as its *control terminal.* The special file **/dev/tty** is a handy name for the control terminal of the process that opens it. This special device can be used by programs that wish to be sure of writing messages on the control terminal no matter how output has been redirected. It can also be used by programs that demand a file name for output, when terminal output is desired and it is tiresome to find out which terminal is currently in use.

Each process has a process ID and a process group ID. There can be only one process with a certain process ID, but more than one process can have a given process group ID. The process group ID of 0 is the null process group, i.e. belonging to process group 0 is like belonging to no process group. Each tty device has a process group ID associated with it.

If a process whose process group ID is 0 opens a tty device, then that tty device becomes the control terminal for that process, and the process group ID of the process is set to that of the tty device unless the process group ID of the tty is 0, in which case the process group ID of both the opening process and the tty device are set to the process ID of the opening process. On last close, the process group ID of the tty is set to 0. Under various conditions, a tty device will send a signal to all the processes with its process group ID.

The **TIOCGPGRP** ioctl can be used to read the process group ID of the tty, and the **TIOCSPGRP** ioctl can be used to set it.

The process group ID and the control terminal of a process are both inherited by a child process during a *fork*(2), even if the control terminal is closed.

Basic modes.

There are several shorts, longs, and structures in the kernel that contain bits, characters, and numbers used by a tty to determine its mode of operation and its current state. These are called things like the 'tty state word', the 'local modes word', the 'sgtty structure', etc. For now, we will just talk about these things and what they do. You can read and write them with various ioctls which will be summarized later, and you can use the stty(1) command to display and to modify these things from the shell.

On first open, the tty characteristics are initialized to default values, and changes persist only until last close. The **TIOCSAVEMODES** ioctl causes the current characteristics to be saved as the defaults for the tty on subsequent first opens. For normal interactive use, process 1 (init(8)) forks a child which opens a tty then execs getty(8) which sets up tty modes then execs login(1). Thus most of the defaults are quickly overridden by getty(8) for login terminals.

The sg_flags word.

Historically, the oldest of the mode words in the tty driver is the **sg_flags** word of the **sgtty** structure. There are two bits in the **sg_flags** word of the **sgtty** structure which determine one of three major modes of operation, characterized by the amount of processing done on the input and output characters. The bits are **RAW** and **CBREAK**. If the **RAW** bit is set, it overrides the **CBREAK** bit. The major features of these modes are:

cooked — The name comes from the fact that this mode is not **RAW**. It is also not **CBREAK**, but once upon a time there was only **RAW** and cooked (Ah, the History of Comedy). In this mode, the driver collects 7-bit input characters, allowing editing like backspacing to be done on the data. A process reading from the device will not get any data until a NL or an *EOT* **(t_eofc)** character, normally ^D, is entered, at which point it gets all the buffered data up through the NL or up to the *EOT* (the *EOT* is not passed to the reading process). All driver functions (input editing, interrupt generation, output processing such as delay generation and tab expansion, etc.) are available in this mode.

CBREAK — This mode is just like cooked mode except that input is not collected and editing is not effective; each input character is made available to a reading process as it is received by the hardware.

RAW — In this mode, 8-bit characters are passed into and out of the device with minimal hassle. Each input character is made available to a reading process as it is received by the hardware.

Other bits in the **sg_flags** word of the **sgtty** structure are:

ECHO If this bit is set, input characters are echoed to the output.

CRMOD If **RAW** is not set and this bit is set, an input CR is read as a NL. On output, NL is translated into two characters: a CR and a NL. If **ECHO** and **CRMOD** are both set, CR and NL will each echo as the two character sequence CR NL.

LCASE It is pointless to try to use an upper-case-only terminal with UNIX. The intent of this bit was try to help a user of an upper-case-only terminal. In the Fortune 32:16, this bit is ignored by the tty driver.

TANDEM When this bit is set, the system outputs a *stop* **(t_stopc)** character, normally ^S, whenever the input queue is in danger of overflowing, and a *start* **(t_startc)** character, normally ^Q, when the input queue has drained sufficiently. This mode is used when the input comes from the output of another machine rather than a person typing at a keyboard, and when the other machine understands this convention. **TANDEM** mode is unaffected by **RAW** or **CBREAK** modes.

EVENP
ODDP If the **EVENP** bit is set, input characters with even parity are accepted. If the **ODDP** bit is set, input characters with odd parity are accepted. If both bits are set or neither bit is set, input parity is ignored, and all input characters are accepted and their parity bit (most significant bit) is set to 0. Input characters which are not accepted because they are the wrong parity are discarded. Characters are ouptut with even parity if **EVENP** is set and **ODDP** is not, otherwise they are output with odd parity. In **RAW** mode, all input parity checking and output parity generation is disabled and full 8-bit characters are input and output.

Output delays and tab expansion.

In cooked and **CBREAK** mode, delays can be invoked after outputting backspaces (^H), form feeds (^L), carriage returns (^M), tabs (^I) and newlines (^J). These delays are controlled by bit fields in the **sg_flags** word of the **sgtty** structure. For compatibility with the past, when the C compiler did not support bit fields, the bit fields are not declared as such in the **sgtty** structure in <sgtty.h> but instead, masks and manifest constants are provided for the job.

The **backspace delay** capability is not implemented.

The **vertical tab delay,** which should really be called the **form feed delay,** is either on or off. If it is on, an output Form Feed is followed by a delay of 2.12 seconds.

The **carriage return delay** can be **CR0** (off), or one of three types: **CR1**, **CR2**, or **CR3**. Delay type 1 lasts about .08 seconds and is suitable for the General Electric Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the DEC VT05 and the TI 700. Delay type 3 is intended for the Concept-100 and is not really a delay. If delay type 3 is set, then enough nulls are output to guarantee that at least nine characters are ouptut on each line.

The **newline delay** can be **NL0** (off) or one of three types: **NL1**, **NL2**, or **NL3**. Delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the DEC VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

The **tab delay** can be off, **TAB0** (off) or one of three types: **TAB1**, **TAB2**, or **XTABS**. Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 2 is unimplemented and is 0. Type 3, called **XTABS**, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces to advance the column to the next multiple of 8.

Input.

UNIX expects terminals to operate in full-duplex mode. Characters may be input at any time, even while output is occurring. Input characters are buffered by the tty driver until some process reads them. Input characters will be discarded if the number of buffered input characters exceeds the maximum allowed, 255, or in extremely rare circumstances when the system's character input buffers become completely choked. In cooked or **CBREAK** mode, a BEL character is echoed when an input character is discarded. In **RAW** mode, if a character is discarded, all buffered input and output characters for that tty are discarded and no BEL is echoed.

In cooked mode, and less so in **CBREAK** mode, special behavior is invoked when certain characters are input. Some of the characters cause editing of the buffered input (cooked mode only), others control the flow of output, and others cause signals to be sent to the process group associated with the device. There are ioctls, to be discussed later, which will select the control character which invokes each kind of special processing.

In cooked mode, terminal input is processed in units of lines. A program attempting to read will normally be suspended until an entire line has been received. No matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing

data. Note, however, that it is much more efficient to read several characters at a time. In cooked mode, an input *EOT* **(t_eofc)** terminates a line of input and starts a new one in much the same way that a NL does, except that an *EOT* is discarded and not read. If an *EOT* is input at the beginning of a line, a read will return with a count of 0, indicating end-of-file.

The **t_brkc** character, normally disabled, acts like a new-line in that it terminates a line, is echoed, and is read. Typically, this character is set to ESC by programs that use it.

It is possible for a program to simulate terminal input using the **TIOCSTI** ioctl, which takes as its third argument the address of a character. The system pretends that this character was typed on the terminal, which must be the control terminal except for the super-user.

If the **LTELETEX** bit in the local modes word is set, the tty driver tries its best to treat videotex three-character sequences that describe accented alphabetic characters as if they were a unit. Such a sequence is a ^Y followed by a character in the range of 0x40 through 0x5F followed by a character in the range of 0x20 through 0xFF.

Input editing (cooked mode only).

In cooked mode, input line editing is done with the *erase* **(sg_erase)** character, normally ^H (same as the backspace key), erasing the last character typed, the *kill* **(sg_kill)** character, normally ^X, erasing the entire current input line, and the *word-erase* **(t_werasec)** character, normally ^W, erasing the last word typed. For the purposes of the *word-erase* character, a word is defined as a sequence of non-blank characters, with tabs counted as blanks. Erasing of input characters never goes back beyond the beginning of the current input line.

The *literal-next* **(t_lnextc)** character, normally ^V, can be typed preceding **any** character to prevent its special meaning. This also works in **CBREAK** mode. The *erase* and *kill* characters may also be entered literally by preceding them with '\' for compatiblity with olden days; the '\' will normally disappear upon typing the next character.

The *reprint* **(t_rprntc)** character, normally ^R, retypes all unread input. Retyping occurs automagically in cooked mode if you attempt to erase over intermingled program output. If the **LRETYPE** bit in the local modes word is set, then output is retyped on every input character which was preceded by program output.

Input echoing and redisplay options.

The driver has several options for handling the echoing of terminal input, controlled by bits in the local modes word. (The name is an

artifact from Berkeley, where the 'new tty' line discipline was originally considered a 'local extension' in the Berkeley system. It is really just another funny-named word containing a bunch of bits controlling tty operation.)

Hardcopy terminals. The **LPRTERA** bit is normally set in the local modes word when a hardcopy terminal is in use. Characters which are logically erased are then printed out backwards preceded by '\' and followed by '/' in this mode.

Crt terminals. The **LCRTBS** bit is normally set in the local modes word when a crt terminal is in use. The terminal driver then echoes the proper number of backspace (^H) characters when input is erased; in the normal case where the *erase* character is a ^H this causes the cursor of the terminal to back up to where it was before the logically erased character was typed. If the input has become fouled due to interspersed asynchronous output, the input is automagically retyped.

Erasing characters from a crt. When a crt terminal is in use, the **LCRTERA** bit may be set to cause input to be erased from the screen with a backspace-space-backspace sequence when character or word deleting sequences are used. The **LCRTKIL** bit may be set as well, causing the input to be erased in this manner with the *kill* character as well.

Echoing of control characters. If the **LCTLECH** bit is set in the local modes word, then non-printing (control) characters are normally echoed as ^X (for some X) rather than being echoed unmodified; DEL is echoed as ^?.

Output flow control.

The *stop* **(t_stopc)** character, normally ^S, will cause output to be suspended. The *start* **(t_startc)** character, normally ^S, will cause output to resume. Extra *stop* characters typed when output is already stopped have no effect. Extra *start* characters typed when output is not stopped have no effect except to turn off page mode (see below). If the **LDECCTQ** bit of the local modes word is set, stopped output will only restart when the *start* character is input; otherwise any character will restart output (page mode characters, discussed below are special here). If the *start* and *stop* characters are both set to the same character, then the function of that character toggles between stopping and starting output. These characters have no effect in **RAW** mode unless the **LOUTFLOW** bit is set in the local modes word, in which case they do have this effect, *and* they are read instead of discarded.

If the **AUTOEN** bit in the tty state word is set, output flow is controlled by the CTS input modem control status bit. I.e. output flows only when CTS is on. Also when **AUTOEN** is set, input is discarded

whenever the DCD input modem control status bit is off.

Flushing input and output.

Output to serial devices, as well as input, is buffered in the tty driver. Depending on the speed of the device, up to 496 characters may be buffered by the driver on their way to the output port. Under certain conditions, the tty driver will discard, or 'flush', buffered output.

The *flush* **(t_flushc)** character, normally ^O, sets the **LFLUSHO** bit in the local modes word, causing subsequent output to be flushed until the **LFLUSHO** bit is cleared by a program or more input is typed. This character has effect in both cooked and **CBREAK** modes and causes pending input to be retyped if there is any pending input.

The **TIOCFLUSH** ioctls can be used to flush the characters in the input and/or output queues depending on the third argument. The **TIOCOUTQ** ioctl can be used to determine the number of characters still in the output queue.

Page mode.

Very fast terminal screen output, such as one finds on the Fortune 32:16 console or on terminals running at 19200 baud, can go by so fast that you can't read it. **Page mode** is intended to solve this problem. When page mode is on, the tty driver keeps a count of the number of lines it has output since the last input character. When this number reaches the number of lines on the page, output stops, the <<STOPPED>>) prompt is printed, and the terminal driver waits for input. The *nxpage* **(t_pagec)** character, normally ^F, enables another page of output, the *nxhalf* **(t_halfc)** character, normally ^B, enables another half page of output, and the *nxline* **(t_linec)** character, normally ^E, enables another line of output. If you type these characters ahead, the amount of output enabled will accumulate.

If the **LPAGE_EN** bit in the local modes word is set, page mode is enabled, which means that it may or may not be turned on. Then if the **LPAGE_ON** bit in the local modes word is set, page mode is turned on. When the **LPAGE_EN** bit is set, **LPAGE_ON** can be set by typing any of the three page mode characters mentioned above. The *start* character turns page mode off, but does not disable it. The *stop* character does not turn page mode on or off, but it does stop output, and then the page mode characters can be used to get more output.

The **TIOCSPAGE** and **TIOCGPAGE** ioctls are used to change the page length and to read it, respectively. Page mode is never enabled if the page length is 0.

Input characters that generate signals.

There are several characters that generate signals in cooked and **CBREAK** mode; such signals are sent to all processes with the same process group ID as the tty. In addition, when input, these characters also flush pending input and output. This same flushing action can be effected with the **TIOCFLUSH** ioctl.

The *interrupt* **(t_intrc)** character, normally ^? (same as ASCII DEL), sends a **SIGINT** signal. This is the normal way to blow away a process from the terminal. In addition, many interactive programs catch this signal and revert to command mode when they catch it. In cooked and **CBREAK** mode, a received break is converted by the driver into an *interrupt* character, and acts just like one. In **RAW** mode, a received break is read as a NUL.

The *quit* **(t_quitc)** character, normally ^\, sends a **SIGQUIT** signal. This signal is used to cause a foreground program to terminate and produce a core image if possible, in a file called **core** in the current directory.

Output.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. (As noted above, input characters are normally echoed by putting them in the output queue as they arrive.) When a process writes characters more rapidly than they can be output through the output hardware, it will be suspended when its output queue exceeds a limit which is automatically selected to be higher for higher output speeds, with a maximum of 496 characters. When the queue has drained down to a threshold, the program is resumed.

If the **LNOEOTOUT** bit in the local modes word is set, the EOT character is not transmitted in cooked or **CBREAK** mode. This is to prevent terminals that respond to it from hanging up; programs using **RAW** or **CBREAK** mode should be careful.

If the **LLITOUT** bit in the local modes word is set, output translations are suppressed in cooked and **CBREAK** mode, and output acts much like **RAW** mode.

Modem control signals

NOTE: Important information on modem control signal information is presented in sio(4). The tty driver knows about the output modem control signals DTR and RTS, and the input modem control signals DCD, DSR, and CTS.

Output modem control signals

Whenever a tty is opened, whether it is the first open or not, the DTR and RTS output modem control signals are both raised. If the

LNOOPENDTR bit of the local modes word is set, DTR will not be raised on open. Similarly, if the **LNOOPENRTS** bit of the local modes word is set, RTS will not be raised on open. If the **HUPCLS** bit of the tty state word is set, DTR and RTS will be lowered on last close. Setting the input speed to 0 causes DTR and RTS to be lowered immediately. The **TIOCSDTR** ioctl raises DTR, and the **TIOCCDTR** ioctl lowers it. The **TIOCSRTS** ioctl raises RTS, and the **TIOCCRTS** ioctl lowers it.

Input modem control signals

If the **NOMDMINTS** bit of the local modes word is set, then the input modem control signals are totally ignored, an input break will act as if it were an ordinary null character, and the discussion of other features relating to input modem status signals and input breaks elsewhere in this section becomes moot. (This bit should only be used if the equipment connected to the port generates noise on one of the input modem control signals, and it is inconvenient to disconnect the offending signal(s), and the status of the other unoffending signals is unimportant. Specifically, some printers generate noise on pin 25. Noise on an input modem control signal can very seriously degrade system performance.)

The current state of the three modem control inputs is available in the tty state word as the **DCD_STATUS**, **CTS_STATUS**, and **DSR_STATUS** bits. Three more bits in the tty state word, **DCD_CHANGE**, **CTS_CHANGE**, and **DSR_CHANGE**, are set to indicate a change since first open or since the most recent **TIOCGETSTATE** ioctl on the device. These bits are useful for detecting transitions that are too quick to be caught by looking at the status bits.

A process can issue a **TIOCSETSIG** ioctl to arrange for a specified signal to be sent to the tty's process group on each transition of selected input modem control bits. The selection of bits is made with a bitmask made up of the **DCD_STATUS**, **CTS_STATUS**, and **DSR_STATUS** bits. The **TIOCGETSIG** ioctl reports the currently selected modem status change signal number and the selected bits. On last close, the bitmask of selected bits and the selected modem status change signal are both set to 0.

Carrier Detect.

The tty driver maintains a bit in the tty state word named **CARR_ON**, for 'carrier on'. The name 'connected' would be more appropriate to its true function, since this bit can often be set when the DCD input is not. Depending on the settings of the **RESPECTCARR**, **IWTCARR**, and **OWTCARR** bits in the tty state word, **CARR_ON** may go up and down with the DCD input signal or it may be independent of it.

If the **RESPECTCARR** bit in the tty state word is not set, then the DCD input signal has no effect on **CARR_ON**, and the **IWTCARR** and **OWTCARR** bits in the tty state word are ignored. In this case, the **CARR_ON** bit is set on first open, and cleared on last close. Both input and output function normally when the **CARR_ON** bit is on. If the **RESPECTCARR** bit in the tty state is set, then **CARR_ON** starts out high on first open, but will then go down and up with the DCD input signal. If DCD makes **CARR_ON** drop while the tty is open, a **SIGHUP** signal is sent to all the processes in the tty's process group, and all current and subsequent reads and writes terminate prematurely, i.e. with a returned count of 0 (reads) or a smaller count than requested (writes). (Pending reads and writes which are interrupted by the SIGHUP signal will return -1.) The SIGHUP is suppressed if the **LNOHANG** bit in the local modes word is set.

