MOTOROLA, INC.
Computer Group
Computer Systems Division
100 Passaic Avenue
One Greenbrock Corporate Center
Fairfield, New Jersey 07006

UNIX® System V/68 and V/88 Release 4

System Administrator's Guide

Part 1: Chapters 1 through 7

R40GUSAG/D2

Preliminary Documentation Packaging

TRADEMARKS

Delta Series, DeltaSERVER, and M88000 are trademarks of Motorola, Inc.
Epson FX-86e is a trademark of Epson America, Inc.
HP and LaserJet are registered trademarks of Hewlett-Packard Company.
IBM is a registered trademark of International Business Machines.
Motorola and the Motorola symbol are registered trademarks of Motorola, Inc.
PostScript is a registered trademark of Adobe Systems, Inc.
Proprinter is a trademark of International Business Machines.
STREAMware is a trademark of INTERACTIVE Systems Corporation.
Teletype is a registered trademark of AT&T.
UNIX is a registered trademark of UNIX System Laboratories, Inc. in the U.S.A. and other countries.

Contents

DRAFT COPY
February 3, 1992
File: MasterToc

DRAFT COPY
February 3, 1992
File: MasterToc

DRAFT COPY
February 3, 1992
File: MasterToc

DRAFT COPY
February 3, 1992
File: MasterToc

DRAFT COPY
February 3, 1992
File: MasterToc

DRAFT COPY
February 3, 1992
File: MasterToc

DRAFT COPY
February 3, 1992
File: MasterToc

Figures and Tables

About This Document

DRAFT COPY
January 31, 1992
File: Cabout

Introduction

This book has been designed to help you do the work of a system administrator for a computer running UNIX® System V/68 or V/88 Release 4 on a Motorola platform. You may be the owner of a small business, personally maintaining and overseeing the operations of a single computer. Or you may be an administrator for a large organization in which many users share a network of computers. In either case, this guide will help you install and maintain various services on your system, and serve the needs of your users.

Among the new features introduced in UNIX System V/68 or V/88 Release 4 are many new software tools for administration, including a new version of the system administration menus. These tools will help you install your machine and software, set up the resources and environments that best fit the needs of your users, do routine maintenance procedures, and provide emergency troubleshooting service.

How This Guide Is Organized

This guide has been designed to allow you to find all the information you need about a particular area of administration in one place. Each chapter covers a discrete administrative function, such as file system administration, security, or the backup service. In addition, appendixes, a glossary, and an index are provided to make this guide easy to use and understand.

Organization of the Chapters

The chapters are arranged in alphabetical order, like the topics of administration presented on the main system administration menu (the menu that appears on your screen when you enter the `sysadm` command).

Organization of Each Chapter

Each chapter describes the software associated with a function, and provides instructions for performing that function. For some functions, UNIX System V/68 or V/88 Release 4 provides a user-friendly menu interface that can help you do administrative functions without using UNIX system shell commands. This interface is accessed through the `sysadm` command. For functions for which this interface is available, you will see instructions for invoking the appropriate menu at the beginning of the relevant chapter. Because the menus (and other screen messages provided with them) are self-explanatory, detailed instructions about how to use a menu that you have accessed are not included in each chapter.

Instead, Appendix C of this guide, "Using the `sysadm` Interface," provides a sample walk-through for one menu, and defines all the components of the menu system. In addition, the interface itself provides on-line "help messages" that you can access while using the menus through the `sysadm` command.

Each chapter provides instructions for accessing the appropriate `sysadm` menu, and a table listing those shell commands that can be used in place of menu options.

Notation Conventions Used in This Guide

This section describes the notation conventions used in this book.

- References to literal computer input and output (such as commands entered by the user or screen messages produced by the system) are shown in a monospace font, as in the following example:

```
$ ls -l report.oct17
-rw-r--r--   1 jim     doc       3239 May 26 11:21 report.oct17
```

- Substitutable text elements (that is, text elements that you are expected to replace with specific values) are shown in an italic font, as in the following example:

  `$ cat` *filename*

  The italic font is a signal that you are expected to replace the word *filename* with the name of a file.

- Comments in a screen display-that is, asides from the author to the reader, as opposed to text that is not computer output-are shown in an *italic* font and are indented, as in the following example:

```
                .
                .
                .
        command interaction
                .
                .
                .
                .
Press RETURN to continue.
```

- Instructions to the reader to type input usually do not include explicit instructions to press the RETURN key at the appropriate times (such as after entering a command or a menu choice) because this instruction is implied for all UNIX system commands and menus.

In one circumstance, however, an instruction to press the RETURN key is explicitly provided: when, during an interactive routine, you are expected to press RETURN without having typed any text, an instruction to do so will be provided, as follows:

```
Type any key to continue: RETURN
$
```

- Control characters are shown by the string CTRL-*char* where *char* is a character such as "d" in the control character CTRL-d. To enter a control character, hold down the CTRL key and press the letter shown. Be sure to type the letter exactly as specified: when a lower case letter is shown (such as the "d" in the example above), enter that lower case letter. If a character is shown in upper case (such as CTRL-D), you should enter an upper case letter.
- The system prompt signs shown in examples of interactive sessions are the standard default prompt signs for UNIX System V/68 or V/88 Release 4
  - the dollar sign (`$`) for an ordinary user
  - the pound sign (`#`) for the owner of the `root` login

How to Use This Guide

This book has been designed to help anyone doing administrative tasks on a computer running UNIX System V/68 or V/88 Release 4. Specifically, it has been written to help you understand the job of an administrator and find out exactly how to set up, configure, and maintain UNIX System V/68 or V/88 Release 4 on a Motorola platform.

The guide assumes that you know how to enter commands at a computer terminal, and that you have an awareness of such UNIX system fundamentals as the directory structure and the shell. You should also feel comfortable using the computer itself; you should know how to turn it on and how to install peripherals (such as modems, terminals, and printers) for it. To familiarize yourself with your computer and set it up for administration, see your computer installation manual and the documentation about the peripherals that came with your computer. You may also want to refer to the *Product Overview* for descriptions of other manuals in this documentation set that might be helpful to you as an administrator.

NOTE

Some of the AT&T 3B2 SVR4 documentation set guides include relevant manpages. In this documentation set, all manpages may be found in the appropriate Reference Manual.

The Reference Manual pages are divided as follows:

- User's Reference Manual: Section 1 and all subsections (except 1M).
- System Administrator's Reference Manual: Sections 1M, 7 and 8.
- Programmer's Reference Manual: Section 2 through 6.

New Administrators

If you have no experience as a UNIX system administrator, use this guide as a textbook for the work you are undertaking. Begin by reading Chapter 1, "Overview of System Administration." This chapter describes the duties of an administrator, suggests how to organize those duties, and tells you where, in this guide, to find more information about each of those duties.

Then you can read individual chapters to learn about those areas of administration with which you need to familiarize yourself. All administrators need to do many of the tasks described in this book. Some activities, however, may or may not be required for your system, depending on your site, your resources, and your customers. Read those sections of this book that are useful for your needs.

Experienced Administrators

If you are an experienced administrator and you know what information you need to gather and what questions you need to have answered, use this guide as a reference book. You will probably be faced with a task for which you want detailed instructions. Begin your search for the instructions by perusing the table of contents (and, if necessary, the index) for the topic you need. To find out whether there's a menu interface available for your task, see the first page of the appropriate chapter or look in Appendix C, "Using the `sysadm` Interface." This appendix contains a complete list of the menus and tasks included in the system administration menu interface.

If You Use the Menus

If you decide to use the menu interface when you do a particular task, you'll need to find out how to access it. See the first page of the chapter that discusses the area of administration associated with your task. Once you've accessed the appropriate menu and you want to find out how to use it, see Appendix C, "Using the `sysadm` Interface."

If You Do Not Use the Menus

If you decide not to use the menu interface, continue reading the chapter until you find instructions for the administrative task you want to perform. These instructions will specify the running of shell commands.

1 Overview of System Administration

DRAFT COPY
February 1, 1992
File: Cover

Introduction

NOTE Most of the work described in this book can be done only by a user who is logged in as `root`. Therefore you must use this login name whenever you're going to do administrative tasks. The explicit instruction to log in as `root` is not included in every administrative procedure; we assume, throughout this book, that you have logged in as `root`.

This chapter presents a general job description for you, the system administrator: it describes, in broad terms, the tasks for which you are responsible, and lists the documentation where you can find procedures for doing those tasks. Almost all the procedures in this book can be done by issuing shell level commands. The descriptions and procedures presented in each chapter specify the appropriate commands.

For many types of work, however, you have a choice of working on the shell command line or working with an interface composed of menus and forms for system administration. Because these menus and forms are invoked by executing the `sysadm` command, we refer to them collectively as "the `sysadm` interface."

On the following page you will find a one-page summary of the commands you need to know to start using the `sysadm` interface. For a more detailed description of how to use these menus and forms, see Appendix C of this guide, "Using the `sysadm` Interface."

Quick Reference to the `sysadm` Interface

Function Keys

The main tool for manipulating the interface is a set of eight function keys. Labels highlighted in reverse video at the bottom of the screen show the function assigned to each; the functions assigned to some keys change for different types of frames, but [F1] is always mapped to [HELP].

If your function keys do not seem to work, you can simulate them using the two-character sequences [CTRL-f] [1] through [CTRL-f] [8]. The [CANCEL] function key dismisses the current frame (except for the main menu, which cannot be canceled). The [CMD-MENU] function key provides a Command Menu of other useful commands.

Menus

To move between menu items, use the down arrow [↓] and up arrow [↑] keys. To select a menu item, use the [ENTER] key or the [ENTER] function key.

Forms

To move to the next field, use the [TAB] key or arrow keys. After filling in a form, press the [SAVE] function key to process the data entered.

Text Frames

A text frame contains more than one logical page of text if the scroll bar on the right frame border contains a caret ^ at the top or a v at the bottom; use the [NEXTPAGE] and [PREVPAGE] function keys to move between these pages.

Command Line

To go to the command line, use the [CTRL-]] or [CTRL-f] [c] character sequence. Any command from the Command Menu can be typed directly here; press [ENTER] to process the command and return to the current frame. Use the `refresh` command to redraw a corrupted screen and the `cleanup` command to dismiss most frames from a cluttered screen.

Exiting from `sysadm`

To exit from the `sysadm` interface, press the [COMMANDS] function key and select the `exit` item, or go to the command line and type `exit` [ENTER]. (The [CANCEL] function key is not equivalent to `exit`.)

See Appendix C for complete information on using the UNIX System V/68 or V/88 Release 4 `sysadm` interface.

The Job of the Administrator

The job of a system administrator is to provide and support computer services for a group of users. Specifically, it's the administrator's job to do the following:

- set up the computer system, including hardware and software
- allocate resources among users
- optimize the use of software resources
- protect software resources
- do routine maintenance chores
- repair defective hardware and software as problems arise

The rest of this section describes the specific tasks associated with each of these broadly defined areas of responsibility.

Setup of Hardware and Software Resources

The checklist below summarizes the steps you need to take when setting up your computer for the first time. When a reference contains a chapter title without a book title, the reference is to a chapter in this book.

Step	Task	Documentation
1	Install the computer	Your computer installation manual
2	Install, connect, and set up the console terminal and peripheral devices	Your computer installation manual, your terminal manual, the "System Setup" chapter, and the "Storage Device Management" chapter
3	Install and connect the console printer	Your terminal manual and your printer manual
4	Install the Boot/ Install tape	*Binary Installation Instructions*
5	Set up ports	"Service Access" chapter
6	Connect printers, terminals, and modems	"Print Service" chapter
7	Customize the system profile (optional)	"User and Group Management" chapter
8	Create groups for users (optional)	"User and Group Management" chapter
9	Assign user logins and passwords	"User and Group Management" chapter
10	Set up a network (optional)	*Network User's and Administrator's Guide* and the "Network Services" chapter

The rest of this section describes the tasks shown in the table and lists the books in which you can find instructions for them.

Steps 1-3: Install the Computer, Console Terminal, and Console Printer

Your first task is the physical installation of your computer, the console terminal, and, if you're planning to have a dedicated printer for the console terminal, the console printer. Begin by installing your computer, following the instructions in the installation manual delivered with it.

Next, install the console terminal and connect it to your computer, as instructed in the terminal manual. Turn on the terminal and set the options for it, as described in the "System Setup" chapter of this book.

To make record keeping more convenient, you may want to hook up a printer to the console terminal for use exclusively by you. If you decide to do this, set up your console printer now, following the instructions in your printer installation manual.

Power up the computer according to the instructions in the "System Setup" chapter.

Install Data Storage Devices

One important category of peripheral devices is storage devices, such as disk drives and cartridge tape drives (also known as block devices and character devices, respectively). These devices allow users to record data on removable storage media. To learn how to install storage devices, see the "Storage Device Management" chapter of this book. (The latter chapter also includes instructions for formatting, copying, and using—as mountable file systems—removable storage media.)

Step 4: Install the Boot/Install Tape

Because the connections between your computer and peripheral devices must be made through the software as well as through the hardware, it's a good idea to install the Boot/Install tape (the basic UNIX system software) next.

Once you have physically installed the hardware and are running the basic UNIX system software on your computer, you need to complete a procedure that will involve answering several questions, such as the following:

- What is the name of this computer?

- If this computer is going to be part of a network, what is its node name?
- What's today's date? What's the current time?

The information provided in your answers will be used frequently by the operating system during daily operations.

To do this procedure, execute the `setup` command. For details, see the "System Setup" chapter and `setup`(1M) in the *System Administrator's Reference Manual*.

Next, install any software packages that you want to make available to your users, such as the Editing Utilities package. Instructions are in the "Software Management" chapter.

Step 5: Set Up Ports

Enable data connections that can be used to log in on your computer. For instructions, see the "Service Access" chapter.

Step 6: Connect Printers, Terminals, and Modems

Now you are ready to connect terminals and other peripheral devices, such as modems and printers, to your system. These connections are made through outlets on your computer called I/O (input/output) ports. Before you can use a port, you must allocate it for use by a particular device. Instructions for doing this are in the "Service Access" chapter of this book. Once you have allocated the ports on your system, connect your terminals, printers, and modems. For details about installing printers, see the manual for your printer and the "Print Service" chapter of this book.

Step 7: Customize the System Profile

Your system is delivered with a default system profile (`/etc/profile`) that defines the basic operating environment for the users on your system. If you want to change any of the parameters of this profile, see the instructions in the "User and Group Management" chapter.

Step 8: Create Groups for Users

Users frequently want to share data but allowing them to do so without restrictions is unadvisable because system security may thus be compromised. To protect data while allowing those users who need to share data to do so, the system allows you to assign users to groups. Once a user has been assigned to one or more groups, the members of those groups are automatically granted permission to use the data in any directories or files created by that user.

To make sure that a group assignment is available for every new user account on your system, even if you do not create groups, UNIX System V provides a file called `/etc/group`. If you do not create groups before assigning user login names and passwords, all new users will be assigned to the group `other` by default.

If you want to create user groups for your system, see the "User and Group Management" chapter for instructions.

Step 9: Assign User Logins and Passwords

Now you are ready to start letting people use the system; you need to create user login names and passwords. There are two reasons for doing this. First, as a security measure, UNIX System V will not allow anyone to log in on your system without a login name and a password.

Second, by creating a login name for someone, you are also giving that person an account on your system. Ownership of an account affords not only access to the computer's resources, but also a working environment defined by the system profile and user profile you provide.

To find out how to assign user login names and passwords, see the "User and Group Management" chapter.

Step 10: Set Up a Network

If you want to enable your users to communicate with users on other computers, and to use the resources (such as printers) available on those computers, you can connect your system to others through a network. See the *Network User's and Administrator's Guide* and the "Network Services" chapter of this book.

Allocation of System Resources

The job of allocating resources implies two tasks: providing and adjusting those resources and overseeing access to them. Both these tasks must be done on an ongoing basis, as the need for them arises.

The first task is to provide resources. Specifically, you can do the following:

- Give users more space in which to work by creating and mounting new file systems. For instructions, see the "File System Administration" chapter.
- Give users additional space for storing data by adding peripheral devices such as disk drives and cartridge tape drives. For instructions, see the manual for the device you are installing.
- Install software packages. For instructions, see the "Software Management" chapter.

Your second responsibility for resource allocation is controlling who has access to your computer. You can do this by assigning login names and passwords to only those authorized by you to use your system. This task is one of the administrator's first jobs when setting up a system, as described under "Step 9: Assign User Logins and Passwords" in the previous section. Unlike other initial setup tasks, however, assigning user login names and passwords is a job you will have to do from time to time, as personnel changes in your workplace demand. For instructions on assigning logins and passwords, see the "User and Group Management" chapter.

For tips about how to keep your system secure by using login names and passwords judiciously, see the "Security" chapter.

The UNIX System V/68 or V/88 Release 4 process scheduler is tuned to provide good performance over a wide range of computing environments. If you have special requirements for process scheduling, however, the scheduler offers great flexibility. For time-sharing processes, you can control how the scheduler assigns priorities and time slices to user processes. You can also configure real-time processes, which allow privileged users direct control over the order in which processes run. See the "Process Scheduling" chapter for details.

Optimized Use of Software Resources

To make the most efficient use of your system, it's a good idea to track, at regular intervals, how resources are being used and how well your system performs in response to users' requests. If your analysis reveals the system is not running as efficiently as possible, you may want to implement performance improvement measures.

In addition, there may be times when the system response slows down noticeably. If this happens, you will need to investigate possible reasons for performance degradation. Such slowdowns are generally caused by either memory bottlenecks or I/O bottlenecks.

UNIX System V provides a set of tools called the System Performance Analysis Utilities (or SPAU) which, when used with other basic UNIX system utilities, allow you to detect possible performance problems.

These utilities can be classified broadly into two sets. One set of utilities reports information about system activity that is being recorded continuously in the operating system kernel. These tools can be used to do the following:

- collect system activity data, automatically and on demand
- display system activity reports on activities such as CPU utilization, paging, buffering, disk activity, queue activity, and tty activity
- time a command and look at system activity during the execution of that command

A second set of utilities reports statistics about disk and file system usage that are gathered only by request. These tools can be used to do the following:

- report disk access location and seek distance
- track files based on size or age
- print the number of free file blocks and inodes
- summarize file system usage

With the information gathered with these tools, you may decide to change the configuration of your system in ways that can improve the response time and overall efficiency of your system. For instructions on using the SPAU utilities, see the "Performance Management" chapter.

Another set of tools, the Accounting Utilities, allows you to check system usage by providing answers to questions such as the following:

- Who is using which resources?
- What patterns of command usage can be identified? For example, which commands are most frequently used?
- How much disk space is being used by which users?

The Accounting Utilities can also be used to calculate billing charges for use of computer resources.

For instructions on using these utilities, see the "Accounting" chapter.

Protection of System Resources

As the system administrator, you are responsible for protecting the data and software on a computer. Here are a few procedures that can help you do this.

- Identify administrative and system functions that should be done only by the administrator (or an authorized delegate) and assign passwords to these tasks. You can do this during the initial setup procedure described in the "System Setup" chapter. You can always change or add to the passwords assigned during that procedure, however. For instructions, see the "User and Group Management" chapter.
- Set up a regular schedule for backing up (copying) the data on your system. Decide how often you must back up various data objects (full file systems, partial file systems, data partitions, and so on) to ensure that lost data can always be retrieved. See the "Backup Service" and "Restore Service" chapters for instructions.
- Control the permissions codes (which restrict access) for important administrative directories and files. You can do this during the initial setup procedure described in the "System Setup" chapter. You can always change or add to the permissions assigned during that procedure, however. For instructions, see the "User and Group Management" chapter.

For general guidelines about how to make sure your system is secure from intrusion, data corruption, and data loss, see the "Security" chapter.

Routine Maintenance

Finally, from time to time you will need to do certain administrative tasks to ensure that your system continues to function in a healthy way.

- Do backups of your system at regularly scheduled intervals. See the "Backup Service" chapter for a discussion of the various types of backups that can be done and suggestions about how often each type should be done.
- Make sure your software is up to date. When new releases of software used on your system become available, install them to provide your users with the best possible tools. For installation instructions, see the "Software Management" chapter.
- Clean the heads on your tape drives regularly.
- Monitor the space available on each of your file systems. If the available space on a file system gets too low, you will have to take some action to increase the space. The possibilities include moving files from a full file system to a file system with more space, emptying or truncating system log files, and asking users to delete unnecessary files. See "Maintaining a File System" in the "File System Administration" chapter.

Some of these tasks can be done only when your computer is operating in firmware state or after it has been powered down. Thus you will need to change the system state of your computer on a regular basis. To save time, you may want to define a default program capable of booting the system, powering down and rebooting the system, and entering firmware state. The "Machine Management" chapter of this book explains how to create such a program.

Repairs of Defective Hardware and Software

An important function of a system administrator is to identify and fix problems that occur in both the hardware and software while the system is in normal use. The UNIX system provides a set of tools that allows you to pinpoint hardware malfunctions. These tools, along with a few troubleshooting suggestions, are described in the "Diagnostics" chapter. This chapter also explains how to handle bad blocks on the hard disk.

Software related problems are handled separately. As the administrator of your system, you must become familiar with the file system so that you can do the following:

- investigate and repair software related errors on a specific file system
- monitor disk usage for all file systems
- track files based on age or size
- create new file systems
- mount and unmount file systems

See the "File System Administration" chapter for instructions.

Guidelines for Good Customer Service

As a system administrator, you are responsible for providing the best possible service to your customers, the users on your system.