A normal tty device, such as /dev/tty01, is considered an 'incoming' device. For each incoming device there can be a corresponding 'outgoing' device, which is the same except that its minor device number is 128 greater than its corresponding incoming device. Logins use incoming devices; cu(1) and uucico(8) use outgoing devices. For example, if /dev/tty01, which is an incoming device, is device major:minor 1:1, then the corresponding outgoing device, which would probably be named /dev/cul0, would be device major:minor 1:129. If the **RESPECTCARR** and **IWTCARR** bits in the tty state word are both set on an incoming device, then if DCD is low, opens will block until DCD comes up, at which point **CARR_ON** is set and sleeping open calls will return. Similarly, the **OWTCARR** bit can be used to cause opens on an outgoing device to wait for carrier.

If a process, such as getty, is waiting for carrier on an incoming device, the corresponding outgoing device can be used, and the process sleeping on the incoming device will continue to sleep until the outgoing device is closed, at which time the incoming process will continue to wait for carrier. If an incoming tty is open, then opens on the corresponding outgoing tty will fail with errno set to EBUSY.

If the **RESPECTCARR** bit in the tty state word is set, the **TIOCWAITCARR** ioctl can be used to explicitly wait until carrier is up. This ioctl returns immediately if carrier is already up at the time of the call.

Breaks.

The **TIOCSBRK** ioctl will set the 'transmit break' bit in the hardware interface causing the transmitted line to go to the mark condition until this condition is reversed by a **TIOCCBRK** ioctl (usually after a delay with *sleep*(3)). The **TCSBRK** ioctl will cause a 250 millisecond break to be transmitted.

An input break in **RAW** mode is read as a NUL character. In cooked or **CBREAK** mode it is read as if it were the *interrupt* character.

More detail on the ioctl commands.

Programs using tty ioctls will include **<sys/types.h>** and **<sgtty.h>** which in turn includes **<sys/ioctl.h>**. Some ioctls require including **<sys/tty.h>**.

```
#include <sgtty.h>

struct sgtty sgtty;

ioctl(filedes, TIOCGETP, &sgtty);
```

This ioctl reads the **sgtty** structure. The **TIOCSETN** ioctl writes it, and the **TIOCSETP** ioctl writes it after waiting until output is quiescent, the flushes input and output.

The **sgtty** structure, and the defines for the bits used in that structure, are defined in *<sgtty.h>*:

```
struct sgttyb {
        char    sg_ispeed;
        char    sg_ospeed;
        char    sg_erase;
        char    sg_kill;
        short   sg_flags;
};
```

The **sg_ispeed** and **sg_ospeed** fields describe the input and output speeds of the device according to the following table. Symbolic values in the table are as defined in *<sgtty.h>*:

B0	0	(hang up dataphone)
B50	1	50 baud
B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A (19200 baud)
EXTB	15	External B

The **EXTA** baud rate is 19200 baud on the Fortune 32:16. Code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program. Half-duplex operation, (such as is used on the Bell 202 dataset is not supported.

The bits in the **sg_flags** field are as follows:

ALLDELAY	0177400	Delay algorithm selection
BSDELAY	0100000	Select backspace delays (not implemented):
BS0	0	
BS1	0100000	
VTDELAY	0040000	Select form-feed and vertical-tab delays:
FF0	0	
FF1	0100000	
CRDELAY	0030000	Select carriage-return delays:
CR0	0	
CR1	0010000	
CR2	0020000	
CR3	0030000	
TBDELAY	0006000	Select tab delays:
TAB0	0	
TAB1	0001000	
TAB2	0004000	
XTABS	0006000	
NLDELAY	0001400	Select new-line delays:
NL0	0	
NL1	0000400	
NL2	0001000	
NL3	0001400	
EVENP	0000200	Even parity allowed on input (most terminals)
ODDP	0000100	Odd parity allowed on input
RAW	0000040	Raw mode: wake up on all characters, 8-bit interface
CRMOD	0000020	Map input CR into LF, output LF to CR LF
ECHO	0000010	Echo (full duplex)
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Return each character as soon as typed
TANDEM	0000001	Input flow control

```
include <sgtty.h>

struct tchars tchars;

ioctl(filedes, TIOCGETC, &tchars);
```

This ioctl reads the **tchars** structure, and the **TIOCSETC** ioctl writes it. This structure contains some of the characters that are special on

input. The **tchars** structure is defined in <*sys/ioctl.h*>:

```
struct tchars {
        char    t_intrc;        /* interrupt */
        char    t_quitc;        /* quit */
        char    t_startc;       /* start output */
        char    t_stopc;        /* stop output */
        char    t_eofc;         /* end-of-file */
        char    t_brkc;         /* input delimiter (like nl) */
```

```
include <sgtty.h>

struct ltchars ltchars;

ioctl(filedes, TIOCGLTC, &ltchars);
```

This ioctl reads the **ltchars** structure, and the **TIOCSLTC** ioctl writes it. This structure contains some of the characters that are special on input. The **ltchars** structure is defined in <*sys/ioctl.h*>:

```
struct ltchars {
        char    t_suspc;        /* stop process signal */
        char    t_dstopc;       /* delayed stop process signal */
        char    t_rprntc;       /* reprint line */
        char    t_flushc;       /* flush output (toggles) */
        char    t_werasec;      /* word erase */
        char    t_lnextc;       /* literal next character */
};
```

The **t_suspc** and **t_dstopc** characters are not implemented.

```
include <sgtty.h>

struct ptchars ptchars;

ioctl(filedes, TIOCGPTC, &ptchars);
```

This ioctl reads the **ltchars** structure, and the **TIOCSPTC** ioctl writes it. This structure contains the characters that are special to page mode. The **ptchars** structure is defined in <*sys/ioctl.h*>:

```
struct ptchars {
        char    t_pagec;        /* show next page */
        char    t_linec;        /* show next line */
        char    t_halfc;        /* show half next page */
        char    t_clearc;       /* clear screen and show next
                                   page */
};
```

The **t_clearc** character is not implemented.

Characters in the **sgtty**, **tchars**, **ltchars**, and **ptchars** structures should be unique, except that the **t_stopc** and **t_startc** can be the same for a toggling effect. Setting one of these characters to 0xFF

disables that function.

unsigned long localmodes;

ioctl(filedes, TIOCLGET, &localmodes);

This ioctl reads the **local modes word**. The **TIOCLSET** ioctl writes it, the **TIOCLBIS** ioctl ORs *localmodes* into it, and the **TIOCLBIC** ioctl ANDs it with the complement of *localmodes*.

The bits of the local modes word are:

LCRTBS	0x000001	Backspace on erase rather than echoing erase
LPRTERA	0x000002	Printing terminal erase mode
LCRTERA	0x000004	Erase character echoes as backspace-space-backspace
LTILDE	0x000008	Convert ~ to ` on output (for Hazeltine terminals)
LMDMBUF	0x000010	Stop/start output when carrier drops
LLITOUT	0x000020	Suppress output translations
LTOSTOP	0x000040	Send SIGTTOU for background output
LFLUSHO	0x000080	Output is being flushed
LNOHANG	0x000100	Don't send hangup when carrier drops
LETXACK	0x000200	Diablo style buffer hacking (unimplemented)
LCRTKIL	0x000400	BS-space-BS erase entire line on line kill
LINTRUP	0x000800	Generate interrupt SIGTINT when input ready to read
LCTLECH	0x001000	Echo input control chars as ^X, delete as ^?
LDECCTQ	0x004000	Only ^Q restarts output after ^S, like DEC systems
LPAGE_EN	0x008000	Enable page mode, does not turn it on.
LPAGE_ON	0x010000	Turn page mode on.
LTELETEX	0x020000	teletex multi-char sequences
LNOEOTOUT	0x040000	don't output EOT's
LOUTFLOW	0x080000	flow ctrl in raw mode & read the chars
LRETYPE	0x100000	Retype whenever echoed data is messed up
LNOOPENDTR	0x200000	Do not raise DTR on open
LNOOPENRTS	0x400000	Do not raise RTS on open

The **LTILDE**, **LTOSTOP**, **LINTRUP**, and **LMDMBUF** bits are

unimplemented.

unsigned long state;

ioctl(filedes, TIOCGETSTATE, &state);

This ioctl reads the **tty state word**, and the **TIOCSETSTATE** ioctl writes certain bits into it.

The interesting bits of the tty state word are:

CARR_ON	0x0000010	Connected
BUSY	0x0000020	Output in progress
XCLUDE	0x0000080	Exclusive-use flag against open
TTSTOP	0x0000100	Output stopped by ctl-s
HUPCLS	0x0000200	Hang up upon last close
TBLOCK	0x0000400	Tandem queue blocked
DCD_STATUS	0x0001000	DCD status
CTS_STATUS	0x0002000	CTS status
DSR_STATUS	0x0004000	DSR status (RI on the sio)
RTS_ON	0x0008000	Software copy of rts on
DTR_ON	0x0100000	Software copy of dtr on
RESPECTCARR	0x0200000	Respect the carrier
IWTCARR	0x0400000	Wait for carrier on incoming open
OWTCARR	0x0800000	Wait for carrier on outgoing open
AUTOEN	0x1000000	DCD & CTS enable receiver and xmitter
DCD_CHANGE	0x2000000	DCD status changed
CTS_CHANGE	0x4000000	CTS status changed
DSR_CHANGE	0x8000000	DSR status changed (RI on the sio)

The **TIOCSETSTATE** ioctl loads only the **XCLUDE**, **HUPCLS**, **IWTCARR**, **OWTCARR**, **RESPECTCARR**, and **AUTOEN** bits. The others are read-only. The **TIOCGETSTATE** ioctl clears the **DCD_CHANGE**, **CTS_CHANGE**, and **DSR_CHANGE** bits after reading them.

include <sgtty.h>

struct mdmsig mdmsig;

ioctl(filedes, TIOCSETSIG, &mdmsig);

This ioctl writes the modem signaling information and the **TIOCGETSIG** reads it. This information is transmitted with the **mdmsig** structure found in *<sgtty.h>*:

```
struct mdmsig {
        unsigned long ms_mask;/* mask of status bits to
                                 signal on */
        unsigned char ms_signo;/* which signal to send */
};
```

A number of other *ioctl*(2) calls apply to ttys, and have the general form:

```
#include <sgtty.h>

ioctl(fildes, command, arg)
item *arg;
```

where *item* would be an unsigned long, or a short, or an int, as appropriate for the ioctl command. Unless otherwise specified, ioctl commands that don't take an argument should be passed a NULL pointer for the *arg*.

With the following commands the *arg* is ignored, although 0 is recommended.

TIOCEXCL Set exclusive-use mode: no further opens are permitted until last close.

TIOCNXCL Turn off exclusive-use mode.

TIOCHPCL On last close, hang up, i.e. drop the DTR and RTS outputs. This is useful when the line is associated with an ACU used to place outgoing calls.

TIOCSAVEMODES
Save the current characteristics as the defaults for the tty on subsequent first opens. This is what is saved: **sgtty** structure, **tchars** structure, **ltchars** structure, **ptchars** structure, local modes word, line discipline, page length, and the **DTR_ON**, **RTS_ON**, **DCD_STATUS**, **CTS_STATUS**, and **DSR_STATUS** bits of the tty state word.

TIOCWAITCARR
Wait until DCD is up if the **RESPECTCARR** bit is set in the tty state word, otherwise return immediately.

In the remaining calls, where arguments are required they are described; *arg* should otherwise be given as the NULL pointer (0).

TIOCFLUSH
If *arg* is 0, all characters waiting in input or output queues are flushed. If *arg* is non-0, it is taken as a pointer to an int. The input queue is flushed if the 1 bit of the int is set, and the output queue is flushed if the 2 bit of the int is set.

TIOCSTI
the argument is the address of a character which the system pretends was typed on the terminal.

TIOCSBRK
the break transmit bit is set.

TIOCCBRK
the break transmit bit is cleared.

TIOCSDTR
DTR output is set.

TIOCCDTR
DTR output is cleared.

TIOCSRTS
RTS output is set.

TIOCCRTS
RTS output is cleared.

TIOCGPGRP
arg is the address of a short into which is placed the process group number of the tty.

TIOCSPGRP
arg is a short (typically a process id) which becomes the process group for the tty.

TIOCOUTQ
reads the number of characters awaiting output into the int whose address is *arg*.

TIOCSPAGE
loads the page length from the int whose address is *arg*.

TIOCGPAGE
reads the page lenth into the int whose address is *arg*.

TCSBRK
Wait for output to be quiescent, and if *arg* is the NULL pointer, output a 250 millisecond break.

FIONREAD
returns in the long integer whose address is *arg* the number of immediately readable characters from the argument unit. This works for files, pipes, and ttys.

FILES

/dev/tty

/dev/tty*

/dev/console

SEE ALSO

csh(1), stty(1),
ioctl(2), signal(2), stty(2),
getty(8), init(8).

NAME

devtype - serial line configuration file

DESCRIPTION

Devtype contains for each serial line the following information:

unix device name

internal name (eg. As described in termcap, printcap ..)

class (P=printer, T=terminal, C=communications device)

number - number of device in it's class

baud rate

external name (eg. location)

service notification port

Status bit 1 = enabled, 0 = disabled

This is an ASCII file. Each field within each entry is separated from the next by a tab. Each entry is separated from the next by a newline.

This file resides in directory /etc.

FILES

/etc/devtype

SEE ALSO

dtinit(1).

NAME

dir - format of directories

SYNOPSIS

#include <sys/dir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user can write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry see, *filsys*(5). The structure of a directory entry as given in the include file is:

```
#ifndef DIRSIZ
#define DIRSIZ  14
#endif
struct   direct
{
        ino_t    d_ino;
        char     d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. Therefore in both cases '..' has the same meaning as '.'.

SEE ALSO

filsys(5).

NAME

disk - Special I/O commands for disk devices

SYNOPSIS

ioctl(fildes, request, addr)
devctl(b_or_c, dev, request, addr)

DESCRIPTION

Special disk maintenance operations are done with the *ioctl* and *devctl* calls. Some special operations may only be done when the device has no outstanding *open(2)* calls, necessitating the use of *devctl*, and some can be done either with devctl or with ioctl.

The different calls use and return data in addr pointers. The various calls and their arguments.

devctl(b_or_c,dev,IOCRCONFB,struct diskconf *)
ioctl(fildes,IOCRCONFB,struct diskconf *)
Read the configuration for this disk into struct diskconf.

devctl(b_or_c,dev,IOCWCONFB,struct diskconf *)
Change the configuration block to that in struct diskconf. There can be no outstanding requests on the drive, and the conf block must be correct.

devctl(b_or_c,dev,IOCFMT,struct diskconf *)
Format the drive. There may be no outstanding requests. The conf block must be correct.

devctl(b_or_c,dev,IOCFMTTRK,struct fmttrk *)
Format the track on the drive. Same as above.

devctl(b_or_c,dev,IOCCHMED,unused)
Force driver to assume that media in drive has changed. There may be no outstanding requests.

devctl(b_or_c,dev,IOCAUTO,struct devauto *)
ioctl(fildes,IOCAUTO,struct devauto *)
Place standard name and mode of device, if it exists, into struct devauto.

devctl(b_or_c,dev,IOCSAFE,unused)
Move disk heads to a safe place, in case of power outage, global or local earthquake, or other acts of God.

devctl(b_or_c,dev,IOCSEEK,cylinder)
ioctl(fildes,IOCSEEK,cylinder)
Seek disk heads to given cylinder.

devctl(b_or_c,dev,IOCRESET,unused)
Force driver to rebuild its concept of the state of the drive. No outstanding requests.

devctl(b_or_c,dev,IOCDRVSTAT,struct diskstat *)
ioctl(fildes,IOCDRVSTAT,struct diskstat *)

Report current state of drive: its head position, any error conditions, and indicate any outstanding accesses.

FILES

/usr/include/sys/ioctl.h

/usr/include/sys/diskconf.h

SEE ALSO

devctl(2), ioctl(2), diskconf(5), floppy(5), wd(5).

DIAGNOSTICS

Devctl and *ioctl* return -1 on error.

LIMITATIONS

It is not guaranteed that all of these commands are implemented in every driver.

NAME

Diskconf

DESCRIPTION

The *diskconf* structure, also known as the *disk configuration block*, occupies the first block of every disk used by the Fortune 32:16. It contains information describing physical characteristics of the disk, the disk partition layout, bad block information, etc. used by the particular disk driver. Mkconf(1), rdconf(1), and format(1) manipulate the config block.

PHYSICAL CHARACTERISTICS

The canonical disk described by the config block consists of one or more platters each of which is a series of *dc_cyls* concentric circles of information-bearing magnetic media. The disk also has *dc_head* read/write heads, each of which has its own platter. The heads can move from outermost cylinder to the innermost. They are usually attached together and must move in tandem. Each individual band is called a track, and there are *dc_head* * *dc_cyls* tracks per disk. Each track is subdivided into *dc_sectrk sectors*. Each sector holds *dc_blksiz* bytes. Each sector is called a block, and the standard address of any sector on the disk is its block number. There are *dc_head* * *dc_cyls* * *dc_sectrk* blocks on a disk. Block numbers increase by sector, then by head, then by cylinder. The UNIX system has a notion of two kinds of blocks: *logical* blocks and *physical* blocks. Logical blocks are 1024 bytes long. The 32:16 built-in floppy disk operates with 1024-byte (or 1K) physical blocks. The Winchester disks operate with 512-byte (or 1/2K) physical blocks. All block numbers in the config block are in physical blocks.

The *dc_wrtred* and *dc_wrtpre* fields stand for *write reduce* and *write*precompensation. These affect the way data is encoded on the surface of the disk via magnetic blips. The *dc_ro_flags* contain one bit per partition. If that bit is on, the partition can be accessed but not changed.

The disk can be divided into *NPART* (defined in /usr/include/sys/diskconf.h) contiguous areas, called *partitions*. Partition X is described by the first block and number of blocks, i.e., it starts at block *dc_sizes[X].dc_poffset* and continues for *dc_sizes[X].dc_psize* blocks. The disk can also contain up to *NPART* level 1 boot program areas. These are indicated in a separate table, similar to the partition table. A boot area must be wholly within a partition, because only the partitions are accessed by the drivers.

When a disk error occurs, the offending block(s) is/are replaced by the *sparing* mechanism. When a block is decided to be unusable, any reference to it is redirected to its replacement block. Spare blocks are pulled from the first cylinder. Thus, data should never be stored on the first cylinder. The field *dc_nbad* and the array *dc_bad[]*

constitute the *bad block map*. The construction of the bad block map differs between devices.

A config block must follow certain rules: *dc_magic* must contain *CONFMAGIC* (defined in <sys/diskconf.h>), *dc_sysid* and *dc__sysid* and also *dc_vid* and *dc__vid* must contain the bitwise negation of each other, respectively. Also, any driver can impose additional restrictions, usually to disallow impossible drive characteristics (i.e. 5000 disk heads on a floppy).

For performance reasons, partition boundaries should be on cylinders. A typical layout for a Seagate 506 disk on a one-hard-disk system:

(4 heads, 16 sectors/track, 153 cylinders, 4.9 megabytes)

	Size, Offset	
Partition 0	192,0	first 3 cylinders
Partition 1	1408,192	appx. 700K of swap space
Partition 2	8192,1590	rest of disk
Boot area 0	80,64	40K, size of /sa/boot

A floppy-UNIX layout would be:

(2 heads, 5 sectors/track, 80 cylinders, 800 kilobytes)

	Size, Offset	
Partition 0	50,0	first 5 cylinders
Partition 1	250,50	250K of swap space
Partition 2	500,300	rest of disk
Boot area 0	40,10	40K

A back-up floppy layout would be:

	Size, Offset	
Partition 0	10,0	first 1 cylinder
Partition 1	0,0	empty
Partition 2	790,50	rest of disk

No boot image.

FILES

/usr/include/sys/diskconf.h

SEE ALSO

format(8), mkconf(8), rdconf(8).

NAME

Disktab

DESCRIPTION

The *disktab* file is a human readable disk configuration data base. Command *Dskselect(8)* reads it to create a configuration block for some selected disk type. Each record in the file stores information about one configuration block. Fields are separated by ":". Each record is stored on one line. Each field in the first record contains name of the corresponding column. Columns can be ordered arbitrarily. Column name is the key column and all names must be unique. The name field is the Fortune System assigned name of a disk. A name is one letter prefix (unique for each manufacturer) and a 1-2 digit suffix (approximate storage capacity in Mega bytes). Other fields describe the sector size, number of sectors/track, number of tracks, number of heads, write pre-com cylinder, write reduce current cylinder, media type, interlace factor, default disk name, default partitions, default boots, manufacturer's name, model number or name, etc. See diskconf(5) for more details.

The format of the disktab file can change in future. So this file should be accessed only through dskselect(8).

FILES

/etc/disktab

SEE ALSO

diskconf(5), dskselect(8).

LIMITATIONS

The first line is so long that you will not be able to edit this file with vi or ed.

NAME

filsys, flblk, ino - format of file system volume

SYNOPSIS

#include <sys/types.h>
#include <sys/flbk.h>
#include <sys/filsys.h>
#include <sys/ino.h>

DESCRIPTION

Every file system storage volume (e.g. /dev/hd[0-3][2-7], /dev/fd[0-3][2-7]) has a common format for certain vital information. Every such volume is divided into a certain number of 1024-byte blocks.

Block 0 is unused. Block 1 is the *super block.* The layout of the super block as defined by the include file *<sys/filsys.h>* is:

```
/*
 * Structure of the super-block
 */
struct    filsys {
          unsigned short s_isize; /* size in blocks of i-list */
          daddr_t s_fsize;        /* size in blocks of entire volume */
          short    s_nfree;       /* number of addresses in s_free */
          daddr_t s_free[NICFREE]; /* free block list */
          short    s_ninode;      /* number of i-nodes in s_inode */
          ino_t    s_inode[NICINOD]; /* free i-node list */
          char     s_flock;       /* lock during free list manipulation */
          char     s_ilock;       /* lock during i-list manipulation */
          char     s_fmod;        /* super block modified flag */
          char     s_ronly;       /* mounted read-only flag */
          time_t   s_time;        /* last super block update */
/* remainder not maintained by this version of the system */
          daddr_t s_tfree;        /* total free blocks*/
          ino_t    s_tinode;      /* total free inodes */
          short    s_m;           /* interleave factor */
          short    s_n;           /* " " */
          char     s_fname[6];    /* file system name */
          char     s_fpack[6];    /* file system pack name */
};
```

S_isize is the address of the first block after the i-list, which starts just after the super-block, in block 2. Thus the i-list is *s_isize*-2 blocks long. *S_fsize* is the address of the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block addresses; if an 'impossible' block address is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The *s_free* array contains, in *s_free[1], ... , s_free[s_nfree-1]*, up to NICFREE free block numbers. NICFREE is a configuration constant. *S_free[0]* is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in the include file <*sys/fblk.h*> is:

```
struct fblk
{
        int     df_nfree;
        daddr_t df_free[NICFREE];
};
```

The fields *df_nfree* and *df_free* in a free block are used exactly like *s_nfree* and *s_free* in the super block. To allocate a block: decrement *s_nfree*, and the new block number is *s_free[s_nfree]*. If the new block address is 0, there are no blocks left, so give an error. If *s_nfree* became 0, read the new block into *s_nfree* and *s_free*. To free a block, check if *s_nfree* is NICFREE; if so, copy *s_nfree* and the *s_free* array into it, write it out, and set *s_nfree* to 0. In any event set *s_free[s_nfree]* to the freed block's address and increment *s_nfree*.

S_ninode is the number of free i-numbers in the *s_inode* array. To allocate an i-node: if *s_ninode* is greater than 0, decrement it and return *s_inode[s_ninode]*. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the *s_inode* array, then try again. To free an i-node, provided *s_ninode* is less than NICINODE, place its number into *s_inode[s_ninode]* and increment *s_ninode*. If *s_ninode* is already NICINODE, don't bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

The fields *s_lasti* and *s_nbehind* are used to avoid searching the inode list from the beginning each time the system runs out of inodes. *S_lasti* gives the base of the block of inodes last searched on the filesystem when inodes ran out, and *s_nbehind* gives the number of inodes, whose numbers were less than *s_lasti* when they were freed with *s_ninode* already NICINODE. Thus *s_ninode* is the number of free inodes before *s_lasti*. The system will search forward for free inodes from *s_lasti* for more inodes unless *s_nbehind* is sufficiently large, in which case it will search the file system inode list from the beginning. This mechanism serves to avoid n**2 behavior in allocating inodes.

S_flock and *s_ilock* are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of *s_fmod* on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to

the disk during the next periodic update of file system information. *S_ronly* is a write-protection indicator; its disk value is also immaterial.

S_time is the last time the super-block of the file system was changed. During a reboot, *s_time* of the super-block for the root file system is used to set the system's idea of the time.

The fields *s_tfree, s_tinode, s_fname* and *s_fpack* are not currently maintained.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-nodes are 64 bytes long, so 16 of them fit into a block. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node as given in the include file <*sys/ino.h*> is:

```
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode
{
        unsigned short di_mode; /* mode and type of file */
        short   di_nlink;       /* number of links to file */
        short   di_uid;         /* owner's user id */
        short   di_gid;         /* owner's group id */
        off_t   di_size;        /* number of bytes in file */
        char    di_addr[40]; /* disk block addresses */
        time_t  di_atime;       /* time last accessed */
        time_t  di_mtime;       /* time last modified */
        time_t  di_ctime;       /* time created */
};
/*
 * the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

Di_mode tells the kind of file; it is encoded identically to the *st_mode field of stat*(2). *Di_nlink* is the number of directory entries (links) that refer to this i-node. *Di_uid* and *di_gid* are the owner's user and group IDs. *Size* is the number of bytes in the file. *Di_atime* and *di_mtime* are the times of last access and modification of the file contents (read, write or create) (see *times*(2)); *Di_ctime* records the time of last modification to the inode or to the file, and is used to determine whether it should be dumped.

Special files are recognized by their modes and not by i-number. A block-type special file is one which can potentially be mounted as a

file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the *di_addr* field is occupied by the device code (see *types*(5)). The device codes of block and character special files overlap.

Disk addresses of plain files and directories are kept in the array *di_addr* packed into 3 bytes each. The first 10 addresses specify device blocks directly. The last 3 addresses are singly, doubly, and triply indirect and point to blocks of 256 block pointers. Pointers in indirect blocks have the type *daddr_t* (see *types*(5)).

For block *b* in a file to exist, it is not necessary that all blocks less than *b* exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

SEE ALSO

stat(2), dir(5), types(5) dcheck(8), fsck(8), icheck(8), mount(8).

NAME

fstab - static information about the file systems

SYNOPSIS

#include <fstab.h>

DESCRIPTION

The file */etc/fstab* contains descriptive information about the various file systems. */etc/fstab* is only *read* by programs, and not written; it is the duty of the system administrator to properly create and maintain this file.

These programs use */etc/fstab: dump, mount, umount, swapon, fsck* and *df.* The order of records in */etc/fstab* is important, for *fsck, mount,* and *umount* sequentially iterate through */etc/fstab* doing their thing.

The special file name is the **block** special file name, and not the character special file name. If a program needs the character special file name, the program must create it by appending an "r" after the last "/" in the special file name.

If *fs_type* is "rw" or "ro" then the file system whose name is given in the *fs_file* field is normally mounted read-write or read-only on the specified special file. The *fs_freq* field is used for these file systems by the *dump*(8) command to determine which file systems need to be dumped. The *fs_passno* field is used by the *fsck*(8) program to determine the order in which file system checks are done at reboot time. The root file system should be specified with a *fs_passno* of 1, and other file systems should have larger numbers. File systems within a drive should have distinct numbers, but file systems on different drives can be checked on the same pass to utilize parallelism available in the hardware.

Fs_type can be specified as "xx" to cause an entry to be ignored. This is useful to show disk partitions which are currently not used but will be used later.

```
#define FSTAB       "/etc/fstab"
#define FSNMLG      16

#define FSTABFMT    "%16s:%16s:%2s:%d:%d\n"
#define FSTABARG(p)         (p)->fs_spec, (p)->fs_file, \
            (p)->fs_type, &(p)->fs_freq, &(p)->fs_passno
#define FSTABNARGS          5

#define FSTAB_RW    "rw"    /* read write device */
#define FSTAB_RO    "ro"    /* read only device */
#define FSTAB_SW    "sw"    /* swap device */
#define FSTAB_XX    "xx"    /* ignore totally */

struct fstab {
```

```
        char    fs_spec[FSNMLG]; /* block special device name */
        char    fs_file[FSNMLG]; /* file system path prefix */
        char    fs_type[3];  /* rw,ro,sw or xx */
        int     fs_freq;     /* dump frequency, in days */
        int     fs_passno;   /* pass number on parallel dump */
};
```

The proper way to read records from */etc/fstab* is to use the routines getfsent(), getfsspec() or getfsfile().

FILES

/etc/fstab

SEE ALSO

getfsent(3)

NAME

group - group file

DESCRIPTION

Group contains for each group the following information:

group name
encrypted password
numerical group ID
a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons. Each group is separated from the next by a carriage return. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

newgrp(1), passwd(1), pwac(1), crypt(3), passwd(5).

LIMITATIONS

The *passwd*(1) command won't change the passwords. The group concept is nowhere near sophisticated enough for 'real' security systems and not really very handy for small systems.

NAME

motd - message of the day

DESCRIPTION

Motd is a file that resides in directory */etc* and is the "message of the day" that is printed out when users login.

SEE ALSO

login(1)

NAME

mtab - mounted file system table

DESCRIPTION

Mtab resides in directory */etc* and contains a table of devices mounted by the *mount* command. *Umount* removes entries.

Each entry is 64 bytes long; the first 32 are the null-padded name of the place where the special file is mounted; the second 32 are the null-padded name of the special file. The special file has all its directories stripped away; that is, everything through the last '/' is thrown away.

This table is present only so people can look at it. It does not matter to *mount* if there are duplicate entries or to *umount* if a name cannot be found.

FILES

/etc/mtab

SEE ALSO

mount(8).

NAME

passwd - password file

DESCRIPTION

Passwd contains for each user the following information:

name (login name, contains no upper case)
encrypted password
numerical user ID
numerical group ID
comment
initial working directory
program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, then */bin/sh* is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

Appropriate precautions must be taken to lock the file against changes if it is to be edited with a text editor; *vipw*(8) does the necessary locking.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(1), pwac(1), crypt(3), getpwent(3), group(5).

LIMITATIONS

User information (name, office, etc.) should be stored elsewhere.

NAME

printcap - printer capability data base

SYNOPSIS

/etc/printcap

DESCRIPTION

Printcap is a data base describing printers. They are described by giving a set of capabilities which they have, and by describing how operations are performed.

Entries in printcap consist of a number of ':' separated fields. The first entry for each printer gives the names which are known for the printer, separated by '|' characters.

TYPES OF PRINTCAP DESCRIPTORS

:xx: This is a switch. It is either present or absent.

:xx#<num>:

<num> must be a decimal number. This is frequently used as the number of an algorithm applicable to the printer being described. Example:

:ir#1:

Use initialization routine 1.

:xx=<string>:

<string> is any string of ascii characters other than ':'. It may also include \E for escape, \nnn for an octal code, \n, etc. Example:

:wh=/etc/diab.whl:

Printwheel descriptor file is /etc/diab.whl

:xx=<expr>:

<expr> is a string as above. It is evaluated as a sequence of numbers and operators. The legal operators are:

+, -, *, /, //, >>, <<, |, ^, &, ~

The symbol // is remainder (% in C), and the rest have their normal C meanings. There is no precedence, and evaluation is strictly left to right. Numbers may be decimal, octal (begins with 0), or hex (begins with 0x). The letter 'I' is interpreted as a number whose value is an inch. Example:

:bv=I/48*125:

Maximum vertical increment is 125/48ths of an inch.

:xx=<format>:

<format> is a string as above. If the string is to be used to output a value, it may contain format specifications. A format specification begins with a '%'. Thus a real '%' must be represented in the string as '%%'. The '%' is followed by an expr

as above, with the addition that 'V' is interpreted as the value to be output. The expr is followed by a 'c', a 'd', or a '"'.

'c' causes the low order 8 bits of the value of the expression to be output as a single byte.

'd' causes the value of the expression to be output in decimal.

'"' causes the string which follows (terminated by another '"' to be output the number of times specified by the value of the expression. The characters in the string are interpreted as 7-bit, so if a '"' must be repeated, it can be expressed as \242.

Examples:

:ah=EG,%Vd,$:
Absolute horizontal positioning on the IDS is done by sending ESC, 'G', ',', the position to move to as a string of decimal digits, ',', '$'.

:iu=E 36%V+1c:
The vertical increment on a Diablo 630 is set by sending ESC, '\036', and the binary value of the size of the increment plus 1 as a single byte.

:rh=El%V>>6&077 0100c%V&077 0100c:
Relative horizontal positioning on the Sanders is done by sending ESC, 'l', and the value in two bytes, each containing 6 bits, and with the 0100 bit on.

:fl=E?E7%V"0ELE@:Forms length on the NEC 3510 is set by putting the printer in format mode (ESC, '?'), clear- ing forms length (ESC, '7'), sending newlines to feed to the bottom of the page (%V"\n"), setting form length to the number of lines just fed (ESC, 'L'), and exiting format mode (ESC, '@').

:xx=<numlist>:
<numlist> must be a list of decimal, octal or hex numbers, separated by ','. This is frequently used as a list of numbers of an algorithms applicable to the printer being described.

DETAILED DESCRIPTION OF /etc/printcap ENTRIES

Bold text:

:bs#<num>:
bold start routine number or

:bs=<string>:
bold start string

:be#<num>:
bold end routine number or

:be=<string>:
bold end string

:bo=<expr>:
bold offset

:bc=<expr>:
bold repeat count

bs can be a number of a routine to enter bold mode or a string which causes the printer to start printing in the bold mode.

be can be a number of a routine to exit bold mode or a string which causes the printer to stop printing in the bold mode.

If 'bs' and 'be' are numbers, they must be identical. Routines currently in existence are 1 and 2.

Routine 1 prints the char 'bc' times, advances the printhead a distance of 'bo', and prints the char 'bc' times. If 'bo' is 0, the char will print 'bc' times (not 2*'bc').

Routine 2 disables bolding.

Overstruck text:

:os#<num>:
overstrike start routine number or

:os=<string>:
overstrike start string

:oe#<num>:
overstrike end routine number or

:oe=<string>:
overstrike end string

os can be a number of a routine to enter overstrike mode or a string which causes the printer to start printing in overstrike mode.

oe can be a number of a routine to exit overstrike mode or a string which causes the printer to stop printing in overstrike mode.

If 'os' and 'oe' are numbers, they must be identical. Routines currently in existence are 1 and 2.

Routine 1 prints each character overstruck with a '/'.

Routine 2 disables overstriking.

Underlined text:

:us#<num>:
underline start routine number or

:us=<string>:
underline start string

:ue#<num>:
underline end routine number or

:ue=<string>:
underline end string

us can be a number of a routine to enter underline mode or a string which causes the printer to start printing in underline mode.

ue can be a number of a routine to exit underline mode or a string which causes the printer to stop printing in underline mode.