Maintaining a System Log

If your system supports multiple users, we strongly recommend that you keep a record of system activities in a log. A system log book can be a valuable tool when troubleshooting transient problems or when trying to identify patterns in the way your system operates and is used. Therefore we recommend that you record any information that may prove useful later including, at least, the following:

- dates and descriptions of maintenance procedures
- printouts of error messages and diagnostic phases
- dates and descriptions of hardware changes
- dates and descriptions of software changes

Keeping Users Informed About Administrative Issues

Users need to know when a computer will be taken out of service, when a new software package will become available, and who to call for help when they encounter a problem with the system. To keep users informed about changes in hardware, software, administrative policies, and procedures, send them electronic messages with one of the communications tools provided with UNIX System V. Use the message of the day file (`/etc/motd`) for sending daily reminders and announcements. (For details, see the "User and Group Management" chapter.) For distributing information on an ad hoc basis, use the `news` facility (see `news`(1) for details).

Shutting Down the System

Many administrative tasks require the system to be shut down to a system state other than multi-user state (see the "System States" section in the "Machine Management" chapter). While the computer is not in multi-user state, your customers cannot access it. As a courtesy to these users, follow the suggestions below when you're planning administrative work that may disrupt their daily activities.

- Schedule tasks that will affect service for periods of low system use.
- Before taking any actions that might affect a user who is logged on, check to see who is on the system by running the `whodo` command (see `whodo`(1M)) or the `who` command (see `who`(1)).
- If the system is in use, give users advance warning about pending changes in system states or maintenance actions by running the `wall` command (see `wall`(1M)). Give users a reasonable amount of time to finish their activities and log off before taking the system down. If possible, tell users when they can expect the system to return to service.
- When unscheduled servicing occurs, run the `wall` command (see `wall`(1M)) to notify users. Again, if possible, tell them when they can expect the system to return to service.

2 Accounting

DRAFT COPY
January 31, 1992
File: Caccount

DRAFT COPY
January 31, 1992
File: Caccount

Introduction

The UNIX system accounting utilities are a family of mechanisms that collect data on system usage by CPU usage, by user, and by process. The utilities include tools for keeping track of connect sessions and disk usage. The accounting utilities can be used for

- charging for usage
- troubleshooting performance problems
- tuning the performance of applications
- managing installation security

To help you access the data captured by these utilities, the accounting utilities provide C language programs and shell scripts that organize the data into summary files and reports.

This chapter describes how the accounting utilities work. Specifically, it describes the numerous files and programs that figure prominently in the accounting system. The chapter also provides samples of the various reports generated by the accounting utilities.

Overview of Accounting

Once it has been set up, UNIX system accounting runs mostly on its own. (For instructions on setting up an accounting system, see "Setting Up Accounting" later in this chapter.) The following is an overview of how accounting works.

- Between system start-up and shutdown, raw data about system use (such as logins, processes run, and data storage) are collected in accounting files.
- Once a day, `cron` invokes the `runacct` program, which processes the various accounting files and produces both cumulative summary files and daily accounting reports. The daily reports are then printed by the `prdaily` program, which is invoked by `runacct`.
- The cumulative summary files generated by `runacct` can be processed and printed monthly by executing the `monacct` program. The summary reports produced by `monacct` provide an efficient means for billing users on a monthly or other fiscal basis.

Types of Accounting

The data collected daily with the procedure described above can help you do four types of accounting: connect accounting, process accounting, disk accounting, and fee calculations. The rest of this introduction defines these four types of accounting.

Connect Accounting

Connect accounting enables you to determine how long a user was logged in, and to obtain information about the usage of tty lines, the number of reboots on your system, and the frequency of the stopping and starting of the accounting software. To provide this information, the system stores records of time adjustments, boot times, the turning on or off of the accounting software, changes in run levels, the creation of user processes, login processes, and `init` processes, and the deaths of processes. These records (produced from the output of system programs such as `date`, `init`, `login`, `ttymon`, and `acctwtmp`) are stored in `/var/adm/wtmp`. Entries in `wtmp` may contain the following information: a user's login name, a device name, a process ID, the type of entry, and a time stamp denoting when the entry was made.

Process Accounting

Process accounting allows you to keep track of the following data about each process run on your system: the user and group IDs of those using the process, the beginning and elapsed times of the process, the CPU time for the process (divided between users and the system), the amount of memory used, the commands run, and the controlling tty during the process. Every time a process dies, the `exit` program collects these data and writes them to the file `/var/adm/pacct`.

The `pacct` file has a default maximum size of 500 blocks that is enforced by the accounting shell script `ckpacct` (normally run as a `cron` job). If `ckpacct` finds that `/var/adm/pacct` is over 500 blocks, it moves the file to `/var/adm/pacct?` where ? is the next unused increment (expressed as a number).

Disk Accounting

Disk accounting allows you to gather (and format) the following data about the files each user has on disks: the name and ID of the user, and the number of blocks used by the user's files. These data are collected by four programs in the accounting package: a shell script called `dodisk` and three C programs that it invokes (`diskusg`, `acctdusg`, and `acctdisk`). `diskusg` gathers the file data by reading file inodes directly from the file system and works only for s5 file systems. `acctdusg` does stat calls for each file in the file system tree to gather data and works for any file system type. `diskusg` is faster than `acctdusg`. `acctdisk` formats the data gathered by `diskusg` and/or `acctdusg` and saves the information in `/var/adm/acct/nite/disktacct`.

The `dodisk` script can be used in either of two ways: in fast mode or in slow mode. Fast mode uses `diskusg` for all s5 file systems and `acctdusg` for all others. The fast mode syntax is:

```
/usr/lib/acct/dodisk file_systems
```

File systems are specified by their special device names (such as `/dev/dsk/c8d0s2`). If the file systems are not specified, then the file systems used are those found in `/etc/vfstab` for which the value of `fsckpass` is `1`.

When run in slow mode, `dodisk` invokes `acctdusg` to gather all the disk accounting information, even from s5 file systems. The slow mode syntax is:

```
/usr/lib/acct/dodisk -o mountpoints
```

If no mountpoints are specified, the root mountpoint is used.

One note of caution: information gathered by running `dodisk` in either fast mode or slow mode is stored in the `/var/adm/acct/nite/disktacct` file. This information is overwritten the next time `dodisk` is used. Therefore, avoid using both modes on the same day. This allows `runacct` to use the information in the `/var/adm/acct/nite/disktacct` file before it is overwritten by new output from `dodisk`.

NOTE

`diskusg` may overcharge for files written in random access fashion, that have created holes in the file. This is because `diskusg` does not read the indirect blocks of a file when determining its size. Rather, `diskusg` determines the size of a file by looking at the *di_size* value of the inode.

Fee Calculations

If you charge your users for special services, such as file restores and remote printing, you may want to use a program called `chargefee` to maintain service accounts. Fees charged to customers are recorded in a file called `/var/adm/fee`. Each entry in the file consists of a user's login name, user ID, and the fee.

Accounting Programs

All the C language programs and shell scripts necessary to run the accounting system are in the `/usr/src/cmd/acct` directory. The `acctcom` program is stored in `/usr/bin`; all other binary programs are stored in `/usr/lib/acct`. These programs, which are owned by `bin` (except `accton`, which is owned by root), do various functions. For example, `/usr/lib/acct/startup` helps initiate the accounting process when the system enters multi-user mode. The `chargefee` program is used to charge a particular user for a special service, such as performing a file restore from tape. Other essential programs in the `/usr/lib/acct` directory include `monacct`, `prdaily`, and `runacct`. These and other programs are discussed in more detail in the following sections.

Setting Up Accounting

To set up system accounting so that it will be running while the system is in multi-user mode (system state 2), four files need to be created and/or modified. These files are `/sbin/rc0.d/K22acct`, `/var/spool/cron/crontabs/adm`, `/sbin/rc2.d/S22acct`, and `/var/spool/cron/crontabs/root`.

If you want accounting to be shut off during shutdown, link `/sbin/init.d/acct` to `etc/rc0.d/k22acct`.

If you want accounting to be turned on when the system is in multi-user mode (system state 2), link `/sbin/init.d/acct` to `/sbin/rc2.d/S22acct`.

Most of the `cron` entries needed for accounting are put into a database called `/var/spool/cron/crontabs/adm`. The entries in this database allow `ckpacct` to be run periodically, `runacct` to be run daily, and `monacct` to be run on a fiscal basis. Figure 2-1 shows several sample entries; your entries may vary. Be sure to append this information to the file to avoid destroying any entries already present. For the `adm` crontab, assign root as the owner, `sys` as the group, and `644` as the permissions mode.

Figure 2-1: Sample `cron` Entries for Accounting

```
------------------entries for adm crontab-----------------------------

#Min  Hour  Day      Month    Day   Command
#          of                 of
#          Month              Week
#-------------------------------------------------------------------------
0     *     *        *        *     /usr/lib/acct/ckpacct
30    2     *        *        *     /usr/lib/acct/runacct 2> /var/adm/acct/nite/fd2log
30    9     *        *        5     /usr/lib/acct/monacct
-------------------------------------------------------------------------
```

The entry for `dodisk` needs to be appended to the root crontab `/var/spool/cron/crontabs/root`. A sample is shown below.

```
--------------------entry for root crontab-------------------------------

#Min  Hour  Day      Month    Day   Command
#           of                 of
#           Month              Week
#-------------------------------------------------------------------------
30    22    *         *       4     /usr/lib/acct/dodisk
```

Once these entries are in the database and the accounting programs have been installed, accounting will pretty much run on its own.

Daily Accounting

Here is a step-by-step summary of how UNIX system accounting works:

1. When the UNIX system is switched into multi-user mode, the `/usr/lib/acct/startup` program is executed. The `startup` program executes several other programs that invoke accounting:

   - The `acctwtmp` program adds a "boot" record to `/var/adm/wtmp`. In this record the system name is shown as the login name in the `wtmp` record. Figure 2-2 presents a summary of how the raw accounting data is gathered and where it is stored.

Figure 2-2: Raw Accounting Data

File	Information	Written By	Format
`/var/adm/wtmp`	connect sessions	`login, init`	`utmp.h`
	date changes	`date`	
	reboots	`acctwtmp`	
	shutdowns	`shutacct` shell	
`/var/adm/pacct?`	processes	kernel (when process ends)	`acct.h`
		`turnacct` switch creates new file when old one reaches 500 blocks	
`/var/adm/fee`	special charges	`chargefee`	
`/var/adm/acct/nite/disktacct`	disk space used	`dodisk`	`tacct.h`

   - The `turnacct` program, invoked with the `on` option, begins process accounting. Specifically, `turnacct on` executes the `accton` program with the argument `/var/adm/pacct`.
   - The `remove` shell script "cleans up" the saved `pacct` and `wtmp` files left in the `sum` directory by `runacct`.

2. The `login` and `init` programs record connect sessions by writing records into `/var/adm/wtmp`. Any date changes (using `date` with an argument) are also written to `/var/adm/wtmp`. Reboots and shutdowns (via `acctwtmp`) are also recorded in `/var/adm/wtmp`.

   When a process ends, the kernel writes one record per process, in the form of `acct.h`, in the `/var/adm/pacct` file.

   Two programs track disk usage by login: `acctdusg` and `diskusg`. They are invoked by the shell script `dodisk`.

   Every hour `cron` executes the `ckpacct` program to check the size of `/var/adm/pacct`. If the file grows past 500 blocks (default), `turnacct switch` is executed. (The `turnacct switch` program moves the `pacct` file and creates a new one.) The advantage of having several smaller `pacct` files becomes apparent when trying to restart `runacct` if a failure occurs when processing these records.

   If the system is shut down using `shutdown`, the `shutacct` program is executed automatically. The `shutacct` program writes a reason record into `/var/adm/wtmp` and turns off process accounting.

   If you provide services on a request basis (such as file restores), you can keep billing records by login, by using the `chargefee` program. It allows you to add a record to `/var/adm/fee` each time a user incurs a charge. The next time `runacct` is executed, this new record is picked up and merged into the total accounting records.

3. `runacct` is executed via `cron` each night. It processes the accounting files `/var/adm/pacct?`, `/var/adm/wtmp`, `/var/adm/fee`, and `/var/adm/acct/nite/disktacct` to produce command summaries and usage summaries by login.

4. The `/usr/lib/acct/prdaily` program is executed on a daily basis by `runacct` to write the daily accounting information collected by `runacct` (in ASCII format) in `/var/adm/acct/sum/rprt`*MMDD*.

5. The `monacct` program should be executed on a monthly basis (or at intervals determined by you, such as the end of every fiscal period). The `monacct` program creates a report based on data stored in the `sum` directory that has been updated daily by `runacct`. After creating the report,

`monacct` "cleans up" the `sum` directory to prepare the directory's files for the new `runacct` data.

The runacct Program

The main daily accounting shell procedure, runacct, is normally invoked by cron during non-prime time hours. The runacct shell script processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by prdaily and monacct for billing purposes.

The runacct shell script takes care not to damage files if errors occur. A series of protection mechanisms are used that attempt to recognize an error, provide intelligent diagnostics, and end processing in such a way that runacct can be restarted with minimal intervention. It records its progress by writing descriptive messages into the file active. (Files used by runacct are assumed to be in the /var/adm/acct/nite directory unless otherwise noted.) All diagnostic output during the execution of runacct is written into fd2log.

If the files lock and lock1 exist when invoked, runacct will complain. These files are used to prevent simultaneous execution of runacct. The lastdate file contains the month and day runacct was last invoked and is used to prevent more than one execution per day. If runacct detects an error, a message is written to the console, mail is sent to root and adm, locks are removed, diagnostic files are saved, and execution is ended.

Reentrant States of the runacct Script

To allow runacct to be restartable, processing is broken down into separate reentrant states. A file is used to remember the last state completed. When each state completes, statefile is updated to reflect the next state. After processing for the state is complete, statefile is read and the next state is processed. When runacct reaches the CLEANUP state, it removes the locks and ends. States are executed as follows:

SETUP — The command turnacct switch is executed to create a new pacct file. The process accounting files in /var/adm/pacct? (except the pacct file) are moved to /var/adm/Spacct?.*MMDD*. The /var/adm/wtmp file is moved to /var/adm/acct/nite/wtmp.*MMDD* (with the current time record added on the end) and a new /var/adm/wtmp is created. closewtmp and utmp2wtmp add records to wtmp.*MMDD* and the new wtmp to account for users currently logged in.

WTMPFIX The wtmpfix program checks the wtmp.*MMDD* file in the nite directory for correctness. Because some date changes will cause acctcon to fail, wtmpfix attempts to adjust the time stamps in the wtmp file if a record of a date change appears. It also deletes any corrupted entries from the wtmp file. The fixed version of wtmp.*MMDD* is written to tmpwtmp.

CONNECT The acctcon program is used to record connect accounting records in the file ctacct.*MMDD*. These records are in tacct.h format. In addition, acctcon creates the lineuse and reboots files. The reboots file records all the boot records found in the wtmp file. CONNECT was previously two steps called CONNECT1 and CONNECT2.

PROCESS The acctprc program is used to convert the process accounting files /var/adm/Spacct?.*MMDD*, into total accounting records in ptacct?.*MMDD*. The Spacct and ptacct files are correlated by number so that if runacct fails, the unnecessary reprocessing of Spacct files will not occur. One precaution should be noted: when restarting runacct in this state, remove the last ptacct file because it will not be complete.

MERGE Merge the process accounting records with the connect accounting records to form daytacct.

FEES Merge in any ASCII tacct records from the file fee into daytacct.

DISK If the dodisk procedure has been run, producing the file disktacct, merge the file into daytacct and move disktacct to /tmp/disktacct.*MMDD*.

MERGETACCT
Merge daytacct with sum/tacct, the cumulative total accounting file. Each day, daytacct is saved in sum/tacct.*MMDD*, so that sum/tacct can be recreated if it is corrupted or lost.

CMS The program acctcms is run several times. It is first run to generate the command summary using the Spacct? files and writes it to sum/daycms. acctcms is then run to merge sum/daycms with the cumulative command summary file sum/cms. Finally, acctcms is run to produce the ASCII command summary files nite/daycms and nite/cms from the files sum/daycms and sum/cms, respectively. The program lastlogin is used to create /var/adm/acct/sum/loginlog, the report of when each user last

logged on. (If runacct is run after midnight, the dates showing the time last logged on by some users will be incorrect by one day.)

USEREXIT
: Any installation-dependent (local) accounting program can be included here. runacct expects it to be called /usr/lib/acct/runacct.local.

CLEANUP
: Clean up temporary files, run prdaily and save its output in sum/rprt*MMDD*, remove the locks, then exit.

runacct Error Messages

The runacct procedure can fail for a variety of reasons; the most frequent reasons are a system crash, /var running out of space, and a corrupted wtmp file. If the active*MMDD* file exists, check it first for error messages. If the active file and lock files exist, check fd2log for any mysterious messages. The following are error messages produced by runacct and the recommended recovery actions:

ERROR: locks found, run aborted
: The files lock and lock1 were found. These files must be removed before runacct can restart. Either two processes are trying to run runacct simultaneously or the last runacct aborted abnormally without cleaning up the locks. Check the fd2log for messages.

ERROR: acctg already run for date: check /var/adm/acct/nite/lastdate
: The date in lastdate and today's date are the same. Remove lastdate.

ERROR: turnacct switch returned rc=?
: Check the integrity of turnacct and accton. The accton program must be owned by root and have the setuid bit set.

ERROR: Spacct?.*MMDD* already exists
: File setups probably already run. Check status of files, then run setups manually, if necessary.

`ERROR: /var/adm/acct/nite/wtmp.MMDD already exists, run setup manually`

`/var/adm/wtmp` has already been copied to `/var/adm/acct/nite/wtmp.MMDD`

`ERROR: wtmpfix errors see /var/adm/acct/nite/wtmperror`

`wtmpfix` detected a corrupted `wtmp` file. Use `fwtmp` to correct the corrupted file.

`ERROR: Invalid state, check /var/adm/acct/nite/statefile`

The file `statefile` is probably corrupted. Check `statefile` and read `active` before restarting.

Files Produced by `runacct`

The following files produced by `runacct` (found in `/var/adm/acct`) are of particular interest:

`nite/lineuse`	`runacct` calls `acctcon` to gather data on terminal line usage from `/var/adm/acct/nite/tmpwtmp` and writes the data to `/var/adm/acct/nite/lineuse`. `prdaily` uses this data to report line usage. This report is especially useful for detecting bad lines. If the ratio between the number of logoffs to logins exceeds about 3:1, there is a good possibility that the line is failing.
`nite/daytacct`	This file is the total accounting file for the day in `tacct.h` format.
`sum/tacct`	This file is the accumulation of each day's `nite/daytacct` and can be used for billing purposes. It is restarted each month or fiscal period by the `monacct` procedure.
`sum/daycms`	`runacct` calls `acctcms` to process the data about the commands used during the day. This information is stored in `/var/adm/acct/sum/daycms`. It contains the daily command summary. The ASCII version of this file is `/var/adm/acct/nite/daycms`.

sum/cms	This file is the accumulation of each day's command summaries. It is restarted by the execution of monacct. The ASCII version is nite/cms.
sum/loginlog	runacct calls lastlogin to update the last date logged in for the logins in /var/adm/acct/sum/loginlog. lastlogin also removes from this file logins that are no longer valid.
sum/rprt*MMDD*	Each execution of runacct saves a copy of the daily report that was printed by prdaily.

Fixing Corrupted Files

Unfortunately, this accounting system is not entirely foolproof. Occasionally, a file will become corrupted or lost. Some of the files can simply be ignored or restored from the backup. However, certain files must be fixed to maintain the integrity of the accounting system.

Fixing `wtmp` Errors

The `wtmp` files seem to cause the most problems in the day-to-day operation of the accounting system. When the date is changed and the system is in multi-user mode, a set of date change records is written into `/var/adm/wtmp`. The `wtmpfix` program is designed to adjust the time stamps in the `wtmp` records when a date change is encountered. However, some combinations of date changes and reboots will slip through `wtmpfix` and cause `acctcon` to fail. The following steps show how to patch up a `wtmp` file.

Figure 2-3: Repairing a `wtmp` File

```
cd /var/adm/acct/nite
fwtmp < wtmp.MMDD > xwtmp
ed xwtmp
   delete corrupted records or
   delete all records from beginning
   up to the date change
w
q
fwtmp -ic < xwtmp > wtmp.MMDD
```

If the `wtmp` file is beyond repair, create a null `wtmp` file. This will prevent any charging of connect time. As a side effect, the lack of a `wtmp` file prevents `acctprc` from identifying the login that owned a particular process; the process is charged to the owner of the first login in the password file for the appropriate user ID.

Fixing `tacct` Errors

If the installation is using the accounting system to charge users for system resources, the integrity of `sum/tacct` is important. Occasionally, mysterious `tacct` records will appear with negative numbers, duplicate user IDs, or a user ID of `65,535`. First, check `sum/tacctprev`, using `prtacct` to print it. If it looks all right, the latest `sum/tacct.`*MMDD* should be patched up, then `sum/tacct` recreated. A simple patchup procedure would be:

Figure 2-4: Repairing a `tacct` File

```
cd /var/adm/acct/sum
acctmerg -v < tacct.MMDD > xtacct
ed xtacct
    remove the bad records
  write duplicate uid records to another file
w
q
acctmerg -i < xtacct > tacct.MMDD
acctmerg tacctprev < tacct.MMDD > tacct
```

The current `sum/tacct` can be recreated by merging all existing `tacct.`*MMDD* files by using `acctmerg`, since the `monacct` procedure removes all the old `tacct.`*MMDD* files.

Restarting `runacct`

Called without arguments, `runacct` assumes that this is the first invocation of the day. The argument *MMDD* is necessary if `runacct` is being restarted and specifies the month and day for which `runacct` will rerun the accounting. The entry point for processing is based on the contents of `statefile`. To override `statefile`, include the desired state on the command line. The following are some sample procedures.

To start `runacct`:

```
nohup runacct  2>  /var/adm/acct/nite/fd2log &
```

To restart `runacct`:

```
nohup runacct 0601  2>  /var/adm/acct/nite/fd2log &
```

To restart `runacct` in a specific state:

```
nohup runacct 0601 WTMPFIX  2>  /var/adm/acct/nite/fd2log &
```

Billing Users

The `chargefee` program stores charges for special services provided to a user, such as file restores, in the file `fee`. This file is incorporated by `runacct` every day.

To register special fees, enter the following command:

```
chargefee login_name amount
```

where *amount* is an integer amount to be charged. Most locations prefer to set up their own shell scripts for this function, with codes for services rendered. The operator then need identify only the service rendered; the system can tabulate the charge.

The monthly accounting program `monacct` produces monthly summary reports similar to those produced daily. (See Figure 2-9 later in this chapter for a sample report.) The `monacct` program also summarizes the accounting information into the files in the `/var/adm/acct/fiscal` directory. This information can be used to generate monthly billing. To generate a monthly billing, many UNIX system administrators customize the accounting process with their own shell scripts.

Setting Up Non-Prime Time Discounts

UNIX system accounting provides facilities to give users a discount for non-prime time system use. For this to work, you must inform the accounting system of the dates of holidays and the hours that are considered non-prime time, such as week-ends. To do this, you must edit the `/etc/acct/holidays` file that contains the prime/non-prime table for the accounting system. The format is composed of three types of entries:

- Comment Lines—Comment lines are marked by an asterisk in the first column of the line. Comment lines may appear anywhere in the file.
- Year Designation Line—This line should be the first data line (noncomment line) in the file and must appear only once. The line consists of three fields of four digits each (leading white space is ignored). For example, to specify the year as 1990, prime time start at 9:00 A.M., and non-prime time start at 4:30 P.M., the following entry would be appropriate:

```
1990  0900   1630
```

A special condition allowed for in the time field is that the time 2400 is automatically converted to 0000.

- Company Holidays Lines—These entries follow the year designation line and have the following general format:

  Date Description of Holiday

  The date field has the format *month/day* and indicates the date of the holiday. The holiday field is actually commentary and is not currently used by other programs. See Figure 2-5 for an example holiday list.

Figure 2-5: Holiday List

Month/Day	Holiday
1/1	New Year's Day
5/28	Memorial Day
7/4	Independence Day
9/3	Labor Day
11/22	Thanksgiving Day
11/23	Day after Thanksgiving
12/25	Christmas Day

Daily Accounting Reports

The `runacct` shell script generates four basic reports upon each invocation. They cover the areas of connect accounting, usage by login on a daily basis, command usage reported by daily and monthly totals, and a report of the last time users were logged in. The four basic reports generated are:

- The daily report shows line utilization by tty number.
- The daily usage report indicates usage of system resources by users (listed in order of UID).
- The daily command summary indicates usage of system resources by commands, listed in descending order of use of memory (in other words, the command that used the most memory is listed first). This same information is reported for the month with the monthly command summary.
- The last login shows the last time each user logged in (arranged in chronological order).

The following paragraphs describe the reports and the meaning of the data presented in each one.

Daily Report

This report gives information about each terminal line used. Figure 2-6 shows a sample daily report.

Figure 2-6: Sample Daily Report

```
Jun 29 09:53 1990  DAILY REPORT FOR sfxbs Page 1


from Thu Jun 28 17:45:22 1990
to   Fri Jun 29 09:51:25 1990
1        runacct
1        acctcon

TOTAL DURATION IS 966 MINUTES
LINE      MINUTES   PERCENT   # SESS    # ON      # OFF
term/23             25        3         7         4         4
term/22             157       16        6         3         3
TOTALS              183       --        13        7         7

--------------------------------------------------------------------------
```

The `from` and `to` lines tell you the time period reflected in the report: the period from the time the last accounting report was generated until the time the current accounting report was generated. It is followed by a log of system reboots, shutdowns, power fail recoveries, and any other record dumped into `/var/adm/wtmp` by the `acctwtmp` program; see `acct`(1M) in the *System Administrator's Reference Manual*.

The second part of the report is a breakdown of line utilization. The `TOTAL DURATION` tells how long the system was in multi-user state (accessible through the terminal lines). The columns are:

`LINE:` the terminal line or access port

`MINUTES:` the total number of minutes that line was in use during the accounting period

`PERCENT:` the total number of `MINUTES` the line was in use, divided into the `TOTAL DURATION`

`# SESS:` the number of times this port was accessed for a `login` session

`# ON:` This column does not have much meaning anymore. It used to list the number of times that a port was used to log a user on; but because `login` can no longer be executed explicitly to log in a new user, this column should be identical with SESS.

`# OFF:` This column reflects not just the number of times a user logs off but also any interrupts that occur on that line. Generally, interrupts occur on a port when `ttymon` is first invoked when the system is brought to multi-user state. Where this column does come into play is when the `# OFF` exceeds the `# ON` by a large factor. This usually indicates that the multiplexer, modem, or cable is going bad, or there is a bad connection somewhere. The most common cause of this is an unconnected cable dangling from the multiplexer.

During real time, you should monitor `/var/adm/wtmp` because it is the file from which the connect accounting is geared. If the `wtmp` file grows rapidly, execute `acctcon -l` *file* `< /var/adm/wtmp` to see which tty line is the noisiest. If the interrupting is occurring at a furious rate, general system performance will be affected.

Daily Usage Report

The daily usage report gives a breakdown of system resource utilization by user. Figure 2-7 shows a sample of this type of report.

Figure 2-7: Sample Daily Usage Report

```
Jun 29 09:53 1990  DAILY USAGE REPORT FOR sfxbs Page 1


     LOGIN  CPU (MINS)   KCORE-MINS    CONNECT (MINS)  DISK    # OF  # OF  # DISK   FEE
UID  NAME   PRIME NPRIME PRIME NPRIME  PRIME   NPRIME  BLOCKS  PROCS  SESS  SAMPLES
0    TOTAL  5     12     6     16      131     51      0       1114   13    0       0
0    root   2     8      1     11      0       0       0       519    0     0       0
3    sys    0     1      0     1       0       0       0       45     0     0       0
4    adm    0     2      0     1       0       0       0       213    0     0       0
5    uucp   0     0      0     0       0       0       0       53     0     0       0
999  rly    3     1      5     2       111     37      0       269    1     0       0
7987 jan    0     0      0     1       20      14      0       15     6     0       0
```

The data provided include the following:

`UID:` The user ID

`LOGIN NAME:` The login name of the user. This information is useful because it identifies a user who has multiple login names.

`CPU (MINS):` This represents the amount of time the user's process used the central processing unit. This category is broken down into `PRIME` and `NPRIME` (non-prime) utilization. The accounting system's idea of this breakdown is located in the `/etc/acct/holidays` file.

`KCORE-MINS:` This represents a cumulative measure of the amount of memory a process uses while running. The amount shown reflects kilobyte segments of memory used per minute. This measurement is also broken down into `PRIME` and `NPRIME` amounts.

`CONNECT` and `(MINS):`
This identifies the amount of "real time" used. What this column really identifies is the amount of time that a user was logged into the system. If the amount of time is high and the number shown in the column `# OF PROCS` is low, you can safely conclude that the owner of the login for which the report is being generated is a "line hog." That is, this person

logs in first thing in the morning and hardly touches the terminal the rest of the day. Watch out for this kind of user. This column is also subdivided into PRIME and NPRIME utilization.

DISK BLOCKS: When the disk accounting programs have been run, the output is merged into the total accounting record (daytacct) and shows up in this column. This disk accounting is accomplished by the program acctdusg. For accounting purposes, a "block" is 512 bytes.

OF PROCS: This column reflects the number of processes that were invoked by the user. This is a good column to watch for large numbers indicating that a user may have a shell procedure that has run out of control.

OF SESS: The number of times a user logged on to the system is shown in this column.

DISK SAMPLES: This indicates how many times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.

FEE: An often unused field in the total accounting record, the FEE field represents the total accumulation of widgets charged against the user by the chargefee shell procedure; see acctsh(1M). The chargefee procedure is used to levy charges against a user for special services performed such as file restores.

Daily Command Summary

The daily command summary report shows the system resource utilization by command. With this report, you can identify the most heavily used commands and, based on how those commands use system resources, gain insight on how best to tune the system. The daily command and monthly reports are virtually the same except that the daily command summary reports only on the current accounting period while the monthly total command summary tells the story for

the start of the fiscal period to the current date. In other words, the monthly report reflects the data accumulated since the last invocation of `monacct`.

These reports are sorted by `TOTAL KCOREMIN`, which is an arbitrary yardstick but often a good one for calculating "drain" on a system.

Figure 2-8 shows a sample daily command summary.

Figure 2-8: Sample Daily Command Summary

```
Jun 29 09:52 1990  DAILY COMMAND SUMMARY Page 1


                                        TOTAL COMMAND SUMMARY
COMMAND NUMBER    TOTAL  TOTAL    TOTAL   MEAN   MEAN     HOG    CHARS  BLOCKS
NAME     CMDS  KCOREMIN CPU-MIN REAL-MIN SIZE-K CPU-MIN FACTOR  TRNSFD    READ

TOTALS   1114      2.44   16.69   136.33   0.15    0.01   0.12 4541666    1926

sh        227      1.01    2.45    54.99   0.41    0.01   0.04  111025     173
fmli       10      0.50    2.06     9.98   0.24    0.21   0.21  182873     223
vi         12      0.35    0.62    44.23   0.55    0.05   0.01  151448      60
sed       143      0.09    0.82     1.48   0.10    0.01   0.55   14505      35
sadc       13      0.08    0.19     1.45   0.44    0.01   0.13  829088      19
more        3      0.04    0.07     2.17   0.59    0.02   0.03   30560       1
cut        14      0.03    0.09     0.28   0.37    0.01   0.33     154      13
uudemon.   76      0.03    0.66     2.30   0.05    0.01   0.29   43661      13
uuxqt      29      0.03    0.30     0.72   0.08    0.01   0.42   80765      35
mail        4      0.02    0.06     0.09   0.37    0.01   0.60    4540       9
ckstr      21      0.02    0.11     0.13   0.17    0.01   0.85       0       4
awk        13      0.02    0.12     0.21   0.15    0.01   0.54     444       2
ps          2      0.02    0.10     0.13   0.17    0.05   0.77    8060      21
find        9      0.02    3.35     5.73   0.00    0.37   0.58  355269     760
sar         1      0.01    0.19     0.24   0.08    0.19   0.80  564224       4
acctdisk    2      0.01    0.01     0.06   1.02    0.01   0.22       0       9
mv         24      0.01    0.14     0.17   0.10    0.01   0.81    3024      36
  .
  .
  .
```

The data provided include the following:

COMMAND NAME
: The name of the command. Unfortunately, all shell procedures are lumped together under the name `sh` because only object modules are reported by the process accounting system. It's a good idea to monitor the frequency of programs called `a.out` or `core` or any other name that does not seem quite right. Often people like to work on their favorite version of backgammon, but they do not want everyone to know about it. `acctcom` is also a good tool to use for determining who executed a suspiciously named command and also if super-user privileges were used.

PRIME NUMBER CMDS
: The total number of invocations of this particular command during prime time.

NON-PRIME NUMBER CMDS
: The total number of invocations of this particular command during non-prime time.

TOTAL KCOREMIN
: The total cumulative measurement of the amount of kilobyte segments of memory used by a process per minute of run time.

PRIME TOTAL CPU-MIN
: The total processing time this program has accumulated during prime time.

NON-PRIME TOTAL CPU-MIN
: The total processing time this program has accumulated during non-prime time.

PRIME TOTAL REAL-MIN
: Total real-time (wall-clock) minutes this program has accumulated.

NON-PRIME TOTAL REAL-MIN
: Total real-time (wall-clock) minutes this program has accumulated.

MEAN SIZE-K
: This is the mean of the `TOTAL KCOREMIN` over the number of invocations reflected by `NUMBER CMDS`.

MEAN CPU-MIN	This is the mean derived between the NUMBER CMDS and TOTAL CPU-MIN.
HOG FACTOR	The total CPU time divided by the elapsed time. This shows the ratio of system availability to system utilization. This gives a relative measure of the total available CPU time consumed by the process during its execution.
CHARS TRNSFD	This column, which may go negative because of overflow, is a total count of the number of characters pushed around by the `read` and `write` system calls.
BLOCKS READ	A total count of the physical block reads and writes that a process performed.

Total Command Summary

The monthly command summary is similar to the daily command summary. The only difference is that the monthly command summary shows totals accumulated since the last invocation of `monacct`. Figure 2-9 shows a sample report.

Figure 2-9: Sample Total Command Summary

```
                                TOTAL COMMAND SUMMARY

COMMAND NUMBER    TOTAL     TOTAL      TOTAL     MEAN    MEAN     HOG      CHARS    BLOCKS
NAME     CMDS  KCOREMIN CPU-MIN   REAL-MIN SIZE-K CPUMIN FACTOR     TRNSFD     READ

TOTALS 301314 300607.70 4301.59 703979.81   69.88  0.01   0.01 6967631360 10596385

troff     480  58171.37  616.15   1551.26   94.41  1.28   0.40  650669248   194926
rnews    5143  29845.12  312.20   1196.93   95.59  0.06   0.26 1722128384  2375741
uucico   2710  16625.01  212.95  52619.21   78.07  0.08   0.00  228750872   475343
nroff    1613  15463.20  206.54    986.06   74.87  0.13   0.21  377563304   277957
vi       3040  14641.63  157.77  14700.13   92.80  0.05   0.01  116621132   206025
expire     14  13424.81  104.90    265.67  127.98  7.49   0.39   76292096   145456
comp     3483  12140.64   60.22    423.54  201.62  0.02   0.14    9584838   372601
ad_d       71  10179.20   50.02   1158.31  203.52  0.70   0.04   11385054    19489
as       2312   9221.59   44.40    285.52  207.68  0.02   0.16   35988945   221113
gone      474   8723.46  219.93  12099.01   39.67  0.46   0.02   10657346    19397
i10       299   8372.60   44.45    454.21  188.34  0.15   0.10   60169932    78664
find      760   8310.97  196.91    728.39   42.21  0.26   0.27   58966910   710074
ld       2288   8232.84   61.19    425.57  134.55  0.03   0.14  228701168   279530
fgrep     832   7585.34   62.62    199.11  121.14  0.08   0.31   22119268    37196
sh      56314   7538.40  337.60 291655.70   22.33  0.01   0.00   93262128   612892
du        624   5049.58  126.32    217.59   39.97  0.20   0.58   16096269   215297
ls      12690   4765.60   75.71    541.53   62.95  0.01   0.14   65759473   207920
vnews      52   4235.71   28.11    959.74  150.70  0.54   0.03   28291679    28285
    •
    •
    •
```

Last Login Report

This report simply gives the date when a particular login was last used. You can use this information to find unused logins and login directories that may be archived and deleted. Figure 2-10 shows a sample report.

Figure 2-10: Sample Last Login

```
Feb 13 04:40 1990  LAST LOGIN Page 1

00-00-00  **RJE** 88-01-01  jlr     88-02-09  cec42   88-02-13  cec20
00-00-00  **rje** 88-01-13  crom    88-02-10  jgd     88-02-13  cec22
00-00-00  3bnet   88-01-14  usg     88-02-10  wbr     88-02-13  cec23
00-00-00  adm     88-01-17  cec11   88-02-11  cec30   88-02-13  cec24
00-00-00  daemon  88-01-17  cec38   88-02-11  cec41   88-02-13  cec25
00-00-00  notes   88-01-17  cec40   88-02-11  cec43   88-02-13  cec26
00-00-00  oas     88-01-18  cec60   88-02-11  cec53   88-02-13  cec27
00-00-00  pds     88-01-19  cec35   88-02-11  cec54   88-02-13  cec3
00-00-00  polaris 88-01-19  cec37   88-02-11  cec55   88-02-13  cec31
00-00-00  rje     88-01-22  dmk     88-02-11  cec56   88-02-13  cec32
00-00-00  shqer   88-01-26  ask     88-02-11  cec57   88-02-13  cec4
00-00-00  sys     88-01-26  cec39   88-02-11  cec58   88-02-13  cec6
00-00-00  trouble 88-01-27  sync    88-02-11  jwg     88-02-13  cec7
00-00-00  usors   88-02-02  pkl     88-02-11  skt     88-02-13  cec8
00-00-00  uucp    88-02-03  ibm     88-02-11  tfm     88-02-13  commlp
00-00-00  wna     88-02-03  slk     88-02-12  cec21   88-02-13  djs
87-07-06  lp      88-02-04  cec59   88-02-12  cec28   88-02-13  epic
87-07-30  dgn     88-02-05  cec33   88-02-12  cec29   88-02-13  jab
87-08-19  blg     88-02-05  cec34   88-02-12  csp     88-02-13  jcs
87-12-08  emna    88-02-05  cec36   88-02-12  drc     88-02-13  mak
88-01-14  s       88-02-05  cec51   88-02-12  emw     88-02-13  mdn
88-01-09  rib     88-02-05  dfh     88-02-12  je      88-02-13  mlp
88-01-25  dmf     88-02-05  fsh     88-02-12  kab     88-02-13  nbh
88-01-25  emda    88-02-05  pkw     88-02-12  rap     88-02-13  rah
          •
          •
          •
```

Looking at the `pacct` File with `acctcom`

At any given time, the contents of the `/var/adm/pacct?` files or any file with records in the `acct.h` format may be examined using the `acctcom` program. If you don't specify any files and don't provide any standard input when you run this command, `acctcom` reads the `pacct` file. Each record read by `acctcom` represents information about a dead process (active processes may be examined by running the `ps` command). The default output of `acctcom` provides the following information: the name of the command (prepended with a # sign if the command was executed with super-user privileges), the user, tty name (listed as ? if unknown), starting time, ending time, real time (in seconds), CPU (in seconds), and mean size (in K). The following information can be obtained by using options: F (the `fork/exec` flag: 1 for `fork` without `exec`), STAT (the system exit status), HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD, and BLOCKS READ.

The options are:

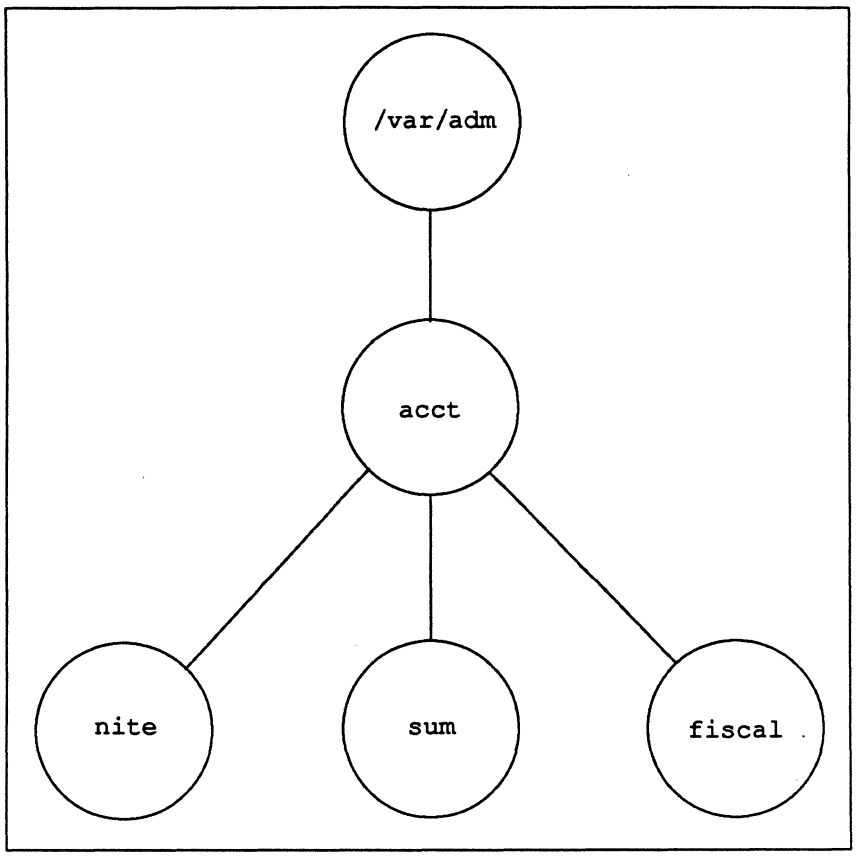
`-a`	Show some average statistics about the processes selected (printed after the output records).
`-b`	Read the file(s) backward, showing latest commands first. (Has no effect if reading standard input.)
`-f`	Print the `fork/exec` flag and system exit status columns.
`-h`	Instead of mean memory size, show the hog factor, which is the fraction of total available CPU time consumed by the process during its execution. Hog factor = *total_CPU_time/elapsed_time*.
`-i`	Print columns containing the I/O counts in the output.
`-k`	Show total kcore-minutes instead of memory size.
`-m`	Show mean core size (shown by default unless superseded by another option).
`-r`	Show CPU factor: *user_time/(system_time + user_time)*.
`-t`	Show separate system and user CPU times.
`-v`	Exclude column headings from the output.
`-l` *line*	Show only processes belonging to the terminal `/dev/`*line*

`-u` *user*	Show only processes belonging to *user*.
`-g` *group*	Show only processes belonging to *group*.
`-s` *time*	Show processes existing at or after *time*, given in the format *hr* `[ :`*min*`[ :`*sec*`] ]`.
`-e` *time*	Show processes existing at or before *time*, given in the format *hr* `[ :`*min*`[ :`*sec*`] ]`.
`-S` *time*	Show processes starting at or after *time*, given in the format *hr* `[ :`*min*`[ :`*sec*`] ]`.
`-E` *time*	Show processes starting at or before *time*, given in the format *hr* `[ :`*min*`[ :`*sec*`] ]`. Using the same *time* for both `-S` and `-E` shows processes that existed at the time.
`-n` *pattern*	Show only commands matching *pattern* (a regular expression as in `ed` except that "+"means one or more occurrences).
`-q`	Don't print output records, just print averages (akin to `-a`).
`-o` *ofile*	Instead of printing the records, copy them in `acct.h` format to *ofile*.
`-H` *factor*	Show only processes that exceed *factor*, where *factor* is the "hog factor" explained in the description of the `-h` option.
`-O` *sec*	Show only processes with CPU system time exceeding *sec* seconds.
`-C` *sec*	Show only processes with total CPU time, system plus user, exceeding *sec* seconds.
`-I` *chars*	Show only processes transferring more characters than the cut-off number specified by *chars*.