If 'us' and 'ue' are numbers, they should be identical. Routines currently in existence are 1 and 2.

Routine 1 prints the characters to be underlined, backs up the print head, and underlines them.

Routine 2 disables underlining.

Double underlined text:

:ds#<num>:
double underline start routine number or

:ds=<string>:
double underline start string

:de#<num>:
double underline end routine number or

:de=<string>:
double underline end string

:di=<expr>:
double underline increment

ds can be a number of a routine to enter double underline mode or a string which causes the printer to start printing in double underline mode.

de can be a number of a routine to exit double underline mode or a string which causes the printer to stop printing in double underline mode.

If 'ds' and 'de' are numbers, they should be identical. Routines currently in existence are 1 and 2.

Routine 1 prints the characters to be underlined, backs up the print head, underlines them, moves down the double underline increment, backs up the print head, underlines again, and moves up the double underline increment.

Routine 2 disables double underlining.

Unrecognized sequences:

:Bs#<num>:
bad sequence start routine number or

:Bs=<string>:
bad sequence start string

:Be#<num>:
bad sequence end routine number or

:Be=<string>:
bad sequence end string

Bs can be a number of a routine to enter bad sequence display mode or a string which causes the printer to start printing in this mode.

Be can be a number of a routine to exit bad sequence display mode or a string which causes the printer to stop printing in this mode.

If 'Bs' and 'Be' are numbers, they should be identical. Routines currently in existence are 1 and 2.

Routine 1 prints the characters to be displayed in this mode, backs up the print head, and prints "^"'s underneath them.

Routine 2 disables the special mode display.

Horizontal motion:

:hf=<numlist>:
list of horizontal positioning routine numbers

hf is a list of routine numbers to be tried for moving horizontally.

All routines in the list will be tried, and the one producing the fewest bytes will be used. Routines currently in existence are 1, 2, 3, 4, and 5.

Routines 1 to 4 output spaces or the string 'bk' to move to the desired location, changing the increment as necessary, using the routine specified in 'hi'.

Routines 2 and 4 change to the largest increment first.

Routines 1 and 3 only change the increment if the distance to be traversed is not an integral multiple of the current increment.

Routines 1 and 2 move left by sending backspaces.

Routines 3 and 4 move left by sending the string 'cr' followed by spaces.

Routine 5 moves to the desired horizontal location by sending the number of minimum increments 'lh' formatted according to 'ah'.

:ah=<format>:
absolute horizontal positioning string

ah is used only if horizontal positioning routine 5 is selected.

:nb:
no backspace (for diablo firmware)

nb prevents routines 1 thru 4 from sending a backspace. The firmware in some Diablo 630's does not always get its backspace increment from the horizontal increment register.

:hi#<num>:
horizontal increment routine

:lh=<expr>:
minimum (littlest) horizontal increment

:bh=<expr>:
maximum (biggest) horizontal increment

:ih=<format>:
horizontal increment string

hi is the number of the routine to set the horizontal increment. This is only used if routine 1, 2, 3, or 4 is specified in the 'hf' list. Routines currently in existence are 1 and 2.

Routine 1 is used to change the horizontal increment by sending the number of minimum increments 'lh' formatted according to 'ih'. It never changes to an increment larger than 'bh'.

Routine 2 does nothing.

:cr=<string>:
carriage return

cr is the string sent to move the printhead to the left margin.

:bk=<string>:
backspace

bk is the string sent to move the printhead left one character position.

Vertical motion:

:vf=<numlist>:
list of vertical positioning routine numbers

vf is a list of routine numbers to be tried for moving vertically.

All routines in the list are tried, and the one producing the fewest bytes is used. Routines currently in existence are 1, 2, and 3.

Routines 1 and 2 output up or down strings to move to the desired location, changing the increment as necessary, using the routine specified in 'vu' or 'vd' as appropriate.

Routine 2 changes to the largest increment first.

Routine 1 only changes the increment if the distance to be traversed is not an integral multiple of the current increment.

Routine 3 is used only for an IDS printer.

Routine 4 outputs the string 'ff' if the destination is the bottom of the page.

:vd#<num>:
downward vertical increment routine number

:vu#<num>:
upward vertical increment routine number

:bv=<expr>:
maximum (biggest) vertical increment

:lv=<expr>:
minimum (littlest) vertical increment

:id=<format>:
downward vertical increment string

:iu=<format>:
upward vertical increment string

vd is the number of the routine to set the downward vertical increment.

vu is the number of the routine set the upward vertical increment. These are only used if routine 1 or 2 is specified in the 'vf' list. Routines currently in existence are 1 and 2.

Routine 1 is used to change the vertical increment by sending the number of minimum increments 'lv' formatted according to 'id' or 'iu'. It never changes to an increment larger than 'bh'.

Routine 2 does nothing.

:ud:
up and down vertical increments are kept in separate registers

This indicates that 'id' and 'iu' load different registers in the printer.

:ii=<format>:
Third vertical increment string

ii initializes another vertical increment register for the IDS printer.

:dn=<string>:
line feed

:up=<string>:
reverse line feed

dn and up are the strings sent to move down or up on the page by the current vertical increment.

:ff=<string>:
form feed

ff is the string sent to eject a page.

:xv=<expr>:
extra vertical increment for good paper registration when moving upwards

When moving paper backwards, it is sometimes necessary to move back some distance beyond the desired location, and then move forward in order to take up paper slack. xv is this distance.

:dm=<string>:
defeat firmware

Some printers have optimizing firmware that takes a sequence such as 'up 3 units, down 2 units', and turns it into 'up 1 unit'. If the hardware on such a printer does not give good vertical registration, setting the 'xv' value to nonzero does not have the desired effect. The firmware defeats the software attempt to make up for the hardware limitations. dm is any string which convinces the firmware that it has printed something. This prevents the firmware from outsmarting the software.

:sd#<num>:
subscript routine number

:su#<num>:
superscript routine number

Routines currently in existence are 1 and 2.

Routine 1 moves vertically by a fraction of a line.

Routine 2 disables superscripting and subscripting.

Miscellaneous:

:is=<string>:
initialization string

is is the string sent to reset the printer to a known state for all attributes except forms length and top of form.

:tf=<string>:
top of form initialization string

tf is the string sent to reset forms length and top of form. This is used by spoolers between print jobs. WP uses fl and fi instead. This string may cause the printer to read its switch settings or set forms length to a fixed value.

:ir#<num>:
initialization routine number

Routines currently in existence are 1 and 2.

Routine 1 does nothing.

Routine 2 is used only by the IDS.

:fi=<expr>:
forms length increment

:fl=<format>:
forms length initialization

The page length is set using fl. The value formatted ('V') is 'fi', or the current line height if 'fi' evaluates to 0.

:p1=<expr>:
pitch if 10 cpi selected from menu

:p2=<expr>:
pitch if 12 cpi selected from menu

:p3=<expr>:
pitch if 15 cpi selected from menu (IDS uses 16.8)

These are the fractions of an inch to be used as a character width for the menu selections labeled 10, 12, and 15 pitch.

:wh=<string>:
printwheel descriptor file

This is the file from which font information is taken.

:of=<string>:
output filter

This is an executable file, which filters wpprfmt's output into printer-specific, print-ready format.

:qo#<num>:
quick output routine number

Routines currently in existence are 1 and 2.

Routine 1 is the normal one to use.

Routine 2 is used only by the IDS.

ALPHABETIC LISTING OF PRINTCAP SWITCHES

:ah=<format>:
absolute horizontal positioning string

:bc=<expr>:
bold repeat count

:be#<num>:
bold end routine number

:be=<string>:
bold end string

:Be#<num>:
unrecognized \ sequence end routine number

:Be=<string>:
unrecognized \ sequence end string

:bh=<expr>:
maximum (biggest) horizontal increment

:bk=<string>:
backspace

:bo=<expr>:
bold offset

:bs#<num>:
bold start routine number

:bs=<string>:
bold start string

:Bs#<num>:
unrecognized \ sequence start routine number

:Bs=<string>:
unrecognized \ sequence start string

:bv=<expr>:
maximum (biggest) vertical increment

:cr=<string>:
carriage return

:de#<num>:
double underline end routine number

:de=<string>:
double underline end string

:di=<expr>:
double underline increment

:dm=<string>:
defeat firmware - send this string to keep the printer from optimizing out our ups and downs, intended to produce good paper registration when moving upwards

:dn=<string>:
line feed

:ds#<num>:
double underline start routine number

:ds=<string>:
double underline start string

:ff=<string>:
form feed

:fi=<expr>:
forms length increment

:fl=<format>:
forms length initialization string

:hf=<numlist>:
list of horizontal positioning routine numbers

:hi#<num>:
horizontal increment routine

:id=<format>:
downward vertical increment string

:ih=<format>:
horizontal increment string

:ii=<format>:
third vertical increment string

:ir#<num>:
initialization routine number

:is=<string>:
initialization string

:iu=<format>:
upward vertical increment string

:lh=<expr>:
minimum (littlest) horizontal increment

:lv=<expr>:
minimum (littlest) vertical increment

:nb:
no backspace (for diablo firmware)

:oe#<num>:
overstrike end routine number

:oe=<string>:
overstrike end string

:of=<string>:
output filter

:os#<num>:
overstrike start routine number

:os=<string>:
overstrike start string

:p1=<expr>:
pitch if 10 cpi selected from menu

:p2=<expr>:
pitch if 12 cpi selected from menu

:p3=<expr>:
pitch if 15 cpi selected from menu (IDS uses 16.8)

:qo#<num>:
quick output routine number

:sd#<num>:
subscript routine number

:su#<num>:
superscript routine number

:tf: reset top of form and set form length to a known value

:ud:
up and down vertical increments are kept in separate registers

:ue#<num>:
underline end routine number

:ue=<string>:
underline end string

:up=<string>:
reverse line feed

:us#<num>:
underline start routine number

:us=<string>:
underline start string

:vd#<num>:
downward vertical increment routine number

:vf=<numlist>:
list of vertical positioning routine numbers

:vu#<num>:
upward vertical increment routine number

:wh=<string>:
printwheel descriptor file

:xv=<expr>:
extra vertical increment for good paper registration when moving upwards

FILES

/etc/printcap
file containing printer descriptions

SEE ALSO

printcap(5)

NAME

profile - setting up an environment at login time

DESCRIPTION

If your login directory contains a file named **.profile**, that file will be executed (via the shell's **exec .profile**) before your session begins; **.profiles** are handy for setting exported environment variables and terminal modes. If the file **/etc/profile** exists, it will be executed for every user before the **.profile**. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
# Tell me when new mail comes in
MAIL=/usr/spool/mail/myname
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
```

Currently /etc/profile defines lpr defaults for users printing from the global menu, system language defaults (eg. LANGUAGE=EN), and specifies XON/XOFF protocol for Fortune terminals.

FILES

s-1$HOME/.profile
/etc/profile

SEE ALSO

login(1), mail(1), menu(1), sh(1), stty(1), su(1).

BUGS

At present, the "/etc/profile" file is only implemented for users whose shell is menu(1).

NAME

termcap - terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals, used, *e.g.*, by *vi*(1) and *curses*(3). Terminals are described in *termcap* by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap*.

Entries in *termcap* consist of a number of ':' separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by characters. The first name is always two characters long and is used by older version six systems which store the terminal type in a 16-bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name can contain blanks for readability.

CAPABILITIES

(P) indicates padding can be specified
(P*) indicates that padding can be based on no. lines affected

Name	Type		Pad?	Description
ae	str	(P)		End alternate character set
al	str	(P*)		Add new blank line
am	bool			Terminal has automatic margins
as	str	(P)		Start alternate character set
bc	str			Backspace if not ^H
bs	bool			Terminal can backspace with ^H
bt	str	(P)		Back tab
bw	bool			Backspace wraps from column 0 to last column
CC	str			Command character in prototype if terminal settable
cd	str	(P*)		Clear to end of display
ce	str	(P)		Clear to end of line
ch	str	(P)		Like cm but horizontal motion only, line stays same
cl	str	(P*)		Clear screen
cm	str	(P)		Cursor motion
co	num			Number of columns in a line
cr	str	(P*)		Carriage return, (default ^M)
cs	str	(P)		Change scrolling region (vt100), like cm
cv	str	(P)		Like ch but vertical only.
da	bool			Display can be retained above
dB	num			Number of millisec of bs delay needed
db	bool			Display can be retained below

dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give :ei=: if ic
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no cm)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing is
im	bool		Insert mode (enter); give :im=: if ic
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by other function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of keypad transmit mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of other keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in keypad transmit mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on other function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no cm)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default \n)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes

pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with is)
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use cm
ti	str		String to begin programs that use cm
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like ce \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the *termcap* file as of this writing. (This particular concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:\
    :is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
    :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:\
    :cm=\Ea%+ %+ :co#80:dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:\
    :im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
    :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries can continue onto multiple lines by giving a \ as the last character of a line, and empty fields can be included for readability (here between the last field on a line and the first field on the next). Capabilities in *termcap* are of three types: Boolean capabilities which indicate the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the

Concept has "automatic margins" (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability am. Hence the description of the Concept includes am. Numeric capabilities are followed by the character '#' and then the value. Thus co which indicates the number of columns the terminal has gives the value '80' for the Concept.

Finally, string valued capabilities, such as ce (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. An '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When an '*' is specified, it is sometimes useful to give a delay of the form '3.5' and specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A \E maps to an ESCAPE character, ^x maps to a control-x for any appropriate x, and the sequences \n \r \t \b \f give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ can be given as \^ and \\. If it is necessary to place a : in a capability it must be escaped in octal as \072. If it is necessary to place a null character in a string capability it must be encoded as \200. The routines which deal with *termcap* use C strings, and strip the high bits of the output very late so that a \200 comes out as a \000 would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it or limitations in *ex*. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor looks there rather than in */etc/termcap*. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

Basic Capabilities

The number of columns on each line for the terminal is given by the co numeric capability. If the terminal is a CRT, then the number of

lines on the screen is given by the li capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the am capability. If the terminal can clear its screen, then this is given by the cl string capability. If the terminal can backspace, then it should have the bs capability, unless a backspace is accomplished by a character other than ^H in which case you should give this character as the bc string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the os capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor never attempts to backspace around the left edge, nor attempts to go up locally off the top. The editor assumes that feeding off the bottom of the screen causes the screen to scroll up, and the am capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on, i.e. am.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the model 33 teletype is described as

t3 33 tty33:co#72:os

while the Lear Siegler ADM-3 is described as

cl adm3 3 lsi adm3:am:bs:cl=^Z:li#24:co#80

Cursor addressing

Cursor addressing in the terminal is described by a cm string capability, with *printf*(3s) like escapes %x in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the cm string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the % encodings have the following meanings:

%das in printf, 0 origin
%2like %2d
%3like %3d
%.like %c
%+xadds x to value, then %.
%>xyif value > x adds y, no output.
%rreverses order of line and column, no output
%iincrements line/column (for 1 origin)
%%gives a single %
%nexclusive or row and column with 0140 (DM2500)
%BBCD (16*(x/10)) + (x%10), no output.
%DReverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the

order of the rows and columns is inverted here, and the row and column are printed as two digits. Thus its cm capability is "cm=6\E&%r%2c%2Y". The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, "cm=^T%.%.". Terminals which use "%." need to be able to backspace the cursor (bs or bc), and to move the cursor up one line on the screen (up introduced below). This is necessary because it is not always safe to transmit \t, \n ^D and \r, as the system can change or discard them.

A final example is the LSI ADM3a, which uses row and column offset by a blank character, thus "cm=\E=%+ %+ ".

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as nd (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as up. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as ho; similarly a fast way of getting to the lower left hand corner can be given as ll; this may involve going up with up from the home position, but the editor never does this itself (unless ll does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as ce. If the terminal can clear from the current position to the end of the display, then this should be given as cd. The editor only uses cd from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as al; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as dl; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as sb, but just al suffices. If the terminal can retain display memory above then the da capability should be given; if display memory can be retained below then db should be given. These let the editor understand that deleting a line on the screen can bring non-blank lines up from below or that scrolling back with sb can bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *termcap*. The

most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "abc def" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the "def" which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability in, which stands for "insert null". If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as im the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as ei the sequence to leave insert mode (give this, with an empty value also if you gave im so). Now give as ic any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give ic, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in ip (a string option). Any other sequence which needs to be sent after an insert of a single character can also be given in ip.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability mi to speed up inserting in this case. Omitting mi affects only speed. Some terminals (notably Datamedia's) must not have mi because of the way their insert mode works.

Finally, you can specify delete mode by giving dm and ed to enter and exit delete mode, and dc to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode

these can be given as so and se respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining - half bright is not usually an acceptable "standout" mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then ug should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as us and ue respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as uc. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as vb; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, this can be given as vs and ve, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as ti and te. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability ul. If overstrikes are erasable with a blank, then this should be indicated by giving eo.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as ks and ke. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and

home keys can be given as kl, kr, ku, kd, and kh respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as k0, k1, ..., k9. If these keys have labels other than the default f0 through f9, the labels can be given as l0, l1, ..., l9. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the termcap 2 letter codes can be given in the ko capability, for example, ":ko=cl,ll,sf,sb:", which says the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as pc.

If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, then this can be given as ta.

Hazeltine terminals, which don't allow '~' characters to be printed should indicate hz. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate nc. Early Concept terminals, which ignore a linefeed immediately after an am wrap, should indicate xn. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), xs should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate xt. Other specific terminal problems may be corrected by adding more capabilities of the form xx.

Other capabilities include is, an initialization string for the terminal, and if, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, is is printed before if. This is useful where if is */usr/lib/tabset/std* but is clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability tc

can be given with the name of the similar terminal. This capability must be last and the combined length of the two entries must not exceed 1024. Since *termlib* routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with xx@ where xx is the capability. For example, the entry

hn 2621nl:ks@:ke@:tc=2621:

defines a 2621nl that does not have the ks or ke capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/etc/termcap file containing terminal descriptions

SEE ALSO

ex(1), more(1). tset(1), ul(1), vi(1), curses(3), termcap(3).

LIMITATIONS

Ex allows only 256 characters for string capabilities, and the routines in *termcap(3)* do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) must not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

NAME

trans - translation table file

DESCRIPTION

Translation table files contain labeled strings. They are often used to lookup a string associated with a particular label. For example, the label "INSERT-DISK" might have a string value of

"Please insert the flexible disk and press <return>."

Software that needs to have a diskette inserted can lookup the label "INSERT-DISK" and display the string that is found. If the translation table is changed, a different message would be displayed. This happens without any change to the software which is doing the lookup and displaying the message.

Much of Fortune Systems' software uses translation tables. This often allows a single program to work in different environments. Only the translation table needs to be changed.

The translation table file consists of two parts. The first contains the labels and the offsets into the file of the corresponding values. The second part conains these values. Both the labels and the string values are terminated by the null character.

SEE ALSO

printstring(1)

LIMITATIONS

The file is normally generated by a string compiler which is currently not available to Fortune customers.

NAME

ttys - terminal initialization data

DESCRIPTION

The *ttys* file is read by the *init* program and specifies which terminal special files are to have a process created for them which allows people to log in. It contains one line per special file.

The first character of a line is either '0' or '1'; the former causes the line to be ignored, the latter causes it to be effective. The second character is used as an argument to *getty*(8), which performs such tasks as baud-rate recognition, reading the login name, and calling *login*. For normal lines, the character is '0'; other characters can be used, for example, with hard-wired terminals where speed recognition is unnecessary or which have special characteristics. (*Getty* has to be fixed in such cases.) The remainder of the line is the terminal's entry in the device directory, /dev.

FILES

/etc/ttys

SEE ALSO

login(1), getty(8), init(8).

NAME

ttytype - data base of terminal types by port

SYNOPSIS

/etc/ttytype

DESCRIPTION

Ttytype is a database containing, for each tty port on the system, the kind of terminal that is attached to it. There is one line per port, containing the terminal kind (as a name listed in termcap (5)), a space, and the name of the tty, minus /dev/.

This information is read by *tset*(1) and by *login*(1) to initialize the TERM variable at login time.

SEE ALSO

login(1), tset(1).

LIMITATIONS

Some lines are merely known as "dialup" or "plugboard".

NAME

utmp, wtmp - login records

SYNOPSIS

#include <utmp.h>

DESCRIPTION

The *utmp* file allows one to discover information about who is currently using the operating system. The file is a sequence of entries with the following structure declared in the include file:

```
/*
 * Structure of utmp and wtmp files.
 *
 * Assuming the number 8 is unwise.
 */
struct utmp {
        char    ut_line[8];   /* tty name */
        char    ut_name[8]; /* user id */
        long    ut_time;      /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of *time*(2).

The *wtmp* file records all logins and logouts. Its format is exactly like *utmp* except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name '~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names '|' and '}' indicate the system-maintained time just before and just after a *date* command has changed the system's idea of the time.

Wtmp is maintained by *login*(1) and *init*(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by *ac*(8).

FILES

/etc/utmp

/usr/adm/wtmp

SEE ALSO

login(1), who(1), ac(8), init(8).

NAME

whl - print wheel (font) descriptor file

DESCRIPTION

A printwheel descriptor file contains descriptions of all the known fonts for a particular printer. Text on a line following a '#' is ignored:

```
#Comment line
+init "xxx"     #the init string is "xxx"
```

The following items appear in the descriptions below. They have the following meanings:

[Name1], [Name2], [Name3], etc.
These are strings by which this entry is identified. They can contain any combination of alphabetics and numerics.

[pitch]
This appears as a fraction of an inch (e.g. 1/10, 1/15). It must be expressed in terms of integers (1/16.8 should appear as 10/168). In a monospaced font, it is the width of a character. In a proportional spaced font, it is the width of an em-space (the widest character).

[ps default width]
This appears as a fraction of the pitch. It must be expressed in terms of integers. This is used only in a proportional spaced font. It is the most common character width in the font. It is used as the default for any character which does not have an explicit width.

[char width]
This appears as a fraction of the pitch. It must be expressed in terms of integers. This is used only in a proportional spaced font. It is the width of the character defined on the current line.

[videotex]
This is the videotex representation of the character being defined.

[string]
This is the string sent to the printer to display the videotex character.

[videotex] and [string] are sequences of characters. [videotex] begins at the beginning of the line, and is terminated by a space or tab. [string] is delimited by '"' at either end. \ sequences recognized as single characters are:

```
\n   =   012
\r   =   015
\f   =   014
```

\t = 011
\E = 033
\[= 033
\b = 0210
\xxx = the character whose octal value is xxx
\x = the character 'x' where x is any character not listed above. In particular:

\\ = '\'
\" = '"'
\# = '#'
\> = '>'
\+ = '+'
\ = ' '

A font description starts with the name line:

>[Name1] [Name2] [Name3] ...(etc.)

and continues until the name line for the next font or end of file.

Next in the file are optional data lines. These can be:

+init "[string]"
[string] is the character sequence which must be sent in order to select this font. If this line is absent, no init string is sent.

+pitch ([pitch])
[pitch] is a fraction of an inch. If this line is absent, a default pitch is assumed.

+ps ([ps default width])
If this line is absent, the font is monospaced. If present, it is proportional spaced. [ps default width] is optional. If present, it is a fraction of the pitch, and is used as the width of any character not having an entry containing a width.

Next in the file are optional character lines.

[videotex] "[string]" ([char width])

These describe the character sequence which must be sent to print the character described on this line, and the width of the character. There are four types of videotex characters described in the file. The following list explains each of these types:

1. Standard printable ascii (non-overstriking): in the range 040-0177.

entry absent
The output string defaults to the ascii character. The character width defaults to [ps default width], or [pitch] if not present.

[videotex] "[string]"
If [string] is null, this character is not available. Otherwise, the output string is [string], and the character width defaults to [ps default width], or [pitch] if not present.

[videotex] ([char width])
The output string defaults to the ascii character. [char width] is a fraction of pitch.

[videotex] "[string]" ([char width])
The output string is [string]. [char width] is a fraction of pitch.

2. Non-overstriking supplementary characters: SS2 (031), followed by a character in the range 040-077 or 0120-0177.

entry absent
This character is not in the font.

[videotex] "[string]"
The output string is [string]. Character width defaults to [ps default width], or [pitch] if not present.

[videotex] "[string]" ([char width])
The output string is [string]. [char width] is a fraction of pitch.

3. Overstriking supplementary characters which can be combined with non-overstriking characters: SS2 (031), followed by a character in the range 0100-0117.

entry absent
This character is not in the font.

[videotex] "[string]"
The output string is [string]. Printing this string leaves the position of the printhead unchanged. If printing the character causes the head to advance, the string must end with '\b'. Since this is an overstrike, it is not considered to have a width.

4. Overstriking supplementary characters already combined with non-overstriking characters: SS2 (031), followed by a character in the range 0100-0117, followed by a printable ascii character (in the range 040-0177).

entry absent
This character is not in the font.

[videotex] "[string]"
The output string is [string]. Character width defaults to [ps default width], or [pitch] if not present.

[videotex] "[string]" ([char width])
The output string is [string]. [char width] is a fraction of pitch.

In types 3 and 4, all descriptions containing the same overstriking character must appear in adjacent lines within a font description (e.g. a-umlaut, o-umlaut, umlaut alone).

EXAMPLE

Sample wheel file entry:

```
# wp print wheel 4 on NEC
# Multilingual-A 803-020004-702
>Multi_lingual 4
+init "^O"
\177""#not on wheel
#supplementary set
# (^Y below indicates that the control character is in
#  the file, not that the file contains the two
#  characters '^' and 'Y').
^YA       "^N[^O\b"          #  All
^YAa      "^N'^O\ba"         #  the
^YAe      "^N'^O\be"         #  ^YA's
^YAi      "^N'^O\bi"         #  must
^YAo      "^N'^O\bo"         #   be
^YAu      "^N'^O\bu"         # adjacent.
^YAc      "^N'^O\bc"         #   .
^YAj      "^N'^O\bj"         #   .
^YB       "^N@^O\b"          # The first three
^YBa      "^NF^O\ba"         # ^YB's are all
^YBe      "^NF^O\be"         # that will be seen.
^YC       "^N>^O\b"
^YBi      "^NF^O\bi"         # These ^YB's will
^YBo      "^NF^O\bo"         # be ignored
^YBu      "^NF^O\bu"         # due to the
^YBc      "^NF^O\bc"         # intervening ^YC.
^YBj      "^NF^O\bj"         # This is an error.
^YD       "^N<^O\b"
\031H     "^NX^O\b"          # \031 is identical to ^Y
\031Ha    "^NA^O\ba"         #   .
\031He    "^NA^O\be"         #   .
\031Hi    "^NA^O\b^NN^O"     #   .
\031Ho    "^NA^O\bo"         #   .
```

```
\031Hu      "^NA^O\bu"      #
^YJ         "^NR^O\b"
^YJA        "^NW^O"
^YJa        "^NV^O"
^YK         "^NE^O\b"
^YO         "^ND^O\b"
^Y#         "^NB^O"
^Y!         "^NC^O"
^Y'         "^NG^O"
^Ya         "^NI^O"
^Yz         "^NK^O"
^Y?         "^NL^O"
^YT         "^NM^O"
^Yj         "^NO^O"
^Yy         "^NP^O"
^YS         "^NQ^O"
^Yv         "^NS^O"
^Y{         "^NT^O"
^Yi         "^NU^O"
^Yq         "^NY^O"
^Y"         "^NZ^O"
^Y6         "^N]^O"
^Y(         "^N^^O"
^YR         "^N_^O"
```

FILES

Any files with the .whl extension.

SEE ALSO

mapvtx(3)

NAME

intro - introduction to section 8

DESCRIPTION

Section 8 of the Programmer's Manual describes system management programs. These utilities mostly reside in */etc* and usually must be run by the system manager. Most of the time the system manager will not need to run these programs directly; they are run automatically when the need arises by the /etc/rc shell file, or are not needed. See (*rc*(8).)

NAME

boot -the standalone shell

SYNOPSIS

/sa/boot

DESCRIPTION

/sa/boot, also known as the "Level 1" boot, is the 32:16's equivalent of the standalone shell. It is a standalone program which normally resides in boot partition 0 of every hard and some floppy disks used on the 32:16. When booting up, a level 0 boot program (usually in ROM), reads in the indicated level 1 boot program (0-7) from the indicated device and drive. The level 1 boot program, usually */sa/boot*, then reads in the boot file name stored in the EAROM. This usually is hd02/unix (or fd02/unix). If there is no file name specified, or the specified file does not exist, a prompt appears and the user may type any file. The syntax supplied to boot must be of the form: *device/file*, where device is one of hd[0-3][0-7] or fd[0-3][0-7]. For instance, to get the reconfiguration menu, (see *reconf*(8)), one can reboot the system, holding the DEL key. First, type the F7 key and *hd02/sa/reconf*. Then type the RETURN key and then the EXECUTE key. The reconfiguration menu will appear.

SEE ALSO

bootmenu(8).

LIMITATIONS

/sa/boot is actually just an image of what is in the boot partition. Changing or removing */sa/boot* will not affect normal operation of the 32:16. */sa/boot* is a standalone program and cannot be invoked through UNIX.

NAME

bootcp - copy a boot image

SYNOPSIS

bootcp *ifile ofile bootno*

DESCRIPTION

Bootcp copies a boot image from *ifile* to *ofile*. *ifile* can be a regular file or a character device. If it is a regular file, it is assumed to be the boot image. If it is a character device, *boono*th boot is read. *ofile* can be a regular file or a character device. If it is a regular file, it will contain the boot. If it is a character device, *bootno*th boot contains the boot. If the *bootno* boot exists, it is overwritten. If it doesn't, space for it is found in partition 0. In either case the configuration block is updated.

SEE ALSO

devctl(2), diskconf(5), disktab(5), rdconf(8).

DIAGNOSTICS

Error messages if the *ifile* does not exist, if the *ifile* is not a regular file or a character special file, if the *ofile* is not a regular file or a character special file or if the partition 0 doesn't have enough space for the boot.

NAME

bootmenu - first power-up menu

DESCRIPTION

The boot menu allows you to decide what the 32:16 does on power-up or reset, which device, drive, and boot program # to used in booting up, the brand(s) of floppy drive(s), the UNIX file name brings up, and the default baud rates used on the front and back ports. Each line of the menu has a short description, the current setting of that field (if appropriate to the command), and the function key which picks that field (highlighted). The EXECUTE key quits the menu and either goes into terminal mode or starts the boot device and drive routine. The HELP key gives a description of how to use each command.

F1	Change Front port speed	current setting
F2	Change Back port speed	" "
F3	Change power-up action	" "
F4	Change boot device	" "
F5	Change boot program number	" "
F6	Change floppy drive #0 type	" "
F7	Change boot file name	" "
F8	Read settings from EAROM	
F9	Save settings into EAROM	
F10	Start system from floppy disk	
EXECUTE	Start up system using information on screen	
HELP	For more information	

The function keys are brightened to indicate they are to be depressed. Pressing F1 to F10 picks one of the possible fields, the chosen field is indicated by showing the description in reverse video. F1-F6 are changed by typing the space key. F7 requires the user to type in a string *device/file,* followed by RETURN. F8 and F9 do not have settings to change; the underlined message confirms the action was taken. F10 sets the proper settings you need to run the cold boot diskette or the floppy-based unix diskette. The EXECUTE key goes ahead and boots, using the information on the screen but not writing the EAROM. The HELP key shows an alternate menu, giving a slightly longer description of each field.

If there is some problem in booting up the "Level 1" boot, the boot menu is entered with a flashing error message.

DESCRIPTION OF COMMANDS

Change Front port speed:

Change Back port speed:

The Fortune keyboard port (TTY00) and the port on the back of the machine (TTY01) are RS-232C serial lines. These can operate at various speeds. The Fortune keyboard should be at 2400

baud. The back port can be used for various purposes and the speed can be set for however it is used. These are default speeds only. They are in effect whenever the MomROM functions as a terminal, and also under UNIX until they are changed. TTY00 can be set to any baud rate if that port is to be used for operation with other than the Fortune integral keyboard.

Change power-up action:

Decide what the machine will do when you turn it on. It can boot from the "Boot Device", show the menu, or act as a terminal with another computer connected to tty01.

Change boot device:

Decide from which peripheral and drive number to boot. The 32:16 can boot from the back port, the floppy disk drive, and some of the I/O option cards.

Booting from the hardware peripherals consists of using data stored on the hardware. Booting from a serial line requires another computer at the other end of the line. Most of the disk drive peripherals can have more than one drive attached to them, and the drives have unique numbers, starting from 0. This field also sets which drive on the above-named device will be used.

Change boot program number:

Each disk drive can have one or more *boot programs*. This number specifies which boot program should be used to find the 'level 1 boot file'. See *bootcp*(8).

Change floppy drive #*n* type:

The 32:16 can have up to four floppy disk drives attached to it. This allows you to indicate which model of floppy disk drive is connected to each of the four channels. You will notice that there are four drives and only one type shown; drive number *n* is changed, where *n* is 0 unless the boot device is set to floppy drive #1, #2, or #3. You should never have to change this.

Read settings from EAROM:

This overwrites the current settings on the screen with the contents of the non-volatile part of the EAROM.

Save settings into EAROM:

This saves the settings on the screen into the permanent part of the EAROM.

Start system from floppy disk:

This sets the other items on the screen to the necessary settings for running the floppy-based FOS system (such as cold boot disk #1). It also sets the EAROM root device and swap device fields (see *reconf*(8)) to "fd02" and "fd01", respectively. This merely allows you to boot up on floppy-based unix once; it only

changes the volatile portion of the EAROM.

See *reconf*(8) for a full description of what these fields do.

EXECUTE

This saves the settings on the screen into the volatile portion of the EAROM. They are only in force until the machine is reset or powered down, at which time the non-volatile contents of the EAROM are consulted.

HELP

This replaces the screen with a list of terse reminders of each key's function; it is no replacement for this document.

See *momrom*(8) for information to run the boot menu.

DIAGNOSTICS

Unexpected processor trap!

This is caused by an unexpected processor trap, and could indicate a program bug or an unexpected device interrupt. Probably what happened is that the indicated level 1 boot program is garbage. If this happens, try booting from a floppy; for hard disk systems, use the cold boot floppy.

Downloading:

This message appears when using terminal mode. The download function has started (because the 32:16 received a ^D from the back port), and the processor is accepting records from the back port.

These messages can appear with the boot menu:

EAROM checksum error!

The Motherboard EAROM has been changed too many times and needs to be replaced, or the data in the EAROM was accidentally changed. The F9 key should make this message go away; if it doesn't, consult your dealer.

Please set boot device The EAROM code for the boot device means nothing.

When booting from the floppy:

No disk in floppy drive

Bad conf block on floppy

Empty boot program on floppy

Bad boot program on floppy

Floppy drive can't seek

Floppy has bad block

Floppy does not respond

The floppy disk boot routine ran into some problem; The first message is obvious. The last message indicates that the floppy controller may be broken. The rest are explained in *boot*(8), *disk*(5) and *diskconf*(8).

Any of the above problems can also occur when booting from the hard disk.

OLD VERSIONS

Older versions of the motherboard PROMs use "There's something wrong, start over" instead of "Unexpected processor trap," and "EAROM may not work" instead of "EAROM checksum error."

SEE ALSO

boot(8), momrom(8), reconf(8).

NAME

chown, chgrp - change owner or group

SYNOPSIS

/etc/chown *owner file...*

/etc/chgrp *group file...*

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner can be either a decimal UID or a login name found in the password file.