Accounting Files

The /var/adm directory structure (see Figure 2-11) contains the active data collection files and is owned by the adm login (currently user ID of 4).

Figure 2-11: Directory Structure of the adm Login



A brief description of the files found in the `/var/adm` directory follows:

`dtmp`	output from the `acctdusg` program
`fee`	output from the `chargefee` program, ASCII `tacct` records
`pacct`	active process accounting file
`pacct?`	process accounting files switched via `turnacct`
`Spacct?.`*MMDD*	process accounting files for *MMDD* during execution of `runacct`

The `/var/adm/acct` directory contains the `nite`, `sum`, and `fiscal` directories, which contain the actual data collection files. For example, the `nite` directory contains files that are reused daily by the `runacct` procedure. A brief summary of the files in the `/var/adm/acct/nite` directory follows:

`active`	used by `runacct` to record progress and print warning and error messages; `active`*MMDD* same as `active` after `runacct` detects an error
`cms`	ASCII total command summary used by `prdaily`
`ctacct.`*MMDD*	connect accounting records in `tacct.h` format
`ctmp`	Output of `acctcon1` program, connect session records in `ctmp.h` format. (`acctcon1` and `acctcon2` have been replaced in UNIX System V/68 or V/88 Release 4 by `acct-con`; they are provided here for compatibility purposes.)
`daycms`	ASCII daily command summary used by `prdaily`
`daytacct`	total accounting records for one day in `tacct.h` format
`disktacct`	disk accounting records in `tacct.h` format, created by the `dodisk` procedure
`fd2log`	diagnostic output during execution of `runacct`; see "Setting Up Accounting" at the beginning of this chapter
`lastdate`	last day `runacct` executed (in `date +%m%d` format)
`lock lock1`	used to control serial use of `runacct`

`lineuse`	tty line usage report used by `prdaily`
`log`	diagnostic output from `acctcon`
`log`*MMDD*	same as `log` after `runacct` detects an error
`owtmp`	previous day's `wtmp` file
`reboots`	contains beginning and ending dates from `wtmp` and a listing of reboots
`statefile`	used to record current state during execution of `runacct`
`tmpwtmp`	`wtmp` file corrected by `wtmpfix`
`wtmperror`	place for `wtmpfix` error messages
`wtmperror`*MMDD*	same as `wtmperror` after `runacct` detects an error
`wtmp.`*MMDD*	`runacct`'s copy of the `wtmp` file

The `sum` directory contains the cumulative summary files updated by `runacct` and used by `monacct`. A brief summary of the files in the `/var/adm/acct/sum` directory follows:

`cms`	total command summary file for current fiscal period in internal summary format
`cmsprev`	command summary file without latest update
`daycms`	command summary file for the day's usage in internal summary format
`loginlog`	record of last date each user logged on; created by `lastlogin` and used in the `prdaily` program
`rprt`*MMDD*	saved output of `prdaily` program
`tacct`	cumulative total accounting file for current fiscal period
`tacctprev`	same as `tacct` without latest update
`tacct.`*MMDD*	total accounting file for *MMDD*

The `fiscal` directory contains periodic summary files created by `monacct`. A brief description of the files in the `/var/adm/acct/fiscal` directory follows:

`cms?`	total command summary file for fiscal period ? in internal summary format
`fiscrpt?`	report similar to `rprt?` for fiscal period ?
`tacct?`	total accounting file for fiscal period ?

Quick Reference to Accounting

- Starting accounting:

  ```
  /usr/lib/acct/startup
  ```

- Turning off accounting:

  ```
  /usr/lib/acct/shutacct
  ```

- Switching the `pacct` file to the `pacct?` file:

  ```
  /usr/lib/acct/ckpacct
  ```

- Examining the contents of `pacct`:

  ```
  acctcom
  ```

- Charging a fee:

  ```
  /usr/lib/acct/chargefee login_name amount
  ```

- Processing accounting files into a daily summary:

  ```
  /usr/lib/acct/runacct 2 > /var/adm/acct/nite/fd2log
  ```

- Doing disk accounting:

  ```
  /usr/lib/acct/dodisk
  ```

- Creating a monthly accounting report:

  ```
  /usr/lib/acct/monacct
  ```

- Printing `tacct.h` files in ASCII format:

  ```
  /usr/lib/acct/prtacct
  ```

3 Crash Dump Subsystem

DRAFT COPY
February 1, 1992
File: Ccrashdump

Introduction

The purpose of this chapter is to explain how to configure and operate the crash dump subsystem. Included in this chapter are sample configurations of crash dump subsystems.

The crash dump subsystem records information about the system at the time of a system panic. The commands used to analyze this information can also be used to diagnose problems in a running system. A crash dump image can later be used to determine the cause of a panic.

NOTE

The crash dump subsystem is disabled by default. You must select the device to receive crash dumps prior to having a panic occur. Otherwise, little information about the cause of the panic will be available.

You can adjust the operation of the crash dump subsystem to satisfy the needs of your site. You can automatically record crash dumps and reboot the system if supported by the hardware, or you can choose to have the crash dump subsystem function interactively.

Assume that all disk and tape devices support the recording of crash dump information unless the device manpage states otherwise.

Managing the Crash Dump Subsystem

The crash dump subsystem operation depends on how you configure the `/etc/init.d/CRASHDUMP` script. This script uses the `/usr/sbin/ldsysdump` and `/usr/sbin/crashconf` programs to manage the crash dump subsystem. You can modify this script to enable or disable crash dumps, set the crash dump device, and set the mode of operation during a panic.

Initialization

When the system is booted, the crash dump subsystem is initialized by the `/etc/init.d/CRASHDUMP` script. The crash dump subsystem searches for the `.dumpsdown` file. This file is present only if the system was shutdown without experiencing a panic. If the `.dumpsdown` file is found, it is removed.

Panic

When the system panics and the crash dump subsystem is enabled, the system will react according to the configuration of the `/etc/init.d/CRASHDUMP` script.

The crash dump subsystem can be set to manual mode which prompts the operator to prepare the dump device to receive the dump image. If the dump device is a disk drive, the operator can issue a command to have the system continue. If the dump device is a tape drive, the operator must ensure that a non-write-protected tape is in the drive before issuing the command to have the system continue. The operator may stop the crash dump using the response options provided on the screen. If any errors occur, the crash dump system reissues the request to verify the device state and retries the operation.

If the crash dump subsystem is configured for automatic mode, the system attempts to take a crash dump.

Retrieval

If automatic retrieval is specified when the system is rebooted after a crash caused by a panic, a crash dump image is saved in the directory designated in the `CRASH-DUMP` script. The crash dump images are not automatically removed or replaced.

If the crash dump image is retrieved to the "/" or "`/usr`" file system, the image may fill the available space and the system will be unable to boot. You must maintain enough space on the file system where the directory exists to accommodate a file slightly larger than the amount of system memory.

If automatic retrieval is not specified, you must retrieve the crash dump manually using the `/usr/sbin/ldsysdump` program [see `ldsysdump`(1M)]. This program can also be used in the event automatic retrieval fails.

Analysis

Once the crash dump is retrieved, you can use the `/usr/sbin/crash` program to analyze the crash dump image [see `crash`(1M)]. Analyzing the crash dump image requires a copy of the `/stand/unix` that was running at the time of the panic.

Determining Where the System Will Save Crash Dumps

Determine if the system has a tape unit, a swap partition, or unused space on a disk that is larger than the amount of physical memory. Use the `/usr/sbin/fmthard` program to create a partition for crash dumps if necessary [see `fmthard`(1M)]. This partition may also be enabled for swapping via the `/usr/sbin/swap` program [see `swap`(1M)]. It is not necessary for the swap partition to be on the root device.

Configuring the Crash Dump Subsystem

After you have decided where you want to save crash dumps, you should edit the CRASHDUMP script to configure the crash dump subsystem. You must log in as root, then issue the following command to edit the CRASHDUMP script:

```
vi /etc/init.d/CRASHDUMP
```

The beginning of the script file provides definitions of the variables you will be setting. The following screen illustrates the default settings of the variables you will be editing.

```
# ************************ editable variables  ************************
DUMPSTART="yes"             # setup the crash dump system
#DUMPSTART="no"             # don't change crash dump setup

DUMPDEVICE=                 # device to dump to

#DUMPMODE="auto"   # without user intervention
#DUMPMODE="manual" # with user intervention
DUMPMODE="off"              # disable crash dumps

#DUMPRETRIEVAL="yes"        # retrieve a crash dump at boot time
DUMPRETRIEVAL="no" # don't retrieve crash dump at boot time

DUMPDIRECTORY=/             # directory of retrieved crash dumps
# ************************************************************************
```

CRASHDUMP Variables

This section provides explanations of the values for each variable in the CRASHDUMP script.

DUMPSTART — determines whether the crash dump subsystem will be configured

- yes — (default) enables crash dump subsystem configuration according to values in DUMPDEVICE and DUMPMODE

`no`	disables crash dump subsystem configuration
`DUMPDEVICE`	specifies device used for recording crash dumps
`pathname`	provides the device where crash dumps will be stored
`DUMPMODE`	determines the mode for the crash dump subsystem
`off`	(default) disables crash dump subsystem
`auto`	enables system to complete crash dumps without operator intervention
`manual`	requires operator intervention for completing a crash dump
`DUMPRETRIEVAL`	determines if crash dumps are retrieved during reboot
`no`	(default) prevents dump images from being retrieved at reboot (crash dumps must be retrieved manually)
`yes`	allows dump images to be retrieved during reboot and placed in `DUMPDIRECTORY`
`DUMPDIRECTORY`	provides the pathname to a directory where retrieved crash dumps are saved during reboot (the `.dumpsdown` file is placed here when the system is shutdown)

NOTE

The default `DUMPDIRECTORY` path is "/". However, if `DUMPRETRIEVAL` is set to `yes`, you should specify a different `DUMPDIRECTORY` path.

The `CRASHDUMP` script is invoked at startup and shutdown. After you have edited the variables in the script, issue the following commands to reconfigure the crash dump subsystem:

```
sh /etc/init.d/CRASHDUMP stop
sh /etc/init.d/CRASHDUMP start
```

Alternately, you can shutdown and reboot the system before the new configuration will take effect.

Sample Crash Dump Configurations

Crash dump information and images can be written to and stored in partitions on disk devices or on tape. This section describes several sample configurations based on the type of storage device chosen.

Dedicated partition

A partition is reserved only for crash dumps. The partition should be tagged SWAP with swapping disabled [see `prtvtoc` (1M) and `fmthard`(1M)]. A dump image is written to this partition during a panic and manually loaded with `ldsysdump` only when ready for analysis.

PROS

saves crash images with least amount of downtime

eliminates operator intervention

eliminates space problems

ensures crash dump is available if needed for analysis

CONS

retains only the latest dump image

reduces available disk space

Example:

```
DUMPSTART = yes
DUMPDEVICE = /dev/rdsk/device_name
DUMPMODE = auto
DUMPRETRIEVAL = no
DUMPDIRECTORY = /
```

Automatic swap partition

A swap partition, specified with the DUMPDEVICE variable, is used to store the crash dump image. The image is automatically retrieved to the DUMPDIRECTORY file system during reboot.

PROS

eliminates operator intervention

allows the retention of several dumps (dependent on available disk space)

faster than tape

CONS

fails to save crash image if file system is full

Example:

```
DUMPSTART = yes
DUMPDEVICE = /dev/rdsk/device_name
DUMPMODE = auto
DUMPRETRIEVAL = yes
DUMPDIRECTORY = pathname
```

Dedicated swap partition

A swap partition, specified with the DUMPDEVICE variable, is used to store the crash dump image. However, the dump is not retrieved automatically. Instead, the operator retrieves it manually immediately after reboot.

PROS

eliminates the need for dedicated disk space

requires operator intervention only if the crash dump image is wanted

faster than methods using automatic retrieval of crash dump images

CONS

allows crash dump images to be corrupted if not retrieved

Example:

```
DUMPSTART = yes
DUMPDEVICE = /dev/rdsk/device_name
DUMPMODE = auto
DUMPRETRIEVAL = no
DUMPDIRECTORY = /
```

Dedicated tape

The crash dump is sent to a tape drive used only for crash dumps. A tape should always be in the drive. The tape should be replaced if the crash dump image must be retained for later analysis.

PROS

eliminates operator intervention (except to replace tapes)

requires no use of disk space except during analysis

CONS

requires a dedicated tape drive

requires more time than writing to a disk device

Example:

```
DUMPSTART = yes
DUMPDEVICE = /dev/rmt/device_name
DUMPMODE = auto
DUMPRETRIEVAL = no
DUMPDIRECTORY = /
```

Manual tape

The system prompts the operator to prepare the tape drive for receiving a crash dump.

PROS

eliminates the need for dedicated disk space or devices

allows operator to abort crash dump process

CONS

requires operator intervention

requires more time than writing to a disk device

Example:

```
DUMPSTART = yes
DUMPDEVICE = /dev/rmt/device_name
DUMPMODE = manual
DUMPRETRIEVAL = no
DUMPDIRECTORY = /
```

4 Backup Service

DRAFT COPY
February 1, 1992
File: Cbackup

DRAFT COPY
February 1, 1992
File: Cbackup

Introduction

This chapter tells you how to perform backups so that you can always recover information that may be lost as a result of human or mechanical error. To help you plan, prepare for, and execute backups, the system provides a set of menus that guide you through the necessary steps in each process. To access the system administration menus for backups, type

```
sysadm backup_service
```

The following menu will appear on your screen:

```
        Backup to Removable Media

Backup History
Personal Backup
Schedule Backup to Tape
System Backup
```

If you prefer not to use this menu, you can do the same tasks by executing shell level commands, instead. The following table shows the shell commands that correspond to the tasks listed on the menu.

Task to Be Performed	`sysadm` Task	Shell Command
Request backup history information	`history`	
`Backup personal files`	`backup`	`cpio`(1)
Schedule automatic backup jobs	`schedule`	`crontab`(1)
Start backup jobs	`backup`	`cpio`(1)

Some of the menu items in the table above have their own underlying menus.

NOTE

Some of the `sysadm` forms that correspond to shell level commands may not offer the full features of that command, but will help administrators or operators with less experience by the use of prompts and help facilities.

The following table shows the menu items offered after the `schedule_task` menu item is chosen.

Task to Be Performed	`sysadm` Task	Shell Command
Add entries to the backup schedule	`add`	`crontab(1)`
Modify entries in the backup schedule	`change`	`crontab(1)`
Remove entries from the backup schedule	`delete`	`crontab(1)`
Display the backup schedule	`display`	`crontab(1)`

Each of these tasks is explained later in this chapter. In addition, the *System Administrator's Reference Manual* provides details about each shell command and its options.

An Overview of the Backup Service

The backup service allows you to make copies of both the data on your system and the partitioning information on your disk so you can restore, automatically, all disk formats or any data later lost from your system. Subsidiary facilities provided by the backup service include tools for creating online history reports of your backups and status reports on current backup jobs.

The first part of this section, "What Is a Backup?," defines the concepts and terminology used to describe the backup service. The second part, "Backup Methods and When to Use Them," explains the various procedures available for backing up your system and gives you some guidelines for selecting the procedure most appropriate for your needs.

What Is a Backup?

A backup operation is any procedure that allows you to copy either the data on your system or the partitioning information on your disk. The data to be copied can be in the form of a file system or a data partition. The data or partitioning information to be copied is referred to as an "originating object." Your copy (or "archive volume") of an originating object is stored on media such as cartridge tapes that are known as the "destination media." In the course of a typically large backup job, a computer operator must mount several such media on the "destination device."

Preparing for a Backup

The `backup` command requires values for a set of parameters that govern the results of a backup. These parameters are defined in a table referred to simply as the backup table. The sample backup table (`/etc/bkup/bkreg.tab`) will be useful in creating your own backup policy. For instructions on creating your own backup table, see "What Is a Backup Table?" later in this chapter.

Running a Backup

Once you have prepared the necessary backup table, you are ready to start a backup. You can run backups in either of two ways: attended or unattended. Unattended backups can be run by having `cron` run `backup` as a background command. The `backup` command, in turn, searches the backup table for all the

operations to be performed at that time, and starts doing them. If operator services are needed for media handling, the operator will be contacted by UNIX system mail and will be instructed to use `bkoper`.

On the other hand, attended backups require operator interaction. If operator services are needed for media handling, the operator will be notified by system prompts (rather than by UNIX system mail). This type of backup is initiated by typing `backup -i` (described below in "Interactive Mode" under "Selecting an Operator Mode") and specifying the backup table that contains the instructions for your operation. Once you have invoked `backup`, the command will read the instructions in the specified table, along with your additional instructions.

Keeping Track of Backup Jobs

A "backup job" is a set of one or more backup operations (each of which is labeled by an identifying tag), that is begun once the `backup` command is invoked. For example, one backup job might consist of three backup operations with the tags `acct3wkly`, `serviceswkly`, and `medrecswkly`.

To help you keep track of your backups, the system keeps records of both current jobs and completed operations. Current jobs are listed in a backup status table (described later in this chapter under "Monitoring Backup Jobs"); completed jobs, in a backup history log (described in "Displaying the Backup History Log").

When a backup job is completed successfully, the job ID for it is removed from the backup status table and the operation tags associated with it are placed in the backup history log. If a backup job is not successfully completed, the job ID for it is kept in the backup status table.

Backup Methods and When to Use Them

One of the most important parameters defined in the backup table is the backup method to be used. (See "Specifying Backup Methods" under "Preparing for Backup Operations" later in this chapter for instructions on specifying a method in your backup table.) There are six methods for backing up files, directories, file systems, and data partitions: incremental file, full file, full image, full disk, full data partition, and migration backups. You can limit your backups to one of these methods or you can use several methods, at different times, to achieve a comprehensive backup strategy for your system.

Each method is characterized by how it answers the following questions:

- What type of information is copied? (For example, does this method allow you to copy files, data partitions, or both?)
- How much information is copied? (For example, does this method allow you to copy only individual files or a complete file system?)
- How is information copied? (For example, is information copied as it appears logically on the originating device or is it copied in raw format, that is, byte by byte?)
- How long does a backup take when you use this method?
- What kind of procedure must be followed to restore information that has been backed up with this method? (See the "Restore Service" chapter for details.)

You should consider each of these questions when deciding which method to use. This section describes all six methods and provides guidelines for using them.

NOTE

If you are backing up the core file system, you must use either the full file backup method or the incremental file backup method; using any other method may corrupt your archive volume. For details about core file systems, see "Requesting Core File System Backups" later in this chapter.

When you have selected a backup method, request it in your backup table.

Full File Backups

This method allows you to copy all files and directories in a file system. This method is useful for backing up file systems that change frequently. It also provides more efficient restore capabilities than the full image backup and full data partition backup methods. By scheduling both full file and incremental file backups for your system, you can be sure that you have a comprehensive backup strategy.

Incremental File Backups

This method allows you to copy only those files and directories in a file system that have been modified or changed since a previous full file or full image backup. This method often requires less time and fewer destination archive volumes than other methods; the time required depends on the number of files that have been changed since the last full backup.

Full Image Backups

This method allows you to copy all the data in a file system. Unlike the full file method, the full image method allows you to copy a raw file system, byte-for-byte, onto the destination medium. This is the fastest method of doing a complete file system backup (because the files are not processed individually) but files and directories backed up in this way may take longer to restore than files and directories backed up with the full file or incremental file backup method. In addition, a spare disk partition equal in size to the origination partition is required to perform any restore operations.

Full Disk Backups

Doing backups with this method means you will be able to reinstall any required boot programs later. By using the full disk backup method, you can copy the entire contents of a disk so that if that disk is later lost, all the information required to rebuild it will be available.

The `fdisk` method backs up the disk VTOC (Volume Table of Contents) and file system characteristics, (as defined by `mkfs`(1M)). If the disk is destroyed, its contents can be recovered by first restoring the disk configuration using the `fdsk` method. Data partitions are restored using the appropriate backup method for each partition.

Full Data Partition Backups

This method allows you to copy a data partition that contains objects other than file systems, such as databases. This method is fast but it does not allow you to restore specific parts of a data partition. Because the structure of the data partition is not in the file system format, the restore service cannot identify individual files or directories; only a full data partition restore is possible.

Migrations

The migration method takes an existing archive created by another backup method and moves (migrates) it to a new location. This new location is reflected in the backup history log and any restore on the object migrated is performed using the original backup method that created the archive.

An archive created by any backup method can be migrated, as long as that archive was created on a disk partition or a file.

The migration method is useful when factors such as staffing, machine cycles, and the availability of storage devices vary over time. For example, you may want day-shift operators (who must avoid tying up too many system resources when many users are logged on) to perform quick disk-to-disk backups, and then schedule night-shift operators to migrate these backups to permanent destination devices during off-hours, when computer usage is low.

Suggestions for Performing Backup Operations

This section suggests a way to approach your backup duties. It describes the type of planning required, the steps involved in preparing for a backup, and the steps for backing up a system. Most of the effort required to use the backup and restore services is needed to prepare the backup procedures. Once this is done, performing backup and restore operations is simple.

The system administrator and computer center operator play different roles in the backup service. On small systems, one individual may perform both roles; on large systems, different people are usually assigned these areas of responsibility. The responsibilities of each are described below.

The Administrator's Tasks

The following is a detailed list of the administrator's responsibilities.