Chgrp changes the group-ID of the *files* to *group*. The group can be either a decimal GID or a group name found in the group-ID file.

Only the super-user can change owner or group, in order to simplify as yet unimplemented accounting procedures.

FILES

/etc/passwd

/etc/group

SEE ALSO

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

NAME

dskselect - select a disk configuration block

SYNOPSIS

dskselect *file* [*disktype*]

DESCRIPTION

Dskselect reads the disktab data base and displays the disk type names on the screen after printing the *prompt*. One of the displayed types can be selected by typing its index. A configuration block of the selected disk is written out to *file*. If the *disktype* is given, the user is not prompted and configuration block for that type is written. The */etc/disktab* file contains the data base of the configuration blocks.

SEE ALSO

devctl(2), disktab(5), rdconf(8).

DIAGNOSTICS

Error messages if /etc/disktab can not be found, *disktype* is not in the disktab file or *file* can not be written.

NAME

format - formats a floppy or a hard disk

SYNOPSIS

format [**-c** *confile*] [**-k**] [**-t** *track* **-h** *head*] *device*

DESCRIPTION

Format is a disk formatter. It formats the *device* and writes a new configuration block on it. You must use *mkfs* if you want to create a new filesystem on the device.

OPTIONS

-c *confile*
reads a configuration block from *confile* instead of *device*.

-k saves the bad block information present on the disk. This avoids the need to respare known bad blocks. This option has no effect on an unformatted hard disk. This option, when used on an unformatted floppy disk, will hang the system.

-t *track* **-h** *head*
formats a single track on a head.

SEE ALSO

mkconf(8), mkfs(8), rdconf(8), diskconf(5), mkfs(8).

DIAGNOSTICS

Error messages if the *confile* or *device* does not exist, configuration block can not be read or is invalid, configuration block can not written to *device*, invalid *track* or *head*, or if formatting was unsuccessful.

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

/etc/fsck -p [*file system...*]

/etc/fsck [**-n**] [**-s***X*] [**-S***X*] [**-t***filename*] [**-y**] [*file system*] ...

DESCRIPTION

The first form of *fsck* preens a standard set of file systems or the specified file systems. It is normally used in the script **/etc/rc** during automatic reboot. In this case *fsck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of I/O overlap to check the file systems as quickly as possible. Normally, the root file system is checked on pass 1, other "root" ("a" partition) file systems on pass 2, other small file systems on separate passes (e.g. the "d" file systems on pass 3 and the "e" file systems on pass 4), and finally the large user file systems on the last pass, e.g. pass 5. A pass number of 0 in fstab causes a disk to not be checked; similarly partitions which are not shown as to be mounted "rw" or "ro" are not checked.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the super block wrong

These are the only inconsistencies which *fsck* with the **-p** option corrects; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot then fails. For each corrected inconsistency one or more lines are printed identifying the file system on which the correction takes place, and the nature of the correction. After successfully correcting a file system, *fsck* prints the number of files on that file system and the number of used and free blocks.

Without the **-p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that a number of the corrective actions are not fixable under the **-p** option 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 a **-n** action. *check, dcheck, fcheck,* and *icheck*combined.

OPTIONS

The following flags are interpreted by *fsck.*

-n Assume a no response to all questions asked by *fsck;* do not open the file system for writing.

-s*X* Ignore the actual free list 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.

The **-s***X option allows for creating an optimal free-list organization:*

-sBlocks-per-cylinder:Blocks-to-skip (for anything else)

If *X* is not given, the values used when the file system was created are used.

-S*X*

Conditionally reconstruct the free list. This option is like **-s***X above except the free list is rebuilt only if there were no discrepancies discovered in the file system. Using* **-S** forces a no response to all questions asked by *fsck.* This option is useful for forcing free list 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 the operator 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.

-y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

If no file systems are given to *fsck* then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.

4. Size checks:

   Directory size not 16-byte aligned.

5. Bad inode format.

6. Blocks not accounted for anywhere.

7. Directory checks:

   File pointing to unallocated inode.
   Inode number out of range.

8. Super block checks:

   More than 65536 inodes.
   More blocks for inodes than there are in the file system.

9. Bad free block list format.

10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode 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. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck is executed).*

Checking the raw device is almost always faster, but does not work on /dev/rhd??.

FILES

/etc/fstab
contains default list of file systems to check.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory. If the root (/) file system has to be modified, the system must be rebooted.

SEE ALSO

fstab(5), filsys(5), reboot(8).

LIMITATIONS

Inode numbers for . and .. in each directory should be checked for validity.

-b and **-g** options from *check* should be available in *fsck.*

NAME

getty - set terminal mode

SYNOPSIS

/etc/getty [char]

DESCRIPTION

Getty is invoked by *init*(8) immediately after a terminal is opened, following the making of a connection. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the terminal line. Arguments other than '0' can be used to make *getty* treat the line specially. Normally, it sets the speed of the interface to 300 baud, specifies that raw mode is to be used (break on every character), that echo is to be suppressed, and either parity allowed. It prints the 'login:' message. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is then changed to 1200 baud and the 'login:' is typed again; a second 'break' changes the speed to 150 baud and the 'login:' is typed again. Successive 'break' characters cycle through the speeds 300, 1200, and 150 baud.

The user's name is terminated by a newline or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *stty*(2)).

The user's name is scanned to see if it contains all upper-case alphabetic characters; if so the user will be warned that he should not be using all capital letters. If the user again types logs in using all capital letters, the system is told to map any future uppercase characters into the corresponding lowercase characters. Finally, login is called with the user's name as argument.

The letter codes and their characteristics follow. If Code is given as argument to *getty*, then that line is used. If a break is received by code 0, then code 0a is attempted. Code 0a "breaks" to code 0b, then code 0c, then back to code 0. Codes 0a, 0b, and 0c are not available as arguments. Similarly, codes 3 and 5 switch to each other.

Code	Stty Commands	Input speed	Output speed
0	ANYP,ECHO	300	300
0a	ANYP,XTABS,ECHO,CRMOD	1200	1200
0b	EVENP,ECHO,FF1,CR2,TAB1,NL1	150	150
0c	ANYP,ECHO,CRMOD,XTABS,LCASE	110	110
-	ANYP,ECHO,CRMOD,XTABS,LCASE	110	110
1	EVENP,ECHO,FF1,CR2,TAB1,NL1	150	150
2	ANYP,XTABS,ECHO,CRMOD	9600	9600
3	ANYP,XTABS,ECHO,CRMOD	1200	1200
4	ANYP,ECHO,CRMOD,XTABS	300	300
5	ANYP,ECHO	300	300
6	ANYP,ECHO	2400	2400
a	ANYP,XTABS,ECHO,CRMOD	50	50
b	ANYP,XTABS,ECHO,CRMOD	75	75
c	ANYP,XTABS,ECHO,CRMOD	110	110
d	ANYP,XTABS,ECHO,CRMOD	134	134
e	ANYP,XTABS,ECHO,CRMOD	150	150
f	ANYP,XTABS,ECHO,CRMOD	200	200
g	ANYP,XTABS,ECHO,CRMOD	300	300
h	ANYP,XTABS,ECHO,CRMOD	600	600
i	ANYP,XTABS,ECHO,CRMOD	1200	1200
j	ANYP,XTABS,ECHO,CRMOD	1800	1800
k	ANYP,XTABS,ECHO,CRMOD	2400	2400
l	ANYP,XTABS,ECHO,CRMOD	4800	4800
m	ANYP,XTABS,ECHO,CRMOD	9600	9600
n	ANYP,XTABS,ECHO,CRMOD	19200	19200

SEE ALSO

login(1), stty(2), ttys(5), init(8).

NAME

halt - prepare to stop

SYNOPSIS

/etc/halt

DESCRIPTION

Halt executes any hardware specific functions prior to shutdown, informs the user of its completion and requests that the machine be turned off. It then executes the *pause*(2) primitive.

Halt is not intended to be called directly; rather it is executed from /etc/rc.

SEE ALSO

pause(2), rc(8), shutdown(8).

NAME

init - process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

Init is invoked inside UNIX as the "first" user program. Its general role is to first make sure that the system is ready to run, and then create a process for each typewriter on which a user may log in.

First it prints a "7" in the count-up sequence (see *countup*(8)), and reads the system language code from the EAROM (see *uconf*(8)) and creates the LANGUAGE= environment variable (see *environ*(5)). This is passed in the environment to all user programs. It then forks */bin/sh* to run the shell script */etc/rc*. See *rc*(8). (If /etc/rc exits with nonzero status, or if /etc/rc is not present, it runs a single user shell on */dev/console.)*

To allow logins, init reads the file */etc/ttys* and forks several times to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input and output and the diagnostic output. After an open succeeds, */etc/getty* is called with argument as specified by the second character of the *ttys* file line. *Getty* reads the user's name and invokes *login* to log in the user and execute the user's login program (usually the menu shell or /bin/sh, see *passwd*(5)).

Ultimately the user's shell will exit. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users, and makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. The *wtmp* entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and *getty* is reinvoked.

Init catches the *hangup* signal (signal 1) and interprets it to mean that *init* should start over. Everybody logged in is preremptorily logged out. To do this use: 'kill -1 1.' If there are processes outstanding which are deadlocked (due to hardware or software failure), *init* will not wait for them all to die (which might take forever), but will time out after 30 seconds.

Init's role is so critical that if it dies, the system will stop with an automatic "Software Error 22". If, at bootstrap time, the */etc/init* program cannot be exec'd, the system will stop with "Software Error 18".

DIAGNOSTICS

File "/etc/rc" missing, running in maintenence mode.
The /etc/rc shell script is gone, and *init* is forking /bin/sh so that the system manager can do something.

Can't execute "/bin/sh". Contact your dealer.
The program /bin/sh is gone, or can't run.

Can't execute "/etc/getty". Contact your dealer.
The program /etc/getty is gone, or can't run.

Can't open "/etc/ttys" file
The terminal types database is gone.

See *rc*(8) to find out how to run the maintenence mode shell.

Can't creat "/etc/utmp" file
The database of currently logged in terminals cannot be maintained. This is not serious, and will not hinder multi-user operation.

FILES

/dev/console, /dev/tty??, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), uconf(8), environ(5), ttys(5), getty(8), rc(8), reboot(8), halt(8), shutdown(8)

NAME

lpd - the line printer daemon

SYNOPSIS

lpd -p *printer* **-i** *PID*

OPTIONS

-p *n*

is the printer number, and can be from 0 - 999. The default is 1.

-i *PID*

This option indicates that the daemon is to pass the standard input along to the filter. In this case, the daemon will hang until the lock file is unlocked, indicating that the previous daemon has finished. When it runs, the PID given is checked against the 'P' line in the control file. If they are the same, the job runs; otherwise it releases the lock file, then again hangs until it is unlocked, the assumption being that there is another **-i** job with a lower priority waiting to run. This option is invoked by giving the **-i** option to lpr. The intention is to allow printing of jobs so large that a copy file would require more disk space than is available.

DESCRIPTION

The line printer daemon is the portion of the printer spooling program which decides which file should be printed next, and what options should be given to the printer filter for each job. It also determines the output device, sets the baudrate, etc. for the output device, and prevents other programs from using the output device.

A daemon is responsible for one printer only. Thus, it is possible to have a number of daemons running at the same time, one for each enabled printer. The daemon is normally started by *lpr*(1) (lpr checks to see if a daemon is active by checking the file "lock" in the appropriate directory; if it is there and is locked (via *lockf*(2)) lpr simply exits, (unless invoked with the *-i* option), otherwise it forks and execs lpd with the appropriate option(s). When it starts, lpd also checks to see if a daemon is active for the printer; if so, it exits unless the **-i** option was given.

The daemon determines which job will print next by examining the control file. If there are two or more jobs with the same priority, the one which occurs first in the directory will be printed first. Every job in the queue is examined every time, since priorities may be altered by *lpmv*(1).

When all jobs in the queue have been printed, the daemon exits. This is less of a load on system resources than having a daemon for each printer which wakes up when jobs are submitted.

After each job, the daemon will write the jobname, the size, the printer and the owner to an accounting file, if the file exists and is writable.

CONTROL FILE

Each line in the control file is treated as a unit, there is no conception of continuation lines, and each line must be 128 characters or less. The first byte of each line is the control byte, the meanings are listed below.

+ As each file in a job is successfully printed, the 'F', 'P', or 'S' is replaced with '+', so that if another attempt to print the job is required, only those jobs which were NOT printed are retried.
A Gives the full path name of the output filter.
a Number of times to try and reprint unsuccessful jobs.
B String to be used as the banner on the header page. Passed to the filter as -b.
C Indicates that the filter is to be passed the 'W' option.
c Indicates that setup is to be done by the filter to enable manual feeding of a page for the sheet feeder. (Not available from lpr, used only by lpdun.)
D The job is to be printed in single sheet mode. Passed to the filter as -d.
E The type of terminal the job was invoked from. Passed to the filter as -E.
e The port the job was invoked from (full pathname). Passed to the filter as -e.
F A filename to be passed to the filter as its input file. It is important to note that all lines containing information for the filter must precede the 'F', 'S' and 'P' lines, since the filter is invoked as soon as any of these lines are encountered.
f The formlength (in lines) to be used for the job. Passed to the filter as -F.
G The real group ID of the user who started the job.
H A header page was specifically requested, as opposed to a default header page. Forces the filter to be passed the -o option.
I The real user ID of the user who started the job.
i A string of options which correspond to keywords in the printcap file, of the form ":xy=#:ab=#:....:" where # will be substuted for the letter 'V' in a printcap formula. Passed to the filter as -I.
J The job name, used when sending mail announcing job completion, also passed to the filter, and reported by *lpq*(1).
K The umask of the user who started the job, used only with 'S' below.

L The owner of the job, passed on to the filter, and used as the person to send mail to, if it is a valid user name, (or looks like a uucp or Arpanet address). Passed to the filter as -o, if a header page is to be printed.

l The maximum output line length to be used. Passed to the filter as -L.

m Send mail via *mail*(1) when the job is completed.

o overly long lines should be truncated on output. Passed to the filter as -t.

P This job is a piped job, and should be run only if the -i option was given, and the PID given with it matches the one on this line. See also 'F'.

Q This is the QID, which is initially the same as the PID of the lpr that made the control file. (It may be altered by *lpmv*(1)). The job with the lowest QID is printed next. This *must* be the first line in the control file.

p The type of printer this job is being sent to. Passed to the filter as -P.

R The number of copies of this job that should be printed.

s This job is being sent to a sheet feeder, the rest of the string is passed to the filter as the '-S' option.

S This line is a command to be run, whose output will be passed to the filter. (See also the note under 'F'.) The command is parsed, and if it contains no Bourne shell metacharacters, an attempt is made to run it via exec. The environment of the user who started the job is passed to the execed program (or to the shell). If the exec fails, or the command contains shell metacharacters, the cmd is run as "sh -c cmd". The umask is set to that on the 'K' line, and the environment is set up from the file whose name is the same as the control file, except that the first letter of the relative portion of the path name is 'e', instead of 'd'. The working directory is set from the 'W' line. The format of the environment file is:

3 digits which give the length of the variable and its value, including the LF at the end; the variable name followed by an '='; the value of the variable; and a LF terminator. This allows environmental variables to contain an arbitrary string, including 8 bit characters.

t This job is to be printed in transparent mode; the filter should output all characters without altering them or masking them. Passed to the filter as -T.

u A filename (always a full pathname) to be unlinked when the job is finished (successfully or not).

W The directory from which the filter should be run, used only with the 'S' line.

Z The size to be reported for this file. Jobs for which the daemon can not determine the size (such as piped input with **-i**, and 'S' cmds) report this size in the accounting file. For cases in which the size can be determined, this is added to the real size.

FILES

/usr/spool/lpd/pr#
spooler directories (# is the printer number)

/usr/spool/lpd/pr#/[cdle]*
Input and control files.

/etc/devtype
For correlation between printer #, printer port, and printer type, etc.

/usr/adm/lpacct
If this file exists and is writable by root, a line of statistics is written for each job. (The size of the job is given as zero if the job was a shell script, or there was no file AND the -l option was given.

dtinit(1), *lprm*(1), *lpmv*(1), *lpq*(1), *lpdun*(1), *lpr*(1), *lpf*(8), *printcap*(5).

NOTES

See the NOTES section in *lpr*(1) for a description of how error messages are handled.

NAME

lpf - the line printer spooler program which actually does output

SYNOPSIS

lpf -b *banner* **-d -D** *name* **-e -E** *name* **-e** *name* **-f# -F # -I** *string* **-j** *name* **-L # -l# -N** *name* **-o** *owner* **-P** *name* **-s# -S** *name* **-T -t -W**

DESCRIPTION

Arguments fall into three classes: required for proper operation, desirable, and optional. Since this program is not intended to be used directly by humans, error checking for invalid/missing arguments is minimal! The intention is that it be started by the line printer daemon *lpd*(8).

The filter is responsible for extracting printer specific strings and capabilities from */etc/printcap*. The program itself is printer independant. If certain strings (carraige return, linefeed, formfeed mainly) are not found, or the printer type is not found, defaults are assumed (the ascii characters 013, 012, and 014, respectively).

The filter also handles asynchronous signals from the *lpdun*(1) program, requesting that printing be suspended, restarted, pages be reprinted, etc. The program *lprm*(1) also signals the filter, requesting that the job be terminated. The SIGBAND (most system calls are not interrupted) signal is used.

REQUIRED ARGUMENTS

-D *name*

name is the relative portion of the output device name, e.g. tty01 for the SIO port.

-f*n* The file descriptor to be used in reading from the daemon, used to verify that the daemon has changed the prstate file as requested. (The file is opened by the daemon, and the descriptor inherited by the filter.) *n* is ANDED with 077 to determine the actual number.

-l*n* *n* is the file descriptor for the output device/file. (The file is opened by the daemon, and the descriptor inherited by the filter.) *n* is ANDED with 077 to determine the actual number.

-s*n* The file descriptor to be used in sending to the daemon, used to request that the daemon change the prstate file. (The file is opened by the daemon, and the descriptor inherited by the filter.) *n* is ANDED with 077 to determine the actual number.

-N *name*

name is the relative portion of the notification device name, e.g. console. This port is used to notify the operator that human intervention is required, e.g. feeding a new sheet of paper.

-P *name*

name is the printer type to be used. It should be one that is legal, i.e., in /etc/printcap.

DESIRABLE ARGUMENTS

-j *name*

name is the job name. It is used on the header page to identify the job.

-e *name*

name is the tty port from which the job is started. Error messages are sent to this port. If not given, or not writable, /dev/tty is tried; if /dev/tty is not writable, messages go to stderr.

-E *name*

name is the terminal type of the user who started the job. This allows messages to be placed properly on the screen. If not given, type FT is assumed.

OPTIONAL ARGUMENTS

-b *string*

string is printed in large letters in the center of the header page (if any). It is truncated to 10 letters.

-d This option informs the filter that the job is to be printed as single sheets. After each page, the filter will send bells (CTRL G) to the notification port and wait for a signal from *lpdun*(1).

-F *n*

Form length for the job is set to *n* lines. The formfeed string is sent every *n* lines (the default is taken from the state file).

-I *string*

This string contains keywords that match entries in /etc/printcap, optionally followed by '=' and a number. This is how the '+' options of *lpr*(1) are implemented. See the 'I' line of the control file in *lpd*(8) for the exact format.

-L *n*

line length in characters; not significant unless **-t** is also given.

-o *name*

name is printed on the header page at top and bottom. It is usually the owner of the job. The presence of this argument is what causes the header page to be printed. If the **-b** option was not given, this is also used as the banner name.

-S *string*

string is the same as the string after the **-S** option of *lpr*(1). It informs the filter that the job is being printed on a printer with a sheet feeder. It has a special significance if the string is "-1" (that is, 'minus one'), which is NOT allowed by lpr. In this case the filter sends the required strings to the printer to enable a

new sheet of paper to be fed from the sheet feeder. This is normally done by using the **-N** option of *lpdun*(1).

-t truncate lines that are too long (as shown by the **-L** option or the default in the state file).

-T transparent mode. All 8 bits of each character in the input are sent directly to the printer; no processing is done, and no reprinting is possible. Normally, non-printing characters are shown the same as they would be by the *-v* option of *cat*(1).

-W Expect input to be in a special format, with key letters indicating the action to be taken. This option corresponds to the **-C** option of *lpr*(1). The following options are ignored if this option is given: *-t, -T, -L,* and *-F*. Currently, only the word processing formatter uses this format. A listing of the special characters and their meaning follows. The number in parentheses indicates the number of bytes that go with the command.

A Software length change. 2 byte arg (12 bits).

B Bin number change. 1 byte arg (6 bit).

C Software ribbon change. LF terminated string arg.

D Delay. 1 byte arg (6 bit).

E End of page.

F Operator forms change. LF terminated string arg.

H Software character set change. LF terminated string arg.

I Turn off overriding un-interruptible mode.

J Turn on overriding un-interruptible mode.

L Operator width change. 2 byte arg (12 bits).

O Software forms change. LF terminated string arg.

P Operator length change. 2 byte arg (12 bits).

R Operator ribbon change. LF terminated string arg.

S Re-Schedule this job for later.

T Software width change. 2 byte arg (12 bits).

W Operator character set change. LF terminated string arg.

X Transmit n bytes interruptible (count is in following byte)

Y Transmit n bytes un-interruptible (count is in following byte)

Count for 'X' and 'Y' commands:

@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

P	Q	R	S	T	U	V	W	X	Y	Z	[	]	^	_
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47 48

'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

p	q	r	s	t	u	v	w	x	y	z	{	|	}	-	del
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80

'' through 'o'

Transmit n bytes interruptible

'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

'p' through 'DEL'

Transmit n bytes un-interruptible

p	q	r	s	t	u	v	w	x	y	z	{	|	}	-	del
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

FILES

/usr/spool/lpd/pr#

spooler directories (# is the printer number)

/usr/spool/lpd/pr#/prstate

The file containing information about the current state of the printer and the spooler. The *lprm, lpmv,* and *lpdun* programs change this file before they signal the filter, so the filter can determine what request was made. See the functional spec for the spooling system for more details about this file. The following printer states are currently understood:

0	IDLE
1	ACTIVE
2	SUSPEND
3	WAITING FOR SHEET
4	WAITING FOR RIBBON
5	WAITING FOR WHEEL
6	WAITING FOR FORM
7	WAITING FOR LINEWIDTH
8	WAITING FOR PAGELENGTH
10	KILLED
11	INTERRUPTED
12	RESTART
13	RESUME
14	TRANSPARENT (-T option or no pages being saved)
15	SETSHEET (see -S-1)

/etc/printcap
For printer specific information

SEE ALSO

dtinit(1), *lprm*(1), *lpmv*(1), *lpq*(1), *lpdun*(1), *lpr*(1), *lpd*(8), *printcap*(5).

NOTES

See the NOTES section in *lpr*(1) for a description of how error messages are handled.

LIMITATIONS

lpf attempts to save pages in memory, so jobs can be restarted up to 9 pages (the actual number is set by *lpdun*(1); see it's **-m** option) before the current one. If there is a very limited amount of memory, this can result in some thrashing if any pages are being saved. The job will still be printed, but very slowly.

lpf does not know about videotext characters. It will pass through sequences in transparent mode or from Fortune:Word, but in normal printing, they will be shown as '^Yx' where x is the escape character.

NAME

mid - display machine serial ID number and/or group ID number

SYNOPSIS

mid [*-s*] [*-g*] [*-h*]

DESCRIPTION

Mid displays the machine serial number and group number. If either the *-s* or *-g* option is specified then only those requested fields will be displayed as described below; otherwise, both numbers will be displayed prefixed with descriptive labels.

Both the serial number and group number are 32-bit numbers which are displayed in "license plate format" (unless the -h option is specified). This format maps a 32-bit number into an ascii string of the form "nnn-CCC-nnn" where *n* is a digit and *C* is an alphabetic consonant.

OPTIONS

-s display the machine serial number in a format suitable for pipelining, i.e. with no descriptive label and immediately followed by a carriage return.

-g display the group number in a format suitable for pipelining.

-h use hexadecimal format for displaying numbers. If both the -s and -g options are specified then the serial number and group number will be displayed, in that order, in a form suitable for pipelining.

NAME

mkconf - create/change a configuration block

SYNOPSIS

mkconf -c *ifile ofile*
mkconf -floppy *ofile*
mkconf [-i ifile] *ofile*
mkconf -rigid *ofile*
mkconf [-S size] *ofile*

DESCRIPTION

Mkconf allows you to create a configuration block and save it on a regular file or a disk device.

OPTIONS

-c Copies the conf block from *ifile* to *ofile*.

-floppy / -rigid
Allows the user to interactively change fields in the configuration block. Cause a reasonable default configuration block to be created for floppy disks and rigid disks, respectively.

-i Reads the configuration block from *ifile* if available or from *device* and uses that as a starting point.

-S Sets the size of the swap partition to *size*.

SEE ALSO

devctl(2), diskconf(5), rdconf(8).

DIAGNOSTICS

Error messages if the device does not exist, if read or write errors occur, or if one of the files does not exist.

NAME

mkdevs - generate canonical device names

SYNOPSIS

mkdevs -p

DESCRIPTION

Every 32:16 device driver has a unique set of file names for the devices it gives access. *Mkdevs -p* prints out a shell script which creates all of these standard device names in /dev, along with permission bits supplied by the driver. The devices are queried via the *IOCAUTO* command available with *devctl(2)* or *ioctl(2)*.

FILES

/dev/*

SEE ALSO

devctl(2), ioctl(2), intro(4), disk(5).

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs -a special [mn]
/etc/mkfs special size [mn]
/etc/mkfs special proto

DESCRIPTION

Mkfs constructs a file system by writing on the special file *special*. In the first form the size of the file system is the same as the size of the partition and is taken from the disk configuration block. In the second form of the command a numeric size is given and *mkfs* builds a file system with a single empty directory on it. The number of i-nodes is calculated as a function of the file system size. (No boot program is initialized in this form of *mkfs*.)

NOTE: All file systems should have a *lost+found* directory for *fsck*(8); this should be created for each file system by running *mklost+found*(8) in the root directory of a newly created file system, after the file system is first mounted.

The second form of *mkfs* is sometimes used in bootstrapping. In this form, the file system is constructed according to the directions found in the prototype file *proto*. The prototype file contains tokens separated by spaces or new lines. The first token is the name of a file to be copied onto sector zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it is the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the i-list. The next set of tokens comprises the specification for the root file. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a six character string. The first character specifies the type of the file. (The characters **-bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or **-** to specify set-user-id mode or not. The third is **g** or **-** for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see *chmod*(1).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkfs* makes the entries . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token $.

```
4872 55
d--777 3 1
usr    d--777 3 1
       sh     ---755 3 1 /bin/sh
       ken    d--755 6 1
              $
       b0     b--644 3 1 0 0
       c0     c--644 3 1 0 0
       $
$
```

The arguments m and n specify the interleave factor. They default to 3 and 68 respectively in the first form of the command. In the second form they are taken from the configuration block. Recommended values for m and n are, respectively, 3 and the number or 1024-byte blocks per disk cylinder.

SEE ALSO

dir(5), filsys(5), format(8), fsck(8), mkconf(8), mklost+found(8), rdconf(8).

LIMITATIONS

There is no way to specify links.
There is no way to specify bad blocks.
Lost+found is not made automatically.

NAME

mklost+found - make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

A *lost+found* directory is created in the current directory containing a number of empty files. These provide empty slots for *fsck*(8). This command should be run immediately after mounting a newly created file system.

SEE ALSO

fsck(8), mkfs(8).

LIMITATIONS

Should be done automatically by *mkfs*.

NAME

mknod - build special file

SYNOPSIS

/etc/mknod name [**b**] [**c**] *major minor*

DESCRIPTION

Mknod makes a special file. The first argument is the *name* of the entry. The second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g. unit, drive, or line number).

The assignment of major device numbers is partly dependent on the order of I/O option boards in the 32:16 expansion slots. The *mkdevs* program queries each available driver for the "real name" of each device, and this program should never be run. If it is necessary to make an alternate name for a device, i.e. */dev/tar* for */dev/rfd02*, the *ln*(1) command should be used.

SEE ALSO

mknod(2), mkdevs(8).

NAME

Motherboard Rom - the startup program for the Fortune 32:16

DESCRIPTION

The motherboard rom, or *momrom*, is run when the 32:16 is turned on or the reset button is depressed. It is not a disk file, instead it is part of the 32:16 hardware. To get the boot menu, turn on the machine while holding down the Cancel/Del key. The 32:16 contains a small store of non-volatile RAM. The contents remain when the 32:16 is turned off. This storage area, the EAROM (electrically alterable read-only memory), contains information the motherboard rom uses.

When momrom runs, it clears the screen, prints "1", and then does one of three things: runs UNIX, acts as a simple computer terminal, or shows the boot menu. The first is the usual action; this does the normal startup sequence ("2 3 4 5 6 7 8 9" etc.). The second, terminal mode, prints "Terminal mode" on the screen and operates as a simple terminal. It assumes that a remote host is connected to the serial port on the back of the machine. The boot menu allows you to change the action taken and the details of booting up, i.e. whether to boot from the hard disk or the floppy disk, what baud rates the keyboard and back serial ports run at, and similar trivia. For a detailed guide to the boot menu, read *bootmenu*(8)). Also, all of the items in the boot menu can also be changed in the reconfiguration menu programs (*reconf*(8)).

Essentially, the boot menu is a stripped-down version of the reconfiguration menu, which only changes the information in the EAROM used in booting up to a stage where the full reconfiguration menu can be run. The keyboard and the built-in CRT is usually used as the system console. However, both the keyboard and back ports can also, when attached to an FIS 1000, be used as the system console. This can be done by changing the "Console location" field in the reconfiguration menu (normally set to CRT) to TTY00 or TTY01. The next time UNIX comes up, or the machine is powered up or reset, the keyboard port (TTY00) or the back port (TTY01) is used as the console, instead of the keyboard/CRT combination.

If the CANCEL/DEL key is held down on the keyboard right after the machine is turned on or reset, the boot menu appears on the CRT/keyboard console. This overrides the EAROM "Boot action" and "Console location" settings. Also, if an FIS 1000 is attached to the front or back port, and any of the arrow keys are held down immediately after reset or power-up, the boot menu appears on FIS. Until the reconfiguration menu is run, or the machine is again reset or powered up, that terminal is the system console.

NAME

mount, umount - mount and dismount file system

SYNOPSIS

/etc/mount [special name **[-r]]**

/etc/mount -a

/etc/umount special

/etc/umount -a

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The file *name* must exist already; it must be a directory (unless the root of the mounted file system is not a directory). It becomes the name of the newly mounted root.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

These commands maintain a table of mounted devices in */etc/mtab*. If invoked without an argument, *mount* prints the table.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors occur when access times are updated, whether or not any explicit write is attempted.

OPTIONS

-a For either *mount* or *umount*, all of the file systems described in */etc/fstab* are attempted to be mounted or unmounted. In this case, *special* and *name* are taken from */etc/fstab*. The *special* file name from */etc/fstab* is the block special name.

-r Indicates that the file system is to be mounted read-only.

FILES

/etc/mtab
mount table

/etc/fstab
file system table

SEE ALSO

mount(2), fstab(5), mtab(5).

LIMITATIONS

Mounting file systems full of garbage crash the system.

Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

pstat - print system status

SYNOPSIS

pstat [**-acCfimMpstxwW**] [**-d** *dumpfile*] [**-u** *pid*]

DESCRIPTION

Pstat prints miscellaneous information from inside the kernel. The option switches determine what is printed. Most of the options cause *pstat* to print out data tables from kernel memory. Most of the information is only of use to those working on kernel development, but it can also be useful for analyzing system resource usage patterns.

OPTIONS

-a With **-p** or **-t**, print out the whole table, not just active table slots.

-c Print total number of clist blocks, number of clist blocks free, number of characters in free clist blocks, and the freelist wait address. Example:

```
clists: 14 total, 14 free, 1736 chars, 1084 wait address
```

-C Print configuration information, i.e. how much memory is plugged in, and what is plugged in to the option slots. Information about an I/O board in an option slot comes from the ROMs on the board. Example:

```
Total Memory = 1024K bytes
Slot A contains Crt, Version 1.1
Slot E contains Hard boot, Version 0.0
```

-f Print the open file table. Example:

```
100 files.  3 active
LOC FLG CNT INO       OFFS
 15  W    1   4          0
 41 RW    7   3         34
 42 RW    9   9       2886
```

The headings are:

LOC
: The index of this table entry.

FLG
: Miscellaneous state bits:
  H open for hole-skipping read
  P pipe
  R open for reading

W open for writing

CNT
Number of file descriptors referencing this file table entry.

INO
The index of the inode table entry for this file.

OFFS
The file offset, see *lseek*(2).

-i Print the incore inode table. Example:

```
100 inodes.  4 active
LOC  FLAGS    CNT DEVICE   INO   MODE NLK UID  SIZE/DEV
  0           6  1, 18     2  40777  45   0      1152
  1      T    1  1, 18  1239 100770   2 101      6764
  2      T    1  1, 18  1304 101777   1   4     26272
  3           2  1, 18    54  20623   2 101     6,  0
```

The headings are:

LOC
The index of this table entry.

FLAGS
Miscellaneous state bits:
A access time must be corrected
C changed time must be corrected
H hole-skipping read is in progess
L locked
M file system is mounted here
P is a pipe
T contains a text file
U update time (*filsys*(5)) must be corrected
W wanted by another process (L flag is on)

CNT
Number of open file table entries for this inode.

DEVICE
Major and minor device number of file system in which this inode resides.

INO
I-number within the device.

MODE
Mode bits, see *chmod*(2).

NLK
Number of links to this inode.

UID
User ID of owner.

SIZE/DEV
Number of bytes in an ordinary file, or major and minor device numbers of special file.

-m Print total physical memory, amount available for user processes, amount of user memory free, maximum memory allowed per process, number of core freelist slots used out of total number available. Amounts are in increments of 1024 bytes. Example:

```
mem total = 1024K, user = 803K, free = 339K,
perproc = 285K coremap: 4/50
```

-M Print amount of memory available for user processes, total physical memory, and maximum memory allowed per process. Amounts are in increments of 1024 bytes. Example:

```
803K user memory available out of 1024K total,
256K maximum per process
```

-p Print the process table. Example:

```
37 processes.  5 active
LOC S  F PRI SIGNAL UID TIM CPU NI PGRP PID  PPID SIZ WCHAN     LINK TEXTP CLKT
 0 1  3   0      0   0 127 199 20    0   0     0   2 runout       0     0    0
 1 1  1  30      0   0 127   0 20    0   1     0  15 (wait)       0 ff41c    0
 2 1  1  30      0 101 127   0 20   32  32     1  42 (wait)       0 ff432    0
 3 1  1  40      0   0 127  50 30    0  29     1  22 (pause)      0     0   45
 4 1  0  28      0   0 127   0 20    0  33     1  15 tty1,1+56    0 ff41c    0
```

The headings are:

LOC
The core address of this table entry.

S Run state encoded thus:
- 0 no process
- 1 waiting for some event
- 3 runnable
- 4 being created
- 5 being terminated
- 6 stopped under trace

F Miscellaneous state bits (hexadecimal):
- 1 loaded
- 2 the scheduler process
- 4 locked

8 swapped out

10 traced

20 used in tracing

40 locked in by *lock*(2).

PRI Scheduling priority, see *nice* (2).