- Establish a backup plan (see "Establishing a System Backup Plan"). This plan should specify the following:
  - backup policies for a site based on factors such as resources, the needs of the users, and management directives
  - a list of file systems and data partitions that should be backed up, and for each file system or data partition, the intervals at which the backups should be done, the backup method to be used, and the types of destination devices to be used
  - how long and where destination media are to be kept before being reused
- Set up backup tables that provide the computer with instructions for implementing your plan. (See "What Is a Backup Table?" for instructions.)
- If your plan includes incremental file system backups and you want to exclude certain files from those jobs, you must create an exception list. (See "Excluding Files from Incremental Backups" under "Specifying Backup Methods," below, for instructions.)
- Either schedule backup jobs that will be invoked automatically, along with reminder messages (see "Establishing a System Backup Plan"), or invoke backup jobs manually (see the appropriate section of "Backup Methods and When to Use Them" earlier in this chapter for instructions).

- When performing backup operations, keep the following considerations in mind:
  - You must have enough destination media to complete the backup.
  - You must allow enough time to complete the backup.
- Check the status of backup jobs (see "Monitoring Backup Jobs").
- Evaluate your backup operations by examining the backup history log (see "Displaying the Backup History Log"). You may want to revise your plan on the basis of this evaluation.

The Operator's Tasks

The operator performs any attended or demand backup jobs scheduled by the administrator. During interactive backups, the operator must respond to system prompts, and mount and remove destination media.

Establishing a System Backup Plan

As an administrator, your first task regarding backups is to establish a system backup plan that specifies the following:

- Which objects need to be backed up?
- How often should the objects be backed up?
- Which is the most appropriate type of destination medium for storing the archive volume being made?
- How many destination media do you need for various backup methods (see "Previewing Backup Operations")? The number of cartridge tapes you need will depend on the size of your local system. As an example of the ratio of file system blocks to backup media, suppose you need to back up the `/usr` file system on your computer and it contains 13,294 blocks. You will have to allocate one cartridge tape for daily incremental backups.
- How much time do you need for various backup methods? As a rule of thumb, allow eight to ten times as much time in your plan for a full file backup as you do for an incremental file backup. To calculate the amount of time you will need, figure out how many destination media you will need for a particular operation (see previous item in this list), and then calculate the amount of time required when you know how long it will take to write to a certain type of medium.
- How many files are being copied during your incremental file backups? If the majority of files in a file system are being copied during incremental file backups, or if an incremental file backup takes almost as long as a full file backup, consider scheduling full file backups more frequently.
- How much time are you willing to devote to restoring a file system? If your plan relies heavily on incremental file system backups, you should keep in mind that all the destination media for each incremental backup may be needed to restore a file system to a consistent state.
- Which are the most appropriate methods for backing up an object?
- Should a backup be invoked automatically or manually?
- In what order should various backup operations be performed?
- Where and how long should backup archives be kept? You may want to save daily incremental backups for a week, weekly full backups for a month, and monthly full backups for a year or indefinitely. You may want to save some volumes off site to ensure that no one will accidentally overwrite an

important archive.

- Where should the historical records of completed backups be stored?

To answer these questions, begin by observing how the resources on your computer (such as disk space and CPU time) are used. Specifically, you need to know which file systems and data partitions are used and how they are used. Consider the approximate rate of change in the file systems studied. Notice whether changes occur throughout the file system or in only a small percentage of the files. If change is frequent and widespread, it may be best to schedule nightly incremental backups and weekly full backups. If change is concentrated in just a few files, a weekly incremental backup and a monthly full backup may be sufficient.

It is also a good idea to reevaluate, periodically, your system resources and how they are used. Keep in mind that if these periodic reevaluations show that the use of your computer is changing, you may revise your backup plan.

Preparing for Backup Operations

After you have established a plan, your next job is to prepare for backup operations by setting up your backup tables. In these tables you must define the parameters that control backup jobs, such as which file system is to be backed up and the day of the week on which the backup is to be done. This section explains how to define the necessary parameters in your backup tables, how to change those parameters, and how to validate the contents of your backup tables.

What Is a Backup Table?

Each backup table defines the following parameters:

- the backup operation tag
- the rotation period (the length of time after which a backup operation should be repeated)
- the days of the week on which the backup operation is to be done
- the backup method to be used
- the priority level of the backup operation
- the origination device from which the backup is to be made
- the destination media on which the backup is to be written

The system supplies a backup table with default values that can be changed. You can use this table or you can create your own backup table with its own default values that can be changed. Either type of backup table can be set up and maintained through the `bkreg` command.

To examine the current backup table for your system (whether it is a system supplied table or a custom table), issue the following command:

```
bkreg -C fields
```

where *fields* is a list of the fields for which you want to see the existing values. Separate the items in your list with either commas or blank spaces. (If you separate items with blank spaces, enclose the entire list in double quotes.) The

following are valid field names: `period`, `cweek`, `tag`, `oname`, `odevice`, `olabel`, `weeks`, `days`, `method`, `moptions`, `prio`, `depend`, `dgroup`, `ddevice`, `dchar`, and `dlabel`. If you type

```
bkreg -C tag,weeks,days,method,moptions,prio,dgroup
```

the following fields will appear on your screen (the values supplied are only examples):

```
# bkreg -C tag,weeks,days,method,moptions,prio,dgroup
Tag:Weeks:Days:Method:Options:Priority:Dgroup
::::::
::::::
::::::
::::::
rootsun:1:0:ffile:::ctape
rootdai:1:1-6:incfile:::ctape
usrdai:1:1-6:incfile:::ctape
usrsun:1:0:ffile:::ctape
```

Specifying Custom Backup Tables

The system-supplied backup table is named `/etc/bkup/bkreg.tab`. For a small system with limited backup operations, this table may be adequate. For large systems consisting of many computers, however, it may be more effective to create several backup tables of your own and sort the required backup operations into the different tables. For example, an administrator responsible for ten computers linked in a group called "services" and twelve computers linked in a group called "accounting" might set up two backup tables called `services` and `acctg`. These tables would be created by issuing the `bkreg` command followed by the `-t` option. The `-t` option would appear as follows:

```
bkreg -t /etc/bkup/services ...
bkreg -t /etc/bkup/accts ...
```

NOTE

The `-t` option to the `bkreg` command can be used in combination with any other options (such as `-a` or `-e`) whenever you want to work with a custom backup table.

You can create your custom backup table in any directory, but it might be convenient to put all backup tables in the `/etc/bkup` directory, where the system-supplied backup table resides.

Assigning or Changing Default Values in a Backup Table

Whether you are using an existing backup table (either system supplied or custom made) or creating a new one, you can assign values to the fields in that table. To assign values to a backup table, run the `bkreg` command, along with the appropriate options. (See `bkreg`(1M) in the *Systems Administrator's Reference Manual*.)

NOTE

If you are creating a new backup table you must identify the rotation period (by using the `-p` option) for that table before any other operations can be performed with that table. See "Specifying the Rotation Period" under "Preparing for Backup Operations" below.

Specifying Backup Methods

For each operation defined in the backup table, you must specify a backup method. This is done by issuing the `bkreg` command with the `-m` option. That option appears as follows:

`-m` *method*

where *method* is the backup method to be used. The following methods are available:

`incfile`	incremental file
`ffile`	full file system
`fimage`	full image

`fdisk`	full disk
`fdp`	full data partition

When you specify a method, you may also include any options applicable to it by using the `-b` option to `bkreg`.

The `-m` option can also be used to request a transfer of existing archives to a new location. This type of information transfer is called "migration." For more information about migrations, see "Requesting Migrations for Backed Up Information."

Method Options

When you specify a backup method (with the `-m` option to `bkreg`), you may also want to identify options that you want to have run with that method. If you then want to specify options to this method, introduce them on the command line with the `-b` flag (`-m` *method* `-b` *method options*). The following options are available:

`-d`	Suppresses the recording of the backup in the backup history log
`-e` *filename*	Specifies a custom exception list where *filename* is a full pathname (`incfile` and `ffile` only)
`-i`	Excludes from the backup those files in which only the `vnode` has been changed (`incfile` only)
`-l`	Creates a long form of the backup history log that includes a table of contents for the backup and information for each file similar to that produced by `ls -l`. (For details, see "Displaying the Backup History Log.")
`-m`	Mounts the originating device in read-only mode before starting the backup and remounts it with its original permissions after completing the backup. This option cannot be used with the core file systems (`fimage`, `ffile`, `incfile` only).
`-o`	Permits an operator to override label checking on destination devices (see the `-o` option to the `getvol`(1M) command and "Monitoring Backup Jobs")

`-s`	Specifies that no table of contents is to be kept on line
`-t`	Creates a table of contents for the backup on additional media instead of in the backup history log
`-v`	Validates the archive on the destination device as the backup operation is being performed to make sure that each block is readable and correct. If this check fails, the destination medium is considered unreadable. If automatic operator mode has been specified, the backup operation fails; otherwise, the operator is prompted to replace the destination medium.

Any of these options can be combined. Multiple options must be separated by blank spaces and must be enclosed in double quotes. The following is an example of how the `-m` and `-b` options appear:

```
-m ffile -b "-v -l"
```

Full File System Method

The full file system backup method copies all directories and files of a specified mounted file system to a destination device. The files are copied in the hierarchical order reflected by the directory structure.

To use this method, specify the `ffile` argument to the `-m` option on the `bkreg` command line (`-m ffile`).

Incremental File System Method

Incremental file backups copy only those files and directories that have been changed since one of the following:

- the last full file or full image backup
- the last full file or full image backup plus the last incremental file backup
- the last *n* days

Therefore, an incremental file backup must be preceded by at least one full backup that can be used as a base. Incremental file backups are useful for file systems that are changed frequently.

To use this method, specify the `incfile` argument to the `-m` option on the `bkreg` command line (`-m incfile`).

The following additional method options are available with the incremental file system backup method:

`-p` *mode* Shows which type of incremental file backup is to be performed, where *mode* is optional and can be any of the following:

0	the previous full file or full image
n(>0)	the last *n* days
No mode specified	The previous full file or full image

`-x` Ignores the exception list; backs up all changed or modified files. (For details, see "Excluding Files from Incremental Backups" below.)

Excluding Files from Incremental Backups

There are some files that are not appropriate to copy during an incremental backup. For example, it is not useful to copy temporary files, such as `/usr/tmp`, `/etc/utmp`, or `/tmp`, or files that can be reconstructed from other files, such as any `.o` files, `core` files, `a.out` files, or `nohup.out` files. In addition, files created by users, such as `dead.letter`, `junk`, `trash`, or `testing`, might not need to be backed up. Also, files that are routinely changed or created daily might not need to be backed up because they are invalid or outdated on subsequent days.

The incremental file backup method is designed to exclude these types of files automatically. Every time you run an incremental file backup, the backup command examines a list of files to be excluded. This list is known as the exception list. This list can be either the system-supplied exception list, `/etc/bkup/bkexcept.tab`, or your own exception list. (See "Creating an Exception List" below.)

You may want to review either list and make sure that it contains all the files you want to exclude from your backups and that it doesn't specify files that you want to have copied. If you want to modify the exception list, see "Modifying an Exception List" below for instructions.

If you do not want any files to be excluded from your incremental file backup (that is, if you want the exception list to be ignored), use the `-b -x` option to the `bkreg` command, when setting up your backup table.

Each entry in the exception list specifies, in the format of a "pattern," a set of one or more files. A pattern may consist of either the name of a single file or a string that includes one or more shell special characters (`\*, ?,` or `[ ]`), and thus represents multiple files. Special characters are similar to those used in shell commands.

For details about maintaining all exception lists, see `bkexcept`(1M) in the *System Administrator's Reference Manual*.

Creating an Exception List

You can create your own custom exception list through the `bkreg` command, as shown in the following example:

```
bkreg -e acct3 -m incfile -b "-e filename"
```

where *filename* is the full pathname of the exception list to be created.

Once this entry has been created, the backup service will use this exception list for the backup selected.

Modifying an Exception List

Use the same procedures to modify either a system-supplied exception list or one you have created. The `-t` option to the `bkexcept` command must be used each time an exception list other than the default list is referenced. If the `-t` option is not used, the default list is referenced.

To add entries to an exception list, type

```
bkexcept -a pattern
```

where *pattern* is a string that represents a file or a set of files, as defined on the `bkexcept`(1M) page in the *System Administrator's Reference Manual*. Items in a list of patterns must be separated by commas or by blank spaces (items separated by spaces must be enclosed in double quotes). For example, suppose you want to add entries for the following items (that is, you want to exclude these items from your incremental backups):

- all subdirectories and files under the `/tmp` directory
- all subdirectories and files under the `/usr/tmp` directory
- any file named `junk`
- the user file named `/usr/accts/clerk3/oldfile`

Type the following command line:

```
bkexcept -a \
/tmp/\*,/usr/tmp/\*,\*/junk,/usr/accts/clerk3/oldfile
```

NOTE

If a command does not fit on one line, escape the newline character between multiple lines by entering a backslash (\) at the end of every line except the last. If the command syntax requires a space between elements in the command, leave a space before each backslash.

As shown in this example, special characters (such as *) must be preceded by a shell escape character (such as the backslash shown above). The escape character prevents the shell from expanding the special character. If you prefer not to use escape characters in your command line, enclose the string of arguments to `-a` in quotation marks, as follows:

```
bkexcept -a \
"/tmp/\*,/usr/tmp/\*,\*/junk,/usr/accts/clerk3/oldfile"
```

Another way to avoid using escape characters on the command line is by entering a list of patterns from standard input. To do this, first enter a dash in place of the pattern argument on the command line. Then specify the desired patterns on separate lines. To end your list, type CTRL-d. The following example shows how to specify patterns in this way.

```
$ bkexcept -t /etc/save.d/except -a -
/tmp/*
/usr/tmp/*
/usr/spool/*
*/trash
/usr/accts/clerk3/oldfile
CTRL-d
$
```

To remove an entry from your exception list, type

```
bkexcept -r pattern...
```

where *pattern* matches an entry in the exception list. For example, to remove `/usr/spool/*` and `/usr/rje/*` from the exception list, type the following:

```
bkexcept -r /usr/spool/\*,/usr/rje/\*
```

You can remove entries by listing *patterns* on a command line, as shown above, or you can remove entries by specifying them on separate lines, by using a dash for the value of *pattern*. To end your list, type CTRL-d, as shown in the following example:

```
$ bkexcept -r -
/tmp/*
/usr/tmp/*
/usr/spool/*
*/trash
/usr/accts/clerk3/oldfile
CTRL-d
$
```

When you have added or removed entries in the exception list, you may want to see what the list contains. To display the full contents of the exception list (in ASCII collating order), invoke `bkexcept` with no options. To tailor the display, type

```
bkexcept -d patterns
```

The following example shows how a display appears on your screen. If you type

```
bkexcept -d /usr,/usr/spool
```

your output might appear as follows:

```
Files in the exception list beginning with /usr:
/usr/at
/usr/games
/usr/jpv/testfiles
/usr/spool/crontab

Files in the exception list beginning with /usr/spool:
/usr/spool/crontab
```

For exception list files containing only a (RETURN) character, `bkexcept -d /usr` returns successfully with a null exception list. For files of zero length (no characters), `bkexcept -d /usr` returns the message

```
search of table failed
```

Converting Exception Lists from Earlier Backup Services

Prior versions of the backup service created exception lists using `ed` syntax. The `bkexcept -C` command translates the entries in these earlier exception lists to the new shell pattern format. The translation is not perfect; not all `ed` patterns have equivalents in shell patterns. For those patterns that have no equivalents, an attempt at translation is made, and the translated version is flagged with the word `QUESTIONABLE`.

Because the translation is not perfect, you need to edit the output of the `bkexcept -C` command before adding it to the default exception list for the current backup service. The following procedure explains how to do this.

Step 1 The translation of the exception list is directed to standard output. Redirect the standard output to a file, such as `checkfile` in the following example:

```
bkexcept -C /etc/save.d/except > checkfile
```

The exception list from the prior version of the backup service specified in this example is `/etc/save.d/except`. Before being converted, the contents of this file appear as follows:

```
# Patterns of filenames to be excluded from saving by savefiles.
# These are ed(1) regular expressions.
/.news_time$
/.yesterday$
/a.out$
/core$
/dead.[a-l]*$
/ed.hup$
/nohup.out$
/tmp/
 .o$
^/etc/mnttab$
^/etc/save.d/timestamp/
^/etc/utmp$
^/etc/wtmp$
^/usr/adm/
^/usr/at/
^/usr/crash/
^/usr/dict/
^/usr/spool/
^/usr/tmp/
```

After this file has been converted by `bkexcept -C`, and you have redirected the output to `checkfile`, the contents of `checkfile` appear as follows:

```
QUESTIONABLE: #  Patterns of filenames to be excluded from saving by savefiles.
QUESTIONABLE: #  These are ed(1) regular expressions.
*/.news_time
*/.yesterday
*/a.out
core
*/dead.[a-l]*
*/ed.hup
*/nohup.out
*/tmp/*
*.o
/etc/mnttab
/etc/save.d/timestamp/
/etc/utmp
/etc/wtmp
/usr/adm/*
/usr/at/*
/usr/crash/*
/usr/dict/*
/usr/spool/*
/usr/tmp/*
```

Step 2 Review the contents of the new file (`checkfile` in this example). Notice that the `QUESTIONABLE` flag is used for two purposes: to mark comments in the old file, and to draw your attention to entries that may not have been translated properly. Delete the `QUESTIONABLE` flags that precede comments, preserving any comments that you want.

Review the entries that may not have been translated properly. Revise those entries that are not in the correct format by deleting the `QUESTIONABLE` flag and modifying the pattern as necessary. If you decide that a translation is adequate, you need only remove the `QUESTIONABLE` flag.

Step 3 After editing the entries in the converted file (`checkfile` in this example), add the contents of this file to the current exception list (`/etc/bkup/bkexcept.tab`). To do this, specify the converted file on the `bkexcept -a` command line, as shown in the following example:

```
bkexcept -a - < checkfile
```

Full Image Method

A full image backup copies an entire file system, byte-for-byte, starting with the first block and ending with the last. A full image backup differs from a full file backup because it does not copy the file system according to its directory structure. Instead, a full image backup copies the data blocks in the order in which they appear on the disk, from the first segment to the last segment.

To use this method, specify the `fimage` argument to the `-m` option on the `bkreg` command line (`-m fimage`).

Full Disk Method

The full disk backup method allows you to copy all the information required (by the restore service) to recover the format of an entire disk. Typically, the disk format is restored, followed by individual file systems and, finally, by the data partitioning information.

To use this method, specify the `fdisk` argument to the `-m` option on the `bkreg` command line (`-m fdisk`).

Full Data Partition Method

A full data partition backup allows you to copy a data partition that contains objects other than file systems, such as databases. Also, it allows you to copy a raw data partition, byte-for-byte, starting with the first block of the data partition and ending with the last.

To use this method, specify the `fdp` argument to the `-m` option on the `bkreg` command line (`-m fdp`).

Requesting Migrations for Backed Up Information

Migration is the process of moving an existing archive to a new location. The archive may have been created by any backup method and its new location is recorded in the backup history log. Any restore operation done with a migrated object must be done with the restore method appropriate for the backup method by which the archive was originally created.

A migration cannot be done until a backup operation has been performed. The fact that an operation has been performed implies that an entry for that operation exists in the backup table. That entry includes values for the originating device and the destination device.

Migrations are useful when factors such as staffing, machine cycles, and the availability of destination devices vary over time.

For example, you may want to schedule an automatic incremental file system backup to a spare disk partition at a specified time, and later have an operator move the resulting archive to tape. Specifically, suppose you want an incremental file backup for the operation known as `mktg3`, specifying `/usr:/dev/dsk/c8d0s2:usr` as the originating device and `:/dev/dsk/c8d1s2:` as the destination device. To request this operation, you must add an entry to the backup table. Issue the `bkreg` command with the options shown below:

```
bkreg -a mktg3 -m incfile \
-o /usr:/dev/dsk/c8d0s2:usr \
-d :/dev/dsk/c8d1s2:
```

Once this entry exists in the backup table, and `crontab` is updated to run `backup` at the required time, the information saved by the backup (the archive) will be stored in the designated destination device (`:/dev/dsk/c8d1s2:`).

To migrate the archive you must edit the entry for the relevant operation in the backup table, specifying a migration and the new destination for that archive. Therefore, you must edit the entry for the `mktg3` operation by issuing a command such as the following:

```
bkreg -e mktg3 -m migration \
-o :/dev/dsk/c8d1s2: \
-d :/dev/rmt/ctape1
```

As shown here, the destination device specified for the incremental backup is also specified as the originating device for the migration.

Once you have edited the specified entry in the backup table, an operator can request the migration with the `backup` command, as long as the migration requested is limited to the originating device you have specified in the table. In this example scenario, the operator now enters the following command:

```
backup -i -o :/dev/dsk/c8d1s2:
```

The results of a migration are as follows:

- An existing archive is stored on a new destination device (from which it can be restored in the same way any other archive is restored).
- The backup history log is updated to show the new location of information that has been migrated.
- The table of contents (for the archive that has been migrated) is updated to show the new location of the archive.

Requesting Core File System Backups

Core file systems are different from other file systems because they contain system software and, therefore, they must always remain mounted even when they are being backed up and restored. (For a description and list of core file systems, see the "File System Administration" chapter.)

A backup done on a core file system should be done on a "demand only" basis; specify `demand` in the `bkreg` table entry for each core file system backup.

Unlike other file systems, core file systems must be backed up only with the `-m ffile` option or the `-m incfile` option to the `bkreg` command. Do not use the full image backup method on a core file system because changes made to files in this system during this type of backup can corrupt the resulting archive. Running an incremental file backup or a full file backup is less dangerous to a core file system because changes made to a core file system during one of these types of backups will not corrupt your destination archive; at worst, you may get an intermediate file copy, or one that is not up to date.

Suppose you want to add an entry with the tag `usrdai` to the default backup table. The originating object to be backed up is the `/usr` file system on the `/dev/rdsk/c1d0s2` device (which is labeled `/usr`). You want to use the incremental file backup method (with the `-m incfile` option) on the next available cartridge tape. Type the following command line:

```
bkreg -a usrdai -o /usr:/dev/rdsk/c8d0s2:usr \
    -c demand -m incfile \
    -d ctape
```

See "Incremental File Backups" under "Backup Methods and When to Use Them" for information on the options listed in the above example.

Specifying Originating Objects

You can define the originating object for a backup operation by using the `-o` *orig* option to the `bkreg` command.

The *orig* argument takes the following form:

oname:odevice[*:omname*]

The following is an example of the `-o` *orig* option:

```
-o /usr:/usr/dev/dsk/c8d0s0:usr
```

Each component of the argument *orig* is defined below:

oname (`/usr`)	The pathname of the originating object. For file system partitions, this is usually the node name on which the file system is mounted. For data partitions, it is any valid pathname. This value is provided to the backup method and validated by the `backup` command. The data partition backup methods (invoked with the `-m fdp` and `-m fdisk` options to `bkreg`) do not require the name of the originating object; the `ffile` and `incfile` methods require *oname*.
odevice	The raw disk partition device name for the originating object.

omname — The volume label for the originating object. For file system partitions, it corresponds to the *volumename* specified with the `labelit` command. A data partition may have an associated volume name. If it does, the name is known only externally (taped on the device); run the `getvol`(1M) command to validate the name. Not all file system types support *omname*. See the "Storage Device Management" chapter for a list of those that do. (For details about volume names and the `labelit` command, see the "Storage Device Management" chapter.)

Specifying Destination Devices

All backup operations require a destination device on which an archive volume can be stored. To specify a destination device, use the `-d` option, as follows:

`-d` *ddev*

where *ddev* takes the form

dgroup:ddevice[*:dchar*][*:dmnames*]

Here both *dgroup* and *ddevice* must be specified and *dchar* and *dmnames* are optional. Colons separate fields and must be included as shown above. *dgroup* is the device group for the destination device. (For a description of device groups, see the "Storage Device Management" chapter.) If *dgroup* is not specified, *ddevice* must be specified and any available destination device in *ddevice* will be used.

dchar describes the characteristics of a destination device and, if specified, it overrides the default characteristics for the device and group. These characteristics are found in `/etc/device.tab`. The critical characteristics are type and capacity; these should be specified with *dchar*. (For details about the format and meaning of *dchar*, see the "Storage Device Management" chapter.)

NOTE: If device characteristics for a backup or restore device are not specified in `/etc/device.tab`, they must be specified in the backup register table.

dmnames is a list of names of the destination media. The items in this list must be separated either by commas or by blank spaces (items separated by blanks must be enclosed in double quotes). Each name in the list corresponds to a *volumename* specified with the `labelit` command. (For details about destination device labels and the `labelit` command, see the "Storage Device Management" chapter.) If *dmnames* is omitted, the `backup` and `restore` commands do not validate the labels on the destination devices.

Specifying the Rotation Period

A rotation period is the number of weeks between invocations of a backup operation. The rotation period for every backup operation is assumed to be one week (the default period) unless it is specified otherwise in the backup table. To set the rotation period for the system-supplied backup table, run

`-p` *rperiod*

where *rperiod* is an integer between 1 and 52 that represents the number of weeks between backups. (To set the rotation period in a custom backup table, use the `-t` option with the `-p` option.)

NOTE: The `-p` option to `bkreg` cannot be used with any other options on the same command line.

By default, a rotation period always begins on a Sunday, but you can change that parameter by entering

`-c` *weeks:days*

where *weeks* is a list of one or more integers between 1 and 52, that cannot exceed the value set by the `-p` *period* option. *days* is a list of either integers between 0 (Sunday) and 6 (Saturday) or the characters `s, m, t, w, th, f,` and `sa`. Both the *weeks* argument and the *days* argument can be specified as either a list of individual items (such as `1, 3, 5`) or as a range of items (such as 1-3). The

items in each list can be separated by either commas or blank spaces (in which case the list is enclosed in double quotes).

For example, if you want a backup operation to be performed every Sunday, Tuesday, Wednesday, Thursday, and Saturday on the first, second, third, and fifth weeks of the year, specify these times as shown below:

```
bkreg -a svcs -m incfile -c 1-3,5:s,t-th,sa
```

Another way of defining the rotation period for a backup operation is by requesting that this operation be performed only on a demand basis. Operations for which this request has been made are not performed regularly; they can be performed only when the `backup` command is invoked with `-c demand`. The following example command line shows how to request demand only status for all the backup operations listed in the `services` table.

```
bkreg -c demand -t /etc/bkup/services
```

Establishing Dependencies and Priorities

Some backup operations should be started before others begin. Some backup operations should not be run at all until other backups have been completed. The backup service lets you handle both these situations by providing a way for you to establish priorities and dependencies when setting up backup tables.

You may want to list your backup operations in the order you want them to run. To do this, assign a priority level to each backup operation defined in the backup table by using the `-P` option and specifying a priority level. The priority level is an integer from 0 to 100 where 0 is the lowest priority and 100 is the highest priority. If a set of backup operations are to be performed at the same time, each backup operation is not started until all others with a higher priority are completed. All backup operations with the same priority can be done simultaneously, unless the priority is 0. All backups with a priority of 0 are performed sequentially in an unspecified order.

You can also specify that a backup operation not be started until a set of other backup operations is completed successfully. These dependencies must be

identified in the backup table. To add a list of dependencies to the backup table, run the -D option, as follows:

```
-D depends
```

depends is a list of the operation tags for the operations on which a particular backup operation depends. Items in the list must be either separated with commas or separated with blank spaces (items separated by blanks must be enclosed in double quotes).

Establishing dependencies is particularly useful when using the migration method. A backup operation's dependencies take precedence over a backup operation's priorities. For example, an administrator may have a backup operation called `acctswkly` that is dependent on completion of the backup operation `SysengFri`. However, `SysengFri` has a priority of 40, which is less than the priority of `acctswkly1` (50). According to the rules of priorities, `SysengFri` should not begin before `acctswkly1`. But because dependencies take precedence over priorities, the `SysengFri` backup operation will be performed before the `acctswkly1` backup. To check the priorities and dependencies of these backup operations, type

```
bkreg -C tag,priority,depends
```

The following is an example of how the information would appear:

```
acctswkly1:50:SysengFri
SysengFri:40:
```

Creating Tables of Contents

Another item that you can specify in the backup table is a table of contents that lists the files and directories on a particular destination device.

NOTE

A table of contents is used by the restore service to locate files and directories to be restored.

Tables of contents can be provided only for backup operations performed with the incremental file, full file, or full image backup methods. Tables of contents are created and changed by using the `-s` and `-t` arguments to the `-m` *method* option. Because arguments to the `-m` *method* option must always be introduced on the command line by the `-b` flag, you would enter the `-s` and `-t` arguments as follows:

```
-m ffile -b "-s -t"
```

You can specify either a long-form or a short-form table of contents. The short form of a table of contents shows the names of the directories or files, and volumes that have been stored on a particular destination device. The long form of the table contains the same information as the short form, plus the information about those directories or files that is normally provided by the `ls -l` command.

By default, the system creates the short form of the table of contents. If you require the long form, specify the `-l` argument after the `-b` flag, as follows:

```
-m method -b -l
```

method may be `ffile`, `incfile`, or `fimage`.

You can store a table of contents in any of three ways:

- online
- on removable destination volumes
- both online and on destination volumes

Alternatively, you can specify that no table of contents be stored. The location of a table of contents is controlled by three factors: the system default, the `-s` argument to `bkreg -b`, and the `-t` argument to `bkreg -b`. These three factors can be used in any of the following combinations:

- To store a table of contents online only, use the system default; do not specify `-s` or `-t` after the `bkreg -b` flag.
- To store a table of contents on removable destination devices only, specify both `-s` and `-t` after the `-b` flag, as follows:

  ```
  -m method -b "-s -t"
  ```

- To store a table of contents both online and on removable destination devices, specify `-t` after the `-b` flag, as follows:

  `-m` *method* `-b -t`

- To request that no table of contents be stored, specify `-s` after the `-b` flag, as follows:

  `-m` *method* `-b -s`

Adding or Changing Backup Table Entries

There are three ways to change the contents of a backup table: you can add a new backup operation to the table, modify the instructions for a backup operation already defined in the table, or remove a backup operation from the table.

Adding an Operation Entry

To add a new backup operation to the table, type

`bkreg -a` *tag*

where *tag* identifies a backup operation. The `-a` option must be followed by the `-o, -c, -m,` and `-d` options and, if you choose, any of the following options that are not required: `-b, -t, -P,` and `-D`. The following table summarizes the options to the `-a` option.

Option and Argument	Argument Form	Meaning
`-a` *tag*	Alphanumeric string of any length	Adds a new entry to the backup table. The `-a` option must be followed by the `-o, -c, -m,` and `-d` options. You can also use the `-b, -t, -P,` and `-D` options; if you do not, the default values for those options are used.

The following example shows how to add an entry to a custom backup table.

```
bkreg -a acct5 -o /usr:/dev/rdsk/c8d0s2:usr \
-c 1,3-10,13:th -m incfile -b "-t -x" \
-d ctape1 \
-t /etc/bkup/wklybu.tab
```

This command line allows you to add an entry for a backup operation named `acct5` to a backup table named `wklybu.tab`. (If `wklybu.tab` does not already exist, it will be created.) The originating object to be backed up is the `/usr` file system on the `/dev/rdsk/c8d0s2` device. The backup operation defined here will be performed every two weeks on Sunday using the incremental file backup method.

The method options specify that a table of contents will be created on a destination device. The backup will be done to the next available destination device.

Modifying an Existing Operation Entry

To modify the contents of a backup operation already defined in a backup table, use the `-e` option, followed by a tag which identifies the existing backup operation you want to modify and any other options for which you want to change the value. The following example command line shows how to do this:

```
bkreg -e acct5 -t /etc/bkup/wklybu.tab \
-o /usr:/dev/rdsk/c8d0s2:usr -m incfile -b "-t -x" \
-d ctape1
```

Option and Argument	Argument Form	Meaning
`-e` *tag*	Alphanumeric string	Allows you to edit an existing table entry. If any of the options `-b`, `-c`, `-m`, `-o`, `-D`, or `-P` are present, the arguments to them replace the current values for the specified entries in the table.

Removing an Operation Entry

To remove the entry for an operation from a backup table, type

```
bkreg -r tag
```

where *tag* specifies the tag of the backup operation you want to remove.

Option and Argument	Argument Form	Meaning
`-r` *tag*	Alphanumeric string	Removes the specified entry

Validating Backup Tables

Before the operations listed in a backup table can be performed, the backup service checks for the consistency of several items (such as partition information) between the destination device and the backup table. Consistency is validated when the `backup` command is invoked. If `backup` is invoked and any of the consistency checks fail, `backup` terminates. If you prefer to validate the consistency of items in the system-supplied backup table manually, issue the following command:

```
backup -n -e
```

This command processes the backup operation without actually running it. By doing this step before running the backup, you can avoid a `backup` failure. (You can perform the same step for a custom backup table by entering the `-t` *table* option after the above command.)

If you want to check your backup tables before requesting a validation check, you can request a display of the contents of your tables. A display consists of a set of entries, each of which defines a backup operation. The following fields are available for display:

period	the number of weeks in the rotation period
cweek	the week in the rotation period to which the current week corresponds

tag	a unique identifier associated with the backup operation
oname	the pathname of the originating object
odevice	the device name of the originating device
olabel	the volume label of the originating device
weeks	the weeks, during the rotation period, in which the backup operation is performed
days	the days of the week on which the backup is performed
method	the backup method used
moptions	the options associated with a particular backup method
prio	the priority level for this backup operation
depend	tags for backup operations that must be completed successfully before this backup operation can begin
dgroup	the device group for a destination device
ddevice	the device name of a destination device
dchar	characteristics of a destination device
dlabel	the volume labels for a destination device

To display a complete table, run the `bkreg` command with the `-A` option. The `-A` option can be followed by the `-h, -s, -v, -t,` or `-c` options.

The output for this command is a set of extremely long lines; it is best used as input to a filter. To obtain a display that is easier to look at, run the following options to `bkreg`:

```
bkreg -C fields
```

Specify only those fields from the list above that you want to see. For example, you may want to display only the backup tags, the weeks of the rotation period, the backup methods used, dependencies, and priorities. If you enter

```
bkreg -C tag,weeks,method,depend,prio
```

the following information is displayed:

```
Tag:Weeks:Method:Depends:Pri
rootsun:1-8:ffile::
rootsp:r,8:ffile::20
usrdai:1-8:ffile::10
usrsun:1-8:ffile::
medrecs3:demand:ffile:mainofc:4
mainofc:demand:ffile::6
newusr2:demand:ffile::2
```

The `-C` option can be followed by one or more of the following options: `-h`, `-v`, `-t`, `-F`, or `-c`.

You may also tailor the information displayed by using either the `-O` or the `-R` option to the `bkreg` command. The `-O` option allows you to display a summary of all originating objects in the table. The `-R` option accesses a summary of all destination devices in the table. The `-O` and `-R` options, like the `-A` option, can be followed by one or more of the following options: `-h`, `-s`, `-v`, `-t`, and `-c`.

Performing Backup Operations

Once you have finished setting up your backup tables, you are almost ready to start running backup operations. Before you do, you will need to answer three questions:

- Which operator mode do you want to use?
- How much destination device space or how many destination archive volumes will you need?
- Do you want to limit your backup operation to a subset of the information normally copied during a defined rotation period (such as only today's files)?

This section defines each of these questions and explains how to find answers to them.

After you have selected an operator mode, have set aside the required number of destination volumes, and have decided which, if any, of the backup table values you want to override, you are ready to begin. You have already requested a backup method in your backup table. The rest of this section provides detailed descriptions of running backup operations using each backup method. It also explains how to back up a core file system (core file systems have special needs not associated with other file systems).

Selecting an Operator Mode

Once your backup tables are created, you must decide whether your backup operation requires operator assistance. Usually an operator is needed to mount destination devices. Some backup operations, however, are small enough that they can be done without any help from an operator. To accommodate both situations, the backup service allows you to run backup operations attended or unattended. For an attended backup, an administrator (or operator) issues the backup command and provides any necessary assistance during the course of the backup job.

Unattended backups are useful if your backup jobs do not require an operator to mount multiple destination devices and if you want your site's backup jobs to be done during off-hours when the computer center is unstaffed.

An unattended backup operation is run without the help of an operator; it is invoked by the system at a time you have specified in the backup table. Attended backups are useful when operators are present and when you want flexibility in the time of day that backup jobs occur.

There are three operator modes to accommodate situations in which:

- an operator is available but not at the terminal (background backups — the default mode)
- an operator is available at the terminal throughout a backup (interactive backups)
- no operator is available (automatic backups)

This section describes each mode and explains when you might want to use it.

Background Mode

If `backup` is invoked with no options, the background mode is used by default (this mode is normally set up to be run by `cron`). In the background mode, whenever a backup operation requires an operator's assistance, it sends a `mail` message to the operator's mailbox. The mail message reads as follows:

```
The following backup requires operator intervention:

job_id     tag       time      volume
back-441   acct3     09:35     /usr:/dev/rdsk/c8d0s2:usr
```

When a new medium is needed for a backup operation running in background mode, the backup operation is suspended until the operator receives the mail, issues the `bkoper` command, and responds to the prompts, thereby enabling the job to proceed. Note that in this mode, this type of suspension delays completion of any scheduled backup operations that depend on this backup or that have a lower priority.

Interactive Mode

If an operator will be present at the terminal during backup jobs, you may want to run your jobs in interactive mode. When you use this mode, the system sends all prompts directly to the standard output of the terminal where the `backup` command was issued. Interactive mode allows the operator on duty to respond to the prompts as they arrive at the operator terminal, to insert or remove destination media as required, and to oversee the progress of the backup operation. To request interactive mode, type

```
backup -i
```

If an operator will be present at the terminal during backup jobs, and wants to monitor an incremental file backup or a full file backup closely, the operator can request the backup service to post the progress of the operation. This request is made by running a backup job in verbose mode. This mode is a form of interactive mode, so to request it, use both the `-i` and the `-v` options, as follows:

```
backup -iv
```

When a backup is run with this command line, the name of every file and directory being backed up is displayed on standard output.

If you want to track the progress of a backup in more detail, request special verbose mode with the `-s` option:

```
backup -is
```

When a backup is run with this command, a dot is displayed as every 100 (512-byte) blocks are transferred to the destination device.

Automatic Mode

If no operator is available either to respond at the terminal or to receive mail messages, you can run your backup job in automatic mode. This mode allows you to schedule jobs in advance and to arrange for the system to start them without operator assistance at scheduled times. If any single backup operation requires operator assistance, that operation fails and the rest of the scheduled backup operations continue. To request automatic mode for your backup job, type

```
backup -a
```

Previewing Backup Operations

There may be times when you want to know the schedule of backup operations for a day without invoking any jobs. The backup service provides two preview capabilities that allow you to (*a*) preview the set of backup operations for a day, or (*b*) get an estimate of the number of destination media required for a backup.

To display the current day's backup operations in the order that they would proceed if invoked (that is, according to priorities and dependencies), type

```
backup -n
```

The following is an example of how information may appear:

```
  Tag     Orig.Name     Orig.Device     Dest.Group     Dest.Device     Pri   Depends On
 usrdai      /usr      /dev/dsk/c8d0s2      ctape      /dev/rmt/ctape1
usr2dai      /usr      /dev/dsk/c8d0s8      ctape      /dev/rmt/ctape1
rootdai       /        /dev/dsk/c8d0s0      ctape      /dev/rmt/ctape1
```

You can preview the same information for any day in a rotation period by adding the `-c` *week:day* option, as shown in the following example:

```
backup -n -c 3:f
```

This command line requests a list of backup operations scheduled for Friday (`f`) of the third (`3`) week in the rotation period.

After previewing the day's schedule of backups, you should find out whether you have enough space on your destination device or archive volumes before initiating a backup. You can find out how may devices you need by using the `-ne` option to `backup`.

The `-ne` and `-c` options can be combined to obtain a report that lists the scheduled backup operations for a specified day, along with an estimate of the number of devices required for each operation.

In addition to providing this information, the `-ne` option validates the consistency between corresponding items in the backup table and on the destination device. If any of the validation checks fail, the service sends an error message

to standard error. This capability enables you to correct problems before initiating an actual backup.

Requesting Limited Backups

When you issue the `backup` command, usually all operations defined in your backup table for the current rotation period are performed. There may be times, however, when you want to run a backup without having all operations performed. For example, you may want to run only backup operations defined for `demand` (that is, operations that are not scheduled to be run regularly). This section describes ways in which you can limit the backup operations to be performed.

- To run only those backup operations scheduled for the current day, use the `backup` command without the `-c` option. To invoke backup operations scheduled for a day other than the current day, type

  `backup -c` *week:day*

  specifying values (integers) for the desired week of the rotation period and values (either integers or letters) for the day. For example, if the current week is the eighth week of the rotation period, but you want to run the backup operations scheduled for Thursday of the seventh week, invoke:

  ```
  backup -c 7:th
  ```

- To run backups that are scheduled to run only on demand, invoke

  ```
  backup -c demand
  ```

- To run backup operations defined in a custom backup table, type

  `backup -t` *table*

- To back up objects on a particular originating device, type

  `backup -o` *orig*

  where *orig* must be of the form *oname*:*odevice*:[*omname*].

- To request that mail be sent to a specified user when the entire backup operation is complete, invoke

```
backup -m user
```

The above options can be used in various combinations on the `backup` command line. For example, suppose you want to invoke those backups listed on your custom backup table (`/etc/bkup/accts.tab`) that are scheduled for Friday of the second week in the rotation period. In addition, when the backup job has been completed, you want mail to be sent to the user with login `supv3`. Invoke this operation by typing

```
backup -t /etc/bkup/accts.tab -c 2:f -m supv3
```

Monitoring Backup Jobs

Backup jobs may require operator assistance for such tasks as mounting cartridge tapes and checking the labels on destination media to verify that the correct volumes are being used. An operator may perform a backup in any of three modes: background mode, interactive mode, or automatic mode. (See "Selecting an Operator Mode" for descriptions of the three modes.) This section explains how an operator interacts with a backup operation being run in background mode.

When a backup job in background mode cannot proceed further without the assistance of an operator, the `backup` command sends a `mail` message to the operator requesting assistance for that job. To find out what needs to be done, the operator must type `bkoper`. The `bkoper` command responds by printing a list of backup jobs for which assistance is needed, such as those shown in the following example:

```
1. back-111 usrsun /dev/dsk/c8d0s1  disk /dev/dsk/c2d1s9 usrsave
2. back-112 fs2daily /dev/dsk/c8d0s8 ctape /dev/rmt/ctape1
```

Each entry contains the following: operation number (the initial digit followed by a period), backup job ID (the `back-`*nnn* label), operation tag, originating device, destination device group, destination device name, and archive volume label. In the example above, the dash displayed as the last item of the second entry shows that no specific volume label is required for the backup operation listed. Backup operations are numbered in the order in which they appear in the backup table.

The `backup` command then displays the following question:

```
Which prompt do you want to respond to?
Type [q] to quit bkoper
Type [h] to display the list of backup operations
Type a number or RETURN to service a backup operation
?
```

To find out what kind of assistance is needed for the first operation listed, press RETURN or type `1`. The `bkoper` command will respond by explaining the task that needs to be done. After performing the task requested, press RETURN to find out what kind of assistance is needed for the next operation listed.

If you want to service an operation other than the current one (that is, other than the one at the top of the list), you can do so with either the `p` (print) keyletter or the `t` (type) keyletter, followed by the number of the relevant operation. For example, if you want to service the second operation before the current one, type

```
p2
```

These are only a few of the responses you may enter after the prompt, `Which prompt do you want to respond to?` For a complete list of possible responses to this question, see `bkoper`(1M) in the *System Administrator's Reference Manual*.