SIGNAL
Signals received (signals 1-32 coded in bits 0-31),

UID
Real user ID.

TIM
Time resident in seconds; times over 127 coded as 127.

CPU
Weighted integral of CPU time, for scheduler.

NI Nice level, see *nice*(2).

PGRP
Process group number of process (usually process ID of the opener of the controlling terminal).

PID
The process ID number.

PPID
The process ID of parent process.

SIZE
Size of process image in clicks (1K bytes). Does not include shared code segment if sharable.

WCHAN
The event for which the process is waiting or sleeping; if blank, the process is running. This field is either the name of an internal kernel variable, like runout, lbolt, buf[10], inode[3], etc., a system call name such as wait or pause (in parentheses), a tty major/minor dev like tty0,0, any of the above with an offset, as in tty1,1+56, or a hex address in rare cases.

LINK
Link pointer in list of runnable processes.

TEXTP
If text is pure, index of text table entry.

CLKT
Countdown for *alarm*(2) measured in seconds.

-s
Print information about swap space.
Example:

```
Swap allocation size is 512 bytes.
Total = 3264, total free = 2236
Largest free chunk is 2158, swapmap: 38 slots, 5 used
```

-t Print table of tty devices. Example:

```
7 ttys
STATE: T=TIMEOUT W=WOPEN O=ISOPEN C=CARR_ON B=BUSY A=ASLEEP
       X=XCLUDE H=HUPCLS S=STOPPED
DEV   RAW CAN OUT  MODE  DEL COL   STATE   PGRP DISC  LOCAL LSTATE
0,  0   0   0   0     e0  0   0   OC        46 ntty 119405      0
6,  0   0   0   0     cd8 0   4   OC        47 ntty 119405      0
```

The headings are:

DEV
Major and minor of the device.

RAW
Number of characters in raw input queue.

CAN
Number of characters in canonicalized input queue.

OUT
Number of characters in output queue.

MODE
Encoded in hexadecimal. Things like echo, tab expansion, etc. See *tty*(4).

DEL
Number of delimiters (newlines) in canonicalized input queue.

COL
Calculated column position of output.

STATE
Miscellaneous state bits:

A process is awaiting output
B busy doing output
C carrier is believed to be on
H hangup on close
O open
S output is stopped
T output timeout delay is in progress
W waiting for open to complete
X open for exclusive use

Use *stty*(1) for complete state.

PGRP
Process group for which this is controlling terminal.

DISC
Line discipline in use for terminal.

LOCAL
Hexadecimal contents of tty.t_local longword.

LSTATE
Hexadecimal contents of tty.t_lstate longword.

-u print information about a process from its 'upage'; the next argument is the process ID. The 'upage' is a data structure that is part of the data segment of a process and is 2K bytes in this implementation for the Fortune 32:16. *Pstat* finds the upage in main memory or swap space, wherever it is. If the process ID argument is -1 and *dumpfile* is the core file of a process that died, this option prints out the information from the upage of the deceased process represented by the core file.

-w Print what version of the kernel this is and when it was built.

-W Like **-w** but also tells who built the kernel and where.

-x Print the text table with these headings:

LOC
The index of this table entry.

FLAGS
Miscellaneous state bits:
K locked
L loading in progress
T *ptrace*(2) in effect
w wanted (L flag is on)
W text not yet written on swap device

SIZE
Size of text segment, measured in clicks (1K bytes).

INO
Core location of corresponding inode.

CNT
Number of processes using this text segment.

CCNT
Number of processes in core using this text segment.

FILES

/dev/mem
default source of tables, kernel data space

/dev/hd01
this or some other disk will be the swap device

SEE ALSO

ps(1), stat(2), filsys(5).

K. Thompson, *UNIX Implementation*

NAME

/etc/rc - Shell script for system startup

DESCRIPTION

/etc/rc is a shell script run by *init*(8) when the system starts up. If there is a file */tmp/.reboot*, then the system is rebooted. If there is a file */tmp/.powerdown*, then */etc/rc* goes through an orderly powerdown sequence, which runs any product shutdown scripts, prints "Software shut down complete", unmounts any file systems in /etc/fstab (to prevent file system damage), prints "Hardware shut down starting, please wait 60 seconds" places a marker for future use that the file systems are in order, and runs */etc/halt*, which merely waits for 60 seconds. After 60 seconds of no disk activity, the disk heads are moved to a safe place, otherwise shutting off power could destroy data.

If neither of the two files exist, */etc/rc* attempts to bring up the system. It prints the number "8" in the countup sequence (see *countup*(8)), builds all the proper special devices (see *mkdevs*(8)), and then prints "9". Next, the screen is cleared and the user is given 5 seconds to hit the DEL key. If he does, then */etc/rc* goes into maintenence mode and the user is allowed to log in as 'root'. The user then sets the time and date.

If the file */etc/.fsclean* does not exist, it assumed that the system was not brought down with */etc/shutdown* but instead crashed in some fashion. Acting on this assumption, */etc/rc* runs 'fsck' to clean up any possible file system damage.

Lastly, any product startup scripts (/m/rc/*.rc) are run.

FILES

/tmp/.reboot

/tmp/.powerdown

/m/rc/*.rc

/etc/.fsclean

/etc/mtab

/etc/utmp

/etc/ptmp

/dev/console

SEE ALSO

fsck(8), login(1), mkdevs(8), mount(8), pstat(1), setdt(8), sh(8), sync(8), update(8).

NAME

rdconf - read configuration block

SYNOPSIS

rdconf [**-T** | **-b**] *device...*

DESCRIPTION

Rdconf reads the configuration block from each *device* and prints it's contents on the standard output in a human readable form.

Device can take several forms and it should be noted that all the programs which manipulate disk drives via *devctl(2)* use this same form. If *device* is of the form

- { **b** | **c** } *major* / *minor*

then the appropriate block or character (b or c) device with the respective major/minor pair of numbers will be used in the *devctl* call. If *device* does not match this form then it is assumed to be either the name of a device special file or a data file in which case either a *devctl* is made on the device or the file is read; this way files can be used to store configuration blocks and used anywhere a device could be used.

OPTIONS

-b causes an image of the configuration structure to be output instead of the human readable text.

-T prints out only the typename from the configuration block. Checks are made to see if the configuration block is legitimate and if not diagnostics are produced; this does not, however, prevent the information in the block from being interpreted and printed.

SEE ALSO

mkconf(8), devctl(2), diskconf(5).

DIAGNOSTICS

Error messages if a device does not exist or if a read error occurs on a device.

NAME

reboot - reboot the system

SYNOPSIS

reboot [**-n**]

DESCRIPTION

Reboot takes down the Fortune System and brings it back up. It asks you to confirm that you want the system to reboot, if the *-n* flag is not given.

OPTIONS

-n Does not ask you if you want the system to reboot.

DIAGNOSTICS

Reboot failed, you must be super-user.

WARNING

Reboot does not issue a sync(1) command.

SEE ALSO

sync(1).

NAME

setdt - set date and time

SYNOPSIS

/etc/setdt [e][t#]

DESCRIPTION

Setdt reads the current time with time(2) and puts it on the screen in human-readable format to be edited. If the time entered is earlier than the current time or more than five days in advance of the current time, the user is told that the time appears to be incorrect and is asked to verify that it is the correct time. When *setdt* determines the user has entered a reasonable time, or has confirmed the time entered is correct, it prints the time and asks the user to verify that they have entered the correct time. If the Cancel key is pressed any time while the program is running, the current time is printed in the format shown above and the program terminates.

OPTIONS

e Indicates that European format is desired, rather than American format. The 'f' or 'F' keys can be used to toggle between formats once the program has been invoked.

t Allows the user to give a time limit in seconds after which the program should time out, e.g. 'setdt t30' gives the user 30 seconds to edit the date, and if the program is still running after 30 seconds it is sent signal 1 (alarm clock). If this option is used it must be the last option given.

SEE ALSO

date(1), stime(2), time(2), ctime(3).

LIMITATIONS

If a yes/no question is answered with "no<EXECUTE>" or "n<EXECUTE>", this is interpreted as a response of "yes".

An hour of "00" is accepted as a valid time in American format.

The 'f' and up-arrow keys are not accepted if the cursor is on the right side of the date or time fields.

NAME

setnswap - set the 'nswap' variable in the kernel binary file

SYNOPSIS

setnswap partition-number confblock-file unix-file

DESCRIPTION

Setnswap reads the configuration block from *confblock-file*, usually a disk device, and determines the number of blocks in *partition-number*. It patches this number into the 'nswap' variable in *unix-file*.

LIMITATIONS

Setnswap assumes the sector size is 512.

NAME

shutdown - orderly shutdown

SYNOPSIS

shutdown

DESCRIPTION

Shutdown takes down the Fortune Operating System in an orderly fashion. It asks you to confirm that you really want the system to quit.

NAME

sync - update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to insure all disk writes have been completed before the processor is halted.

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), halt(8), reboot(8), update(8).

NAME

/etc/uconf - change basic system parameters

SYNOPSIS

/etc/uconf - UNIX program
/sa/reconf - Stand-alone program

DESCRIPTION

The reconfiguration menu allows you to use the 32:16 with its various peripheral devices. The various fields allow you to decide what the system will do when it powers up, and the size of various operating system resources. */etc/uconf* can only be run by the system manager. */sa/reconf* operates in generally the same way as /etc/uconf, except it is a stand-alone program, i.e. it runs before UNIX comes up.

To run it you must hold down the CANCEL/DEL key when the system is powered up or rebooted, and change the "Boot file name" (F7) field to *disk*/sa/reconf, where disk is probably "hd02" for the hard disk or "fd02" for the floppy disk. Type EXECUTE and the reconf menu eventually appears. To leave the menu, type F3, and the level 1 boot program (the ":" prompt) re-appears and you can then bring in UNIX (*disk*/unix). (See *boot*(8), *bootmenu*(8), and *momrom*(8).)

The menu consists of fields and values for the fields. Upon entry to the program the current value of the fields as stored in the EAROM is displayed.

Certain of the fields simply have a list of possible values that are cycled through by pushing the space bar. Others require the data to be manually entered; for these entries, the space bar either has no effect or enters a space character in the field. In either case, simply type the <RETURN> key to move on to the next field.

Certain of the fields also have associated with them "Used/Total" information. In any case, that information is not crucial, so it can be ignored.

The information in some of the fields may also be in overstruck. See the description of "Set params auto?".

Field Definitions

Power up action: Decide what the machine will do when you turn it on. It can boot from the "Boot Device", show the menu, or act as a terminal, with another computer connected to tty01.

Boot Device: Decide from which peripheral to boot. The 32:16 can boot from the back port, the floppy disk drive, or some of the Fortune peripheral options, such as the hard disk controller.

Boot drive #: Most of the disk drive peripherals can have more than one drive attached to them, and the drives have unique numbers,

starting from 0. This sets which actual drive will be used on the above-named *boot device*.

Boot Program: Each disk drive can have up to eight *boot programs*. This number specifies which boot program will be used.

Boot file: This is a file name in the UNIX file system in the boot partition (usually hd02/unix or fd02/unix).

Flex drive #1, #2, #3, #4: The 32:16 can have up to four floppy disk drives hooked up to it. This allows you to indicate which model of floppy disk drive is connected to each of the four channels.

TTY00 port speed & TTY01 port speeds:
The Fortune keyboard port (TTY00) and the port on the back of the machine (TTY01) are RS-232C serial lines. These can operate at various speeds. The Fortune keyboard should be at 2400 baud. The back port can be used for various purposes and the speed can be set up for whatever it is used. These are default speeds only. They are in effect whenever the MomROM functions as a terminal, and also under UNIX until they are changed. TTY00 can be set to any baud rate if that port is to be used for operation with other than the Fortune integral keyboard.

Console location: You may want to run a standard video-display terminal or even a printing terminal as the console. This item allows you to use tty00 or tty01 for this purpose. Typically, you would figure out an appropriate speed for the terminal, and plug it into tty01.

Timezone: This allows you to tell the UNIX time-keeping system about your timezone. All of the world's timezones (except Saudi Arabia) are available here, in either verbal or numerical form.

Daylight savings: This field allows you to tell the UNIX time-keeping system whether or not daylight savings time ever applies in your hemisphere; the system uses Daylight Savings at the appropriate time of the year.

Line frequency: Sets the line frequency of the of the ambient AC power supply. If you are in America, this is 60 hertz; otherwise, it is probably 50 hertz. If this is set wrong the integral CRT will flicker.

Language: The language field indicates the natural language used in this machine's application software. It controls the LANGUAGE= environment variable.

Floating point?: The 32:16 supports floating point software using a large emulator in the kernel, and some of the software distributed by Fortune needs this package. You should set this field to YES unless you know that you don't need it.

Hex number: This field is not used in the 2.0 release, but is included for future use.

Now we come to the really interesting part of the menu. The next eight fields allow you to change how well UNIX performs under various loads. If you are confused by any of this, change the "Set params auto?" field to YES. It automatically calculates generally acceptable values to use for the fields that are overstruck when the "Set params auto?" field is set to YES.

Number buffers: Unix performs disk-handling through intermediate in and out boxes called buffers. The more file-handling (disk activity) occurs, the more the buffers are used. If you are performing much activity on a certain few disk files, for instance a data-base, increasing the number of buffers might have cause an improvement.

Number inodes: There is an in-core inode required for every *active* (open) file, and the number of inodes limits the number of active files. These in-core structures are also another kind of buffer cache. If you are dealing with many files, increasing the number of in-core inodes can also have an effect on performance. It should also be increased if the "Software Error 142" message ever appears on the console.

Number files: There is a also file structure in core for every open file. The number of file structures limits the total number of files that can be open at one time. Having more of them will not help performance. But not having enough of them could make it so you can't get anything done. Increase this number if the "Software Error 129" message ever appears on the console.

Number clists: Clists are to terminals as buffers are to disk files. Increasing these to large amounts is only useful when running the 32:16 with much I/O activity to printers, networks, or terminals other than the integral CRT.

Number processes: Every program that runs is a process. Many programs, to work effectively, use 2 or more processes. and the 32:16 can only run so many processes. If you try to run too many programs at once, you will get the message "No more processes" (or something similar) from the shell; this means you need room for more processes.

Max process size: The largest amount of memory (the most critical system resource) that a process can use up. The number you enter is multiplied by 1024 bytes.

Set params auto? If this is set to YES, the operating system ignores the above eight fields, and instead uses rules of thumb. It does a pretty good job, and you should only need to control these fields if you always use your system in a certain way. It bases its estimation of the above fields based on the total memory on your system and on your estimate of the number of users that will be using the system. If this field is set to YES, the values of the fields are displayed

crossed out, meaning that those values are ignored and the automatically calculated parameters used instead.

Appx. # of users: This is a guess as to how many users you will have using your 32:16. This number is useful to UNIX in "Set params auto?". Note that if you want to just increase everything proportionately, but aren't really increasing the number of users on the system, you can still just make this number larger. The numbers which are ignored when this field is set to YES are struck out in the menu, when the field is so set. This field does not control how many people are allowed to log in.

/etc/uconf also displays certain system resource usage parameters: the ram usage, swap space usage, and the actual system parameters listed above "Set params auto" which can be changed. These numbers are already tuned to run well with a general mix of applications, but if you wish to fine-tune these numbers for a specific type of computing load, this display is of aid. Please note that setting any of these too low crashes UNIX or otherwise hinders the system from working well. Also, the *pstat(1)* program helps identify performance problems.

Saving your changes, or what does that bottom line mean?

The settings in the configuration menu are stored in a special kind of computer memory called an EAROM (electrically alterable read-only memory). This memory has two levels: one whose contents goes away when the computer is turned off (known as volatile memory), and another one whose contents is permanent (non-volatile). /sa/reconf allows you to save your changes in one or the other; /etc/uconf only saves things in the permanent area, and these changes do not take effect until Unix comes up again.

The *F1* key puts your changes in non-volatile memory, so that whenever you turn the system off and on, the changes stay in effect. There is an important problem here: Every time that you SAVE your changes with *F1* into non-volatile memory, the EAROM degrades. The "EAROM has been changed # times" line at the bottom records only this, not changes to the volatile part. After the EAROM has been changed around 1000 times, it can fail. When this happens, a flashing message (see below) appears at the bottom of the screen, This does not necessarily mean that the EAROM has gone bad, it may also mean that the data in the EAROM may not have been properly maintained.

The *F2* key throws away any changes you may have done, and reads what was in the EAROM when you last powered up. This is useful if you made a change and didn't really mean it.

In the stand-alone version, typing the *F3* key causes the changes you made to be used on this boot-up only, they are not saved in the

non-volatile portion of the EAROM, and the level 1 boot prompt (":") appears. Turning the 32:16 off and on again (or pushing the reset button) causes the configuration menu to revert to its old state. In the stand-alone version, the *F4* key indicates that the contents of the screen are used on the next boot-up only, but the program goes back to the beginning of the boot-up process (before the "1" appears) instead of merely going back to the level 1 boot prompt (":").

In /etc/reconf, the *F3* key merely exits the program and *F4* does nothing.

FILES

bootmenu(8)

SEE ALSO

boot(8), bootmenu(8), momrom(8), pstat(8).

DIAGNOSTICS

EAROM checksum error!

This message appears when the earom has either gotten old and started to malfunction, or it was accidentally altered by a program malfunction. The F1 key should make this error go away. If it doesn't, consult your dealer. A checksum is computed from the data in the EAROM and is then stored in the EAROM. If this checksum is incorrect, the EAROM may have shuffled off this mortal coil, or may have been improperly altered by a program bug.

NAME

update - periodically update the super block

SYNOPSIS

/etc/update

DESCRIPTION

Update is a program that executes the *sync*(2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file. This program has been subsumed into the kernel, and does not need to be run.

SEE ALSO

sync(1), sync(2), init(8).

LIMITATIONS

With *update* running, if the machine is powered down just as the *sync* is executed, a file system can be damaged.