If, after the original list of operations has appeared, servicing is required for other operations (and you are still interacting through the `bkoper` command), the following message appears:

```
There are new backup operations requiring service.
```

When you have finished servicing all the operations that require assistance, the following message is displayed:

```
No more backup operations are waiting for operator action at
   this time.
```

If you want to interrupt your session with `bkoper` to perform a task at the shell level, type `!` and enter the desired command. To quit the `bkoper` session altogether, type `q`.

Checking Job Status

You can check the status of backup jobs (and the backup operations included in each job) by using the `bkstatus` command.

Each backup job progresses through the six states listed below.

pending	The `backup` command has been invoked and the operations listed in the backup table for the specified day are scheduled to occur.

active	All backup operations have been assigned destination devices and archiving is currently underway, or a suspended backup has been resumed.
waiting	The backup job is waiting for operator assistance, such as the mounting of a new tape.
suspended	The backup job has been suspended by an invocation of `backup -S`.
failed	The backup job has failed or has been canceled.
completed	The backup job has been completed successfully.

The status of backup operations is recorded in the `/etc/bkup/bkstatus.tab` table. In the table, each state is represented by the first letter of the relevant status: `p` (pending), `a` (active), `w` (waiting), `s` (suspended), `f` (failed), or `c` (completed).

You may display the backup status table in any of several ways. For all information about backup operations that are in progress (those labeled `a`, `p`, `w`, or `s`), invoke `bkstatus` with no options. To include information about backups with a status of `f` (failed) or `c` (complete), enter

```
bkstatus -a
```

A display such as the following will appear on your screen:

```
Jobid      Tag     User    Oname       Odevice          Start Time     Dest    Status
back-459   UsDly   oper1   /usr/spool  /dev/dsk/c8d0s8        -        ctape     f
back-459   PtsDly  oper1   VTOCc8d0    /dev/dsk/c8d0s7        -        ctape     c
back-395   SysDai  oper2   /sys        /dev/dsk/c8d0s9  May 26 16:45   ctape     c
```

If a full report is not required, you can limit the types of information that are displayed. For example, you can restrict a report to information about only specified jobs by invoking

```
bkstatus -j jobids
```

This command will not show those operations that were completed or failed. In addition, all *jobids* must be of the form `back-`*number*.

To restrict a report to information about jobs in particular states, invoke

```
bkstatus -s states
```

where *states* is a list of key-letters that are concatenated, comma-separated, or blank-separated (items separated by blanks are enclosed in double quotes), as shown in the examples below:

- `apf`
- `a,p,f`
- `"a p f"`

All three examples specify that the report should include only information about backup operations that are active or pending, or that have failed.

To restrict the report to information about backup operations invoked by specified users, type

```
bkstatus -u users
```

where *users* is a list of user logins that are separated by commas or blank spaces (items separated by blanks are enclosed in double quotes). The report will not include operations that were completed or failed.

By default, only one week's worth of backup status information is saved in this table. If you want backup status information to be saved for more than one week, type

```
bkstatus -p n
```

where *n* is the number of weeks for which you want information to be saved. You may find it useful to save status information for longer periods, such as a month, so you can examine patterns of servicing particular backup jobs.

Controlling Jobs in Progress

There may be times when you want to suspend a job, cancel a job, or resume a suspended job. You can request these actions by using the `backup` command options `-S`, `-C`, and `-R` (respectively), followed by the appropriate job ID. For example, suppose the backup job with the job ID `back-3288` is in progress when you need the destination device for another purpose. Suspend the job by invoking

```
backup -S -j back-3288
```

Entering `backup -S` without a job ID suspends all outstanding backup operations that were begun by the user entering this command.

Whenever the backup service receives a suspend request, it remembers the destination device currently in use, rewinds the destination device (if appropriate), and yields control of it. When the destination device becomes available for the backup again, you can resume the job by invoking

```
backup -R -j back-3288
```

Entering `backup -R` without a job ID resumes all outstanding backup operations that were begun by the user entering this command. The backup service revalidates the volume label and begins the backup from the beginning of the current volume, not from the point where the suspend request was received.

To cancel the backup job begun in the example above, type

```
backup -C -j back-3288
```

Entering `backup -C` without a job ID cancels all outstanding backup operations that were begun by the user entering this command.

In this case, the backup service rewinds the destination medium currently in use, and yields control of the destination device. In addition, the status display will reflect that this backup job has been canceled.

Displaying the Backup History Log

The `backup` command automatically records all backup operations that have been completed successfully in a backup history log (`/etc/bkup/bkhist.tab`). You can examine the contents of this log through the `bkhistory` command. When invoked without any options, `bkhistory` displays a summary of the contents of the backup history log that includes the following information:

`tag`	A unique identifier associated with a backup operation
`date`	The date and time when a backup operation was performed
`method`	The backup method used for the backup (`incfile, ffile, fimage, fdp, or fdisk`) and whether a migration was requested
`destination`	The name of the device that received the backup archive
`dmname`	The labels of the volumes that received the backup archive
`vols`	The number of volumes required to hold the entire backup archive
`TOC`	Whether a backup archive contains a table of contents and, if so, in what form. The four possible values of `TOC` are

`online`	The TOC is present online (on the hard disk).
`Arch`	The TOC is present in a volume on a destination device (a removable medium).
`Both`	The TOC is present both online and on a destination device (a removable medium).
`None`	No TOC exists.

Backups are listed alphabetically by operation tag. When a backup operation has been performed more than once during the period reported in the display, the most recent backup is listed first. The following is a sample display of information from a backup history log:

Figure 4-1: Sample Display of a Backup History Log

```
Tag      Date            Method   Destination  Dlabels  Vols  TOC
-------------------------------------------------------------------
rootdai  Feb24 12:26     incfile  /usr2/tmp/m  cpio1,   2     Online
         1989                     nt           cpio3
rootsun  Feb24 12:31     ffile    ctape        ctape1   4     Archive
         1989


usr2dai  Mar02 23:41     ffile    ctape        ctape1   3     None
         1989
usrdai   Jan28 20:29     incfile  file         med1,    3     Archive
         1990                                  med2,
                                               med3
```

Note that some entries in the `Dlabel` field may begin with the `!` character. This shows that the volume listed has been reused.

Customizing the History Log Display

The `bkhistory` command allows you to customize both the contents and the format of a display. This section explains how to do both.

Customizing the Contents of the Display

The default display contains full details about completed backup operations. You can restrict the type of information that is displayed by using the `-d`, `-t`, or `-o` options to the `bkhistory` command. To restrict a display to information about backups performed on specified dates, enter

```
bkhistory -d dates
```

where *dates* is a list of one or more dates separated by commas. The dates must conform to the syntax used with the `date`(1) command, with one exception: the only argument required is *month*.

To restrict a display to backup operations with specified tags, enter

```
bkhistory -t tags
```

To restrict a display to backup operations with specified originating devices, enter

```
bkhistory -o orig
```

where *orig* is in the following format: *oname*:*odevice*:[*omname*].

These options can be combined, as shown in the following example:

```
bkhistory -d 01,02,03 -o "/usr:/dev/rdsk/c8d0s2 \
/back:/dev/rdsk/c8d0s8"
```

This command line restricts the display to backup operations completed in January, February, and March from two originating devices: `/usr:/dev/rdsk/c8d0s2` and `/back:/dev/rdsk/c8d0s8`.

Customizing the Format of the Display

In the default display, each field is labeled by a header (such as `Tag`) and has a specified length. Entries that exceed the designated field length wrap to the next line within the field.

You can change the format of the display by using one or more of the following options: `-f`, `-h`, or `-l`. The `-h` option suppresses the headers in the display. This option is useful when the contents of the display are to be filtered by another process, such as an editor.

The `-f` option allows you to suppress field wrap and specify a character for separating fields in your display. This option cannot be used without the `-h` option. To invoke this option, type

```
bkhistory -h -fc
```

where the value of *c* is the character that will appear as the field separator. The fields in the display appear together in one line. For example, to display the output shown in Figure 3-1 in this format, type

```
bkhistory -h -f
```

The following display will appear:

```
rootdai;Feb 24 12:26 1990;incfile;/usr2/tmp/mnt;cpio1,cpio3,2;Online
rootsun;Feb 24 12:31 1990;ffile;ctape;;4;Archive
usr2dai;Mar 02 23:41 1990;ffile;ctape;;3;None
usrdai;Feb 24 12:26 1990;incfile;file;med4,med6;3;Both
usrdai;Jan 28 20:29 1990;incfile;file;med1,med2,med3;3;Archive
```

For clarity, when selecting a separator, do not choose a character that is likely to appear in a field. For example, do not use a colon as a field separator if the display will contain dates in which a colon is used to separate hours from minutes.

To produce the long form of the display, type

```
bkhistory -l
```

The long form includes the information shown in Figure 3-1, along with the information produced by the `ls -l` command. A display produced in this format looks like the following example:

```
Tag      Dlabel   File information
rootdai  cpio1    -r--r--r--    1 root  sys    2898   Mar 30 11:42  /etc/passwd
rootdai  cpio1    -rw-r--r--    1 root  sys    329    Jan 17 16:05  /etc/group
rootdai  cpio1    -rwxr--r--    1 root  other  646452 Mar 29 12:10  /unix
rootdai  cpio3    -rwxr-xr-x    1 bin   bin    33746  Mar 07 1987   /lib/cc
rootdai  cpio3    -rw-rw-r--    1 root  other  18616  Mar 29 12:10  /lib/libld.a
rootdai  cpio3    -rwsr-xr-x    1 root  sys    11242  Oct 09 17:30  /bin/newgrp
usrdai   med4     -rwxr-xr-x    1 root  other  851    Mar 29 12:10  /usr/oam/bin/add
usrdai   med4     -rwxr-xr-x    1 root  other  759    Mar 29 11:46  /usr/oam/bin/res
```

The entries in this display have values in the `File information` field because the `-l` option to the `bkreg` command was specified for the operations described.

Truncating the Backup History Log

Without some type of control, the backup history log would grow without bounds as more and more backups were completed. Therefore, by default, the system removes any entries that are older than one week. You can override this default and save history information for additional weeks by invoking

```
bkhistory -p period
```

where *period* is the number of weeks for which you want information to be saved in the backup history log. Keep in mind, however, the longer the history log, the greater the number of automatic restore operations that can be done.

Quick Reference to the Backup Service

- Adding an entry to a backup table:

  `bkreg -a` *tag*

  where *tag* identifies a backup operation. The `-a` option must be followed by the `-o`, `-c`, `-m`, and `-d` options and, if you choose, any of the following options that are not required: `-b`, `-t`, `-P`, and `-D`.

- Adding files to a custom exception list for incremental backups:

  `bkexcept -t` *filename* `-a` *pattern* ...

  where *filename* is the full pathname of the custom exception list, and *pattern* is a list of files and/or sets of files specified by the shell special characters *, ?, or [ ] that are comma-separated or blank-separated and enclosed in quotes.

- Adding files to the system-supplied exception list for incremental backups:

  `bkexcept -a` *pattern* ...

  where *pattern* is a list of files and/or sets of files specified by the shell special characters *, ?, or [ ] that are comma-separated or blank separated and enclosed in quotes.

- Checking the status of backup jobs:

  The status of a backup operation is shown by one of the following keyletters: `p` (pending), `a` (active), `w` (waiting), `s` (suspended), `f` (failed), `c` (completed)

- Displaying the status of backup operations that have either failed or been completed:

  `bkstatus -a`

- Interrupting a backup job:

  `backup -S|-C|-R [-j` *jobid*`] [-u` *users*`] [-A]`

  where `-S` suspends the backup job with the specified *jobid*, `-C` cancels the backup job with the specified *jobid* or cancels the backup job issued by *users* with the specified logins, `-R` resumes the backup job with the

specified *jobid*, and `-A` suspends, resumes, or cancels all running backup jobs.

- Defining the number of weeks of backup status information that will be saved:

  ```
  bkstatus -p n
  ```

  where *n* is the number of weeks for which information is to be saved.

- Defining and/or limiting the backup operations to be invoked:

  ```
  backup -t table -o orig -c week:day|demand -m user
  ```

  where *table* is the complete pathname of a custom backup table, and *orig* is the list of originating objects from which backups are to be made. *orig* must be of the form *oname*:*odevice*:[*omname*]. *week* is an integer specifying the week and *day* is an integer or character string specifying the day of the rotation period on which backups will be performed. *user* is the name of the user who is to be notified by `mail` when the backup job is complete.

- Displaying the contents of a backup table:

  ```
  bkreg -C fields|-A|-O|-R -t table
  ```

  where `-C` produces a summary display of the specified *fields*, `-A` displays all fields, `-O` displays a summary of all originating objects, and `-R` displays a summary of all destination devices. `-t` *table* is the name of the backup table.

- Editing an existing entry in a backup table:

  ```
  bkreg -e tag
  ```

  where *tag* identifies a backup operation. If any of the options `-b`, `-c`, `-d`, `-m`, `-o`, `-D`, or `-P` are present, they replace the current settings for the specified entry in the table.

- Displaying the contents of the backup history log:

  ```
  bkhistory
  ```

- Invoking backup operations that are run only on demand:

  ```
  backup -c demand
  ```

- Limiting the growth of the backup history log:

  `bkhistory -p` *period*

  where *period* is the number of weeks for which information will be saved.

- Previewing backup operations:

  `backup -n -e -c` *week:day* | `demand`

  where `-n` alone displays the current day's backup operations, `-e` estimates the number of destination device volumes required, and `-c` *week:day* | `demand` specifies the week and day of the rotation period (or the demand operations) to be previewed.

- Removing an entry from a backup table:

  `bkreg -r` *tag* `-t` *table*

- Removing files to a custom exception list for incremental backups:

  `bkexcept -t` *filename* `-r` *pattern* ...

  where *filename* is the full pathname of the custom exception list, and *pattern* is a list of files and/or sets of files specified by the shell special characters *, ?, or [ ] and are comma-separated or blank-separated and enclosed in quotes.

- Removing files from the system-supplied exception list for incremental backups:

  `bkexcept -r` *pattern* ...

  where *pattern* is a list of files and/or sets of files (specified by the shell special characters *, ?, and [ ]). *pattern* must match an entry in the exception list exactly.

- Requesting the long form of the backup history display:

  `bkhistory -l`

- Restricting the status information that is displayed:

  `bkstatus [-j` *jobids*`] [-s` *states*`] [-u` *users*`]`

  where *jobids* is a list of job IDs, *states* is a list of keyletters representing operation status, and *users* is a list of user login names.

- Selecting an operator mode while invoking the current day's backup operations:

  ```
  backup [-a|-i]
  ```

  where the default system response (in the absence of options) is to send a `mail` message to the operator when a backup operation needs assistance; `-i` prompts the operator at the terminal, and `-a` assumes no operator is present and fails any operations requiring assistance.

- Setting a rotation period for a backup table:

  `bkreg -p` *period* `-w` *cweek* `-t` *table*

  where *period* is the number of weeks in the rotation period, *cweek* is the current week of the rotation period, and *table* is the name of a backup table if a custom backup table is to be used

- Servicing backup operations:

  ```
  bkoper
  ```

  initiates an interactive session by displaying a list of backup operations

- Storing a table of contents online only:

  Tables of contents are stored online by default.

- Storing a table of contents on removable destination devices only, by specifying both `-t` and `-s`:

  `bkreg -m` *method* `-b "-t -s"`

- Storing a table of contents both online and on removable destination devices by specifying `-t`:

  `bkreg -m` *method* `-b -t`

- Storing a table of contents neither online nor on removable destination devices by specifying `-s`:

  `bkreg -m` *method* `-b -s`

- Tailoring the display of the contents of the backup history log:

  `bkhistory -d` *dates* `-o` *orig* `-t` *tags*

  where *dates* is a list of dates that restricts the report to backup operations

performed on the specified dates, *orig* restricts the report to the specified originating devices, and *tags* is a list of operation tags.

- Translating an exception list from `ed` syntax to `cpio` format:

  `bkexcept -C` *old_file* `>` *new_file*

  where *old_file* is the filename of the exception list in `ed` command syntax, and *new_file* is a temporary file that you edit before giving it to `/etc/bkup/bkexcept.tab` for input. After editing the file you can enter `bkexcept -a - <` *new_file*.

- Validating the contents of a custom backup table:

  `backup -n -t` *table*

  where *table* is the name of a custom backup table.

5 Diagnostics

DRAFT COPY
February 1, 1992
File: Cdiag

DRAFT COPY
February 1, 1992
File: Cdiag

Introduction

This chapter tells you how to perform diagnostic tests (to identify problems on your system) and how to handle bad blocks on your hard disk.

Two sets of diagnostic tests are provided with your system: one set to check your hard disk, and another to check all other hardware. The diagnostics used to locate, report, and repair hard disk errors reside on the hard disk. Diagnostics for identifying other hardware problems reside in the firmware, and can only be run while the system is in firmware state (system state 5). See the "Machine Management" chapter for a description of firmware state and other system states.

To find out which diagnostic tests to perform to identify a particular problem, contact your service representative.

Bad block handling and associated recovery procedures are also described in this chapter. The bad block handling feature is controlled by an automated process called the `hdelogger` daemon; the system administrator does not need to invoke it.

To help you perform diagnostic activities, the UNIX system provides a menu interface. To access a menu of diagnostic tasks, type

```
sysadm diagnostics
```

The following menu will appear on your screen:

```
1       Diagnosing System Errors

diskrepair - Advises about Disk Error Repairs
diskreport - Reports on Disk Errors
```

If you prefer not to use the menu, you can perform these tasks by using shell-level commands instead. The following table shows the shell commands and firmware programs that correspond to the tasks on the menu.

Task to Be Performed	sysadm Task	Shell Command or Firmware Program
Accessing the diagnostic monitor (firmware program)	n/a	shutdown -y -i5 BUG
Adding hand-written reports to the disk error queue	n/a	hdeadd
Advice on disk error repairs	diskrepair	n/a
Generating a hard disk error report	diskreport	hdelogger -f
Repairing a bad block that caused a system panic	n/a	shutdown, umountall -k, hdeadd, fsck -fD, mount

Overview of Diagnostics

Diagnostics programs are sets of tests, or "phases," that help you evaluate and sometimes repair problems. There are three categories of diagnostic phases: "default" (those that are run automatically), "demand" (those that are run only when you request them), and "interactive" (those that require your participation). In addition, the diagnostics software includes several tools for repairing some problems once they have been identified. You should run diagnostics at regular servicing intervals and whenever your computer sends a failure message to your console terminal. To find out which diagnostic phases you should run for a particular problem, contact your service representative.

Disk Diagnostics

To ensure that disk errors are recorded whenever they occur, a daemon process called `hdelogger`, which monitors disk status, runs constantly on your system. Disk errors cannot be repaired, but the UNIX system provides software tools for making usable the block (section) of the disk in which errors are found. These tools, collectively known as the "bad block handling feature," are described in the "Bad Block Handling" section of this chapter.

Recovering from System Trouble

This section provides instructions for handling hardware problems and software errors and describes how to return the system to a usable state.

A system experiencing trouble may be in any of the following conditions:

- The system is running, but throughput is degraded.
- The system is running after an automatic reboot.
- The system is halted or in an unknown state.

NOTE

You must log in as `root` at the console terminal before proceeding.

Identifying System Trouble

Consult the information about troubleshooting in your computer installation manual for problems that occurred during the first-time setup of your computer. If you need additional information, call your service representative.

Bad Block Handling

A disk is a magnetic medium on which digital data (measured in bits and bytes) are stored in logical sections called "blocks." A block usually contains 512, 1024, or 2048 bytes of data and is handled by the UNIX system as a single unit. Because the bit density is very high (millions of bits of data are packed into a small space), the magnetic properties of a disk must be precisely uniform. Variations in uniformity mean that some bit patterns may be stored more easily in one block than in another. This variation in uniformity is normally insignificant. However, when these variations become so great that data can no longer be stored reliably in a particular block, that block is labeled "bad." There are three categories of bad blocks, described under "Bad Block Recovery" later in this chapter.

For SCSI drives, if the pattern of data stored in a block coincidentally matches the nonuniformity of the disk, a bad block may escape recognition. However, the nonuniform block will eventually be discovered if the disk remains active. Bad blocks are discovered when a data read or write fails after several successive attempts. Read failures alone do not guarantee the existence of a bad block because they may also be caused by problems in the format of the disk, a failure in the disk controller, or hardware flaws. Write failures generally signal problems with the disk's format or failures in the disk or disk controller hardware.

The bad block handling feature does not distinguish genuine bad blocks from other types of problems; it reports all types of read and write failures, most of which are not caused by bad blocks. (Reports can be viewed by issuing the command `hdelogger -f`.) The distinction is unimportant, however, because all read and write failures indicate problems that must be repaired.

To fix read and write problems, you must reformat the disk or arrange to repair the hardware. In either case, you should call your service representative, especially if several failures occur about the same time.

How the UNIX System Handles Bad Blocks

You cannot really "fix" a bad block; you can only find a way to work around it. One way you can work around a bad block is by using a "media-specific data area."

A media-specific data area is a small portion of a disk that is isolated from the default data area; that is, it is not used by normal UNIX system commands and system calls. This isolated portion of the disk contains a description of the properties of the disk and other media-specific data.

The media-specific data area includes a set of blocks called the surrogate image region. The purpose of this region is to hold the data that should have been stored in a bad block. The media-specific data area also includes a mapping table to map a bad block on the disk to one of the surrogate image blocks. The disk driver software in the operating system has a map showing the original location of the data in the bad block and the new location of that data in the surrogate image block. Data are read from or written to the surrogate image block via the addresses in this mapping table. Address mapping is transparent to software programs accessing the data.

The UNIX system has a software feature called bad block handling. This feature extends the useful life of an integral hard disk by

- detecting and recording blocks that are no longer usable
- reminding you that you need to "fix" some bad blocks that have been identified
- restoring the usability of the disk in spite of the bad blocks that exist

Most disk drives are manufactured with a few defective blocks. Bad blocks detected in the manufacturer's quality control checks are identified on a label on the drive when it is delivered. The bad block handling feature provides special software for recording the known bad blocks and for mapping any additional ones that are encountered. If a surrogate block becomes bad, the software remaps the original bad block to a new surrogate block.

NOTE

This feature can be used only with hard disk devices. There is no comparable feature for tapes.

New bad blocks will seldom occur as long as you do not move or vibrate the disk drive while the disk is spinning. When a new bad block does occur, the data stored in the bad block are lost and the disk may be unusable in its current state. The bad block handling feature helps restore the usability of a disk. How much data you lose depends on the adequacy of the backup procedures you have established. (For a discussion of backup procedures, see the "Backup Service" chapter.)

Blocks that Cannot Be Mapped

There are a few special blocks in the isolated portion of the disk that cannot be mapped: the disk block containing the physical description of the disk and the disk block or blocks containing the mapping table. All other blocks, including the surrogate image blocks, can be mapped.

When Are Bad Blocks Detected?

Bad blocks are detected when read or write operations fail for several successive attempts. When these failures occur, data being read or written at the time are lost, but the system can restore the use of the disk by mapping the bad blocks to usable surrogate blocks.

There are several questions that are frequently asked about bad block handling.

- Why does the system not try to discover that a given block is bad while the system still has the data in memory?

  Besides the undesirable increase in system size and complexity, severe performance degradation would result. Also, a block can become defective after the copy in memory no longer exists.

- Why does the system not periodically test the disk for bad blocks?

  Because to do a valid surface analysis of the disk, all the data currently on the disk would be destroyed. The disk manufacturer has tested the disk using extensive bit pattern tests and special hardware. After these initial tests, it is unlikely you would find additional bad blocks.

- Why are disks with manufacturing defects used?

  By selling disks that contain a modest number of defects, manufacturers can greatly increase their yield and thereby reduce the cost of each disk. Many manufacturers take advantage of this cost reduction to provide a more powerful system at a low price.

Bad Block Recovery

The bad block handling feature provides mechanisms for detecting bad blocks and mapping them to surrogate blocks. The remainder of this section describes three ways that recovery from bad blocks can be handled.

Automatic Recovery from a Bad Block

One of the tasks that the UNIX system performs regularly is a check of the hard disk error log for bad blocks. This check is always performed when the system is in multi-user state (system state 2).

The following scenario explains the automatic process of handling bad blocks. Physical block 3 is a surrogate block on the disk for physical block 42. Block 42 is a block in the middle of a text file that is five data-blocks long. Block 3 has become a bad block since the file was last read. Now the file needs to be read again. When block 42 is reached, the driver for the integral disk sees that block 42 is mapped to block 3 and attempts to read block 3. But block 3 is now bad and cannot be read. When the integral disk driver determines that block 3 is unreadable, the following messages are sent to the system console:

```
WARNING: unreadable CRC hard disk error: maj/min = 7/0

        block # = 3


Disk Error Daemon: successfully logged error for block 3 on disk
maj=7 min=0
```

NOTE

CRC stands for Cyclic Redundancy Check, an error checking method. The numbers assigned to `maj` and `min` are the major and minor device numbers of the disk on which the error occurred. (See the "Device Identification Via Special Files" section of the "Storage Device Management" chapter for a complete description of major and minor device numbers.)

The attempt to read this text file has failed. When you notice the message on the console, run the `shutdown` command to change the system to single-user state (system state 1). While `shutdown` is running, the following message is sent to the system console:

```
Disk Error Daemon: Disk maj=7 min=0: 1 errors logged
```

In this scenario, a bad block has been identified, reported, and logged by automated mechanisms in the system. Most bad blocks are handled in this way. The following sections describe how these automated mechanisms work.

Identifying Disks

Disks are identified by their major and minor device numbers (on both single-disk and multi-disk computers). Messages printed by the bad block handling mechanisms use the major and minor numbers rather than any other names. The utilities of the bad block handling feature can be given these numbers as arguments when more specialized operations must be used.

Detecting Bad Blocks

The disk driver detects a bad block when the disk fails to access that block after several attempts. To be sure that the problem is not being caused by the position of a read or write head, the driver repositions the read and write heads between access attempts. When the driver is still unable to access this block, it can safely be assumed that the block is defective.

Reporting and Logging Bad Blocks

When a block is determined to be inaccessible, the disk driver reports the defective block to the bad block logging mechanism which, in turn, notifies the system administrator by sending a message to the system console. For example, in the scenario described above, the following message was sent to the console:

```
WARNING: unreadable CRC hard disk error: maj/min = 7/0
        block # = 3
```

This output is used as input to the hard disk error logger commands. In this case, for example, device 7/0 and block #3 would be arguments to the `hdelogger` commands.

The logging mechanism then attempts to record the error in the disk error log located in the isolated portion of the disk. If the error is logged successfully, the following message is sent to the system console:

```
Disk Error Daemon: successfully logged error for block 3 on disk
maj=7 min=0
```

The logging mechanism is run by the driver for the hard disk error log (`hdelog`) and by a hard disk error daemon process called `hdelogger` that is, in turn, run by the `/sbin/init` process. The error log driver provides both mechanisms for reports and access to the reserved disk areas needed by the bad block handling feature. (This driver can queue a maximum of 18 reports.)

The disk error daemon has another reporting role. When the system state changes (for example, when you turn on your computer, shut it off, or shut down to system state 1), a daemon process checks the error log. If the daemon finds outstanding bad block reports in a log, it sends a message to the system console. For example, in the scenario described earlier, the following message was sent:

```
Disk Error Daemon: Disk maj=7 min=0: 1 errors logged
```

The `hdelogger` daemon runs in system states 2, 3, and 4.

NOTE The bad block handling daemon does not run in system states 1, 5, or 6. System state 1 (also referred to as "s" or "S") is single-user state. System states 5 and 6 are used for returning to firmware and for rebooting the operating system, respectively.

Interactive Recovery from a Bad Block

Not all bad block errors can be handled automatically; recovery from some types of errors requires your assistance. Which recovery procedure is required depends on the system state of the computer when the error occurred.

Errors in System State 1 (Single-User State)

If errors happen while your system is in system state 1, the error reports stay queued until a system state is used in which the bad block handling daemon will also run. However, if you shut your system off or reboot it without going to another system state, the error reports in the queue are lost. When errors occur while in system state 1, only the messages from the logging mechanism and disk driver are sent to the system console.

If you get errors while in system state 1 and you are not ready to fix them (the mechanism for fixing them takes error reports from the queue as well as from the disk error logs), you can switch to another system state to force them to be logged. You will get a `successfully logged` message for each error that occurred. When all reports are logged, you can switch back to system state 1. When you do, a reminder message from the disk error daemon will be sent to the console.

System Panics and Firmware-Detected Errors

If there is a disk error that results in the loss of a critical operating-system path, such as the path to the swap space, the operating system panics. A system panic occurs after the reports from the logging mechanism and disk driver are sent to the console, but before the report is logged.

NOTE

If an error is detected by the firmware, the error is reported to the console, but it is not logged. In these cases, YOU MUST WRITE DOWN THIS CONSOLE REPORT. When the system comes back up, use the `/usr/sbin/hdeadd` command and manually add this handwritten report to the disk error queue.

The firmware will report a disk read or write error as:

```
Error Status: XXXX
```

where XXXX is a code indicating the cause of the error. Only a few of the possible values for XXXX (such as `$13` and `$22`) indicate a bad block. But because the firmware does not report the cylinder head and sector numbers of the bad block,

you must access the same file being accessed by the firmware when the system is in a single- or a multi-user state. The disk driver will then report these numbers on the console if it is a bad block. These numbers are then used as arguments to the `-B` option of the `hdeadd` and `hdefix` command — that is:

`-B` *cyl# head# sector#*

The following screen shows example messages that appear when a bad block causes a system panic. Assume that physical block number 463 is in your swap space. Although, unknown to you, this block has recently become bad, the operating system writes data into it. (These data cannot be read with the disk's current pattern of magnetic biases.) When the operating system tries to read this block, the disk driver determines that the block is unreadable, reports the condition, and fails the read. The swapper process runs next, discovers the read failure, and causes the panic. All process activity is halted at this point, including the disk error daemon, and the following messages are sent to the console.

```
WARNING: unreadable CRC hard disk error: maj/min = 7/0
        block # = 463

PANIC: swapseg: i/o error in swap
```

NOTE

Because of the system panic, the system cannot record this error. Therefore you must write down or print out the numbers 7/0 and 463.

Unless you are already in system state 5, you must bring the system down to system state 1 with the `shutdown` command (`/sbin/shutdown -y -g0 -i1`) to minimize the chances of getting another swap-generated panic. Once you are in system state 1, enter the following command:

```
hdeadd -a -D 7 0 -b 463
```

This command reports the bad block to the operating system's logging mechanism. If this swap error occurred on Saturday, January 13, 1990, at 02:01:00 hours, the full report would be as follows:

```
# hdeadd -a -D 7 0 -b 463
hdeadd: logging the following error report:
        disk maj= 7 min=0
        blkaddr= 463, timestamp= Sat Jan 13 02:01:00 1990
        readtype= 1, severity= 2, badrtcnt= 0, bitwidth= 0
WARNING: unreadable CRC hard disk error: maj/min = 7/0

         block # = 463


Disk Error Daemon: successfully logged error for block 463 on disk maj= 7 min= 0
#
```

You can also use the `hdelogger -f` command to double check the error status and obtain the following report (assuming this is the only error in the log).

```
# hdelogger -f

Disk Error Log: Full Report for maj= 7 min= 0
        log created:  Mon Jan  1 12:13:14 1990
        last changed: Sat Jan 13 02:01:04 1990
        entry count: 1
        phys blkno     cnt   first occurrence       last occurrence
0:      463             1    Jul 13 02:01:00 1990   Jul 13 02:01:00 1990
TOTAL: 1 errors logged
#
```

If the firmware or booter detected the error, it may not be possible to boot the system from your hard disk. However, if you have a tape device on your system and a bootable tape containing the bad block utilities, you can boot from this tape and try to repair the bad blocks as described in "Manual Recovery from a Bad Block" below. If you do not have a tape device on your system or if you do not have a bootable tape containing the bad block utilities, contact your service representative.

The Special Case of a Bad Error Log Block

Though unlikely, the new bad block could be the block in which the disk error log resides. Obviously, if the system cannot access the log because this log resides in a bad block, errors cannot be recorded.

Manual Recovery from a Bad Block

To fix a bad block manually:

1. Completely shut down the machine to system state 1.

```
shutdown -y -g0 -i1
```

2. Unmount all file systems except root.

```
umountall -k
```

3. Print the list of bad blocks to be mapped.

```
hdelogger -f
```

4. Map the bad block numbers into head/cylinder tuples.

```
# xformtrk -Id -Oc m323182
Enter bad block information as <logical dev> <block>  ;  end with a period
0 463
.
3 1 0
```

5. Map out the bad block(s) with `dinit`.

```
# dinit -n m323182 /dev/rdsk/c8d0s7
Enter bad head/cylinder/logical sector
3 1 0
```

Although `xformtrk` can map multiple block numbers into head/cylinder tuples, `dinit` only maps one block at a time. After each bad block is mapped out, you should run `fsck` on the affected partition. (In the example shown, this is unnecessary because block #463 is in the `swap` partition.)

6. Remove any reports for the bad block(s) from the error log.

```
hdeadd -d -D 7 0 -b 463
```

Dealing with Data Loss

Although the useful life of disk hardware is greatly extended with the bad block handling feature, once a bad block is logged, the data in the block are lost. You must be prepared to restore files or file systems from archives. When restoring data from archives, users may encounter unavoidable inconveniences in the form of lost files and lost work on existing files.

Under rare circumstances, a bad block may occur in the Volume Table of Contents (VTOC) block. In such a case, you may have to reformat the disk and restore the contents of the entire disk from an archive. Sometimes you may also need to restore the special code (the program) for booting the system (it is not in a file system).

6 File System Administration

DRAFT COPY
February 1, 1992
File: Cfiles

DRAFT COPY
February 1, 1992
File: Cfiles

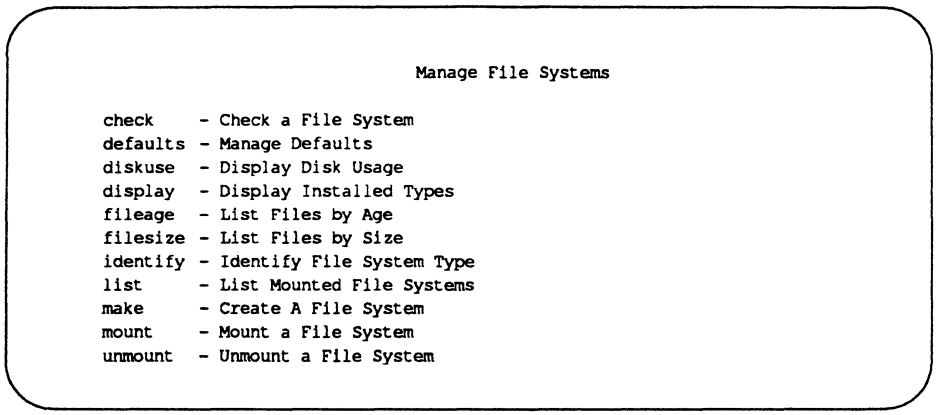
Prologue

This chapter provides the information required to administer disk file systems. Other file system types, such as `/proc`, are not discussed. (For information on `/proc`, see the manual page `proc`(4) in the *System Administrator's Reference Manual*. For information on remote file systems (e.g., `rfs` and `nfs`) see the *Network User's and Administrator's Guide*.)

You may approach file system administration in either of two ways: by using administrative menus or by issuing commands directly to the shell. Generally speaking, if you are new to system administration it is probably preferable to use the administrative menus, while if you are more experienced you may appreciate the greater speed and flexibility that may be attained by entering commands at the shell level.

To perform administrative tasks using the menus provided by the system, you must type `sysadm file_systems`. When you do so, you will reach the main menu for file system administration, which is shown below.

Figure 6-1: Main Menu for File System Administration



The menus are intended to be self-explanatory and will not be further described in this chapter.

If you prefer not to use the menus, you can perform administrative tasks by issuing commands directly to the shell. The following table shows the shell commands that are functionally equivalent to the selections on the main menu for file system administration.

Figure 6-2: Menus and Shell Commands for Performing Administrative Tasks

Task Description	Menu Item	Shell Command
Check a file system	`check`	`fsck`(1M)
Manage `vfstab` defaults	`defaults`	any editor
Display disk usage	`diskuse`	`df`(1M)
Display installed types	`display`	`crash`(1M)
List files by age	`fileage`	`ls -t`
List files by size	`filesize`	`du`(1M)
Identify a file system type	`identify`	`fstyp`(1M)
List file systems	`list`	`mount`(1M)
Create a file system	`make`	`mkfs`(1M)
Mount a file system	`mount`	`mount`(1M)
Unmount a file system	`umount`	`umount`(1M)

Many of the tasks listed in this table are described in detail later in this chapter. Complete information about each shell command is available in the *System Administrator's Reference Manual*.

Introduction

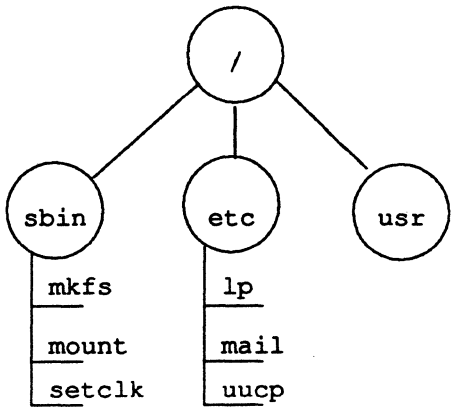
UNIX System V/68 or V/88 Release 4 supports a variety of file system types (*FSTypes*) of varying characteristics. Because of the great range of possible *FSTypes*, it is impossible to provide administrative information that would apply to every conceivable type. The material presented here describes the administration of the *FSTypes* `s5`, `ufs`, and `bfs` (the boot *FSType*).

We begin by describing the `s5`, `ufs`, and `bfs` *FSTypes*. We then discuss the relationship between a file system and a storage device, the methods of administering and maintaining a file system, and how to check a file system for consistency. In each section separate subsections describe the methods applicable to each of the *FSTypes* under discussion.

How a File System Is Organized

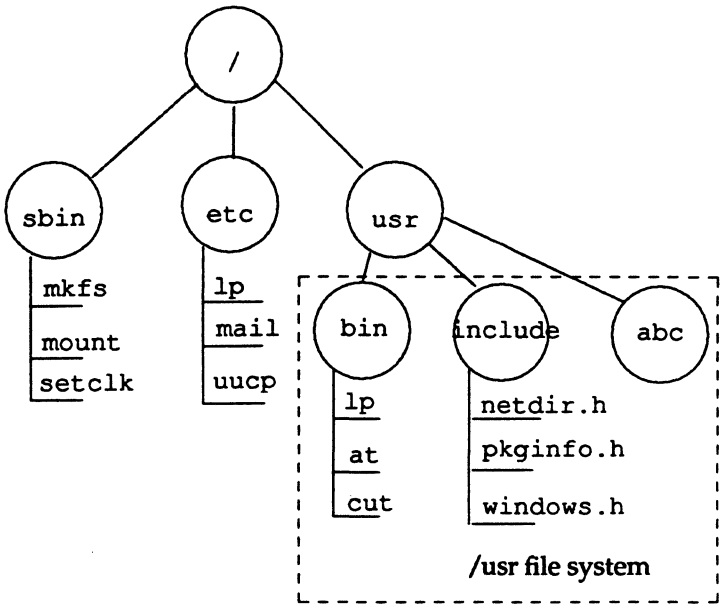
A primary function of the UNIX operating system is to support file systems. In the UNIX system a file is a string of bytes with no other structure implied. Files are attached to a hierarchy of directories. A directory is merely another type of file that the user is permitted to use, but not to write; only the operating system can write directories. The combination of directories and files make up a file system. Figure 6-3 shows the relationship between directories and files in a UNIX file system. The circles represent directories.

Figure 6-3: A UNIX File System



The starting point of any UNIX file system is a directory that serves as the root of that file system. Somewhat confusingly, in the UNIX operating system there is always one file system that has the name `root`. Traditionally, the root directory of the `root` file system is represented by a single slash (`/`). The file system diagramed in Figure 6-3 is a `root` file system. If we mount another file system onto the `root` file system at a directory called `/usr`, the result can be illustrated by the diagram in Figure 6-4.

Figure 6-4: Adding the `/usr` File System



A directory such as `/usr` that is used to form the connection between the `root` file system and another mountable file system is sometimes called a "leaf" or "mount point." Regardless of the term used, such a directory is the root of the file system that descends from it. The name of that file system is the name of the directory. In our example the name of the file system is `/usr`.

The diagrams in Figures 6-3 and 6-4 may be convenient representations of the file and directory structure of file systems, but they are not a particularly accurate or helpful way of illustrating how the UNIX operating system views a file system. The sections that follow describe *FSTypes* as they appear to the operating system.

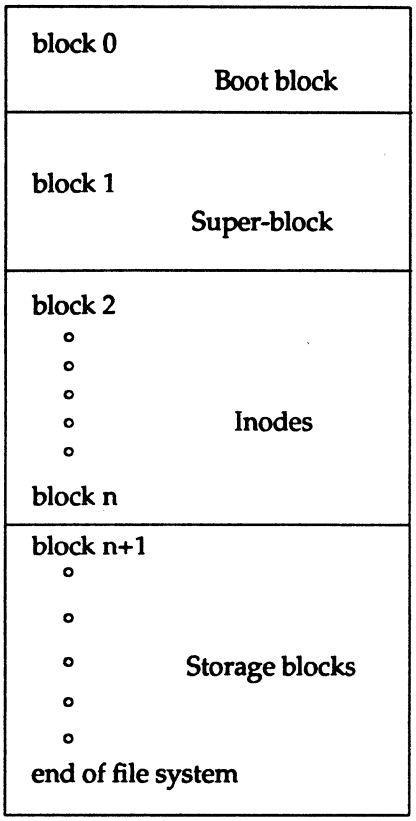
The s5 File System Type

The operating system views an s5 file system as an arrangement of addressable blocks of disk space that belong to one of four categories:

- block 0 (the boot block)
- block 1 (the super-block)
- a variable number of blocks comprising the i-list
- a variable number of storage blocks, most containing data, some containing the free list and indirect addresses

This layout is illustrated in Figure 6-5.

Figure 6-5: The UNIX View of an s5 File System



The s5 Boot Block

Although considered to be part of the file system, the boot block is not actually used by it. It is reserved for storing procedures used in booting the system. However, not all file systems are involved in booting. When a file system is not to be used for booting, the boot block is left unused.

The s5 Super-Block

Much of the information about the file system is stored in the super-block, including such things as:

- file system size and status
  - label (file system name)
  - size in logical blocks
  - read-only flag
  - super-block modified flag
  - date and time of last update
- inodes
  - total number of inodes allocated
  - number of free inodes
  - array of 100 free inode numbers
  - an index into the array of free inode numbers
- storage blocks
  - total number of free blocks
  - array of 50 free block numbers
  - an index into the array of free block numbers

Note that the super-block does not maintain complete lists of free inodes and free blocks, but only enough to meet current demands as the file system is used. At almost any time, unless the file system is close to running out of inodes and storage blocks, there are sure to be more free inodes and free blocks than are listed in the super-block.

s5 Inodes

The term "inode" (i-node) stands for information node. A list of inodes is called an i-list and the position of an inode in an i-list is called an i-number.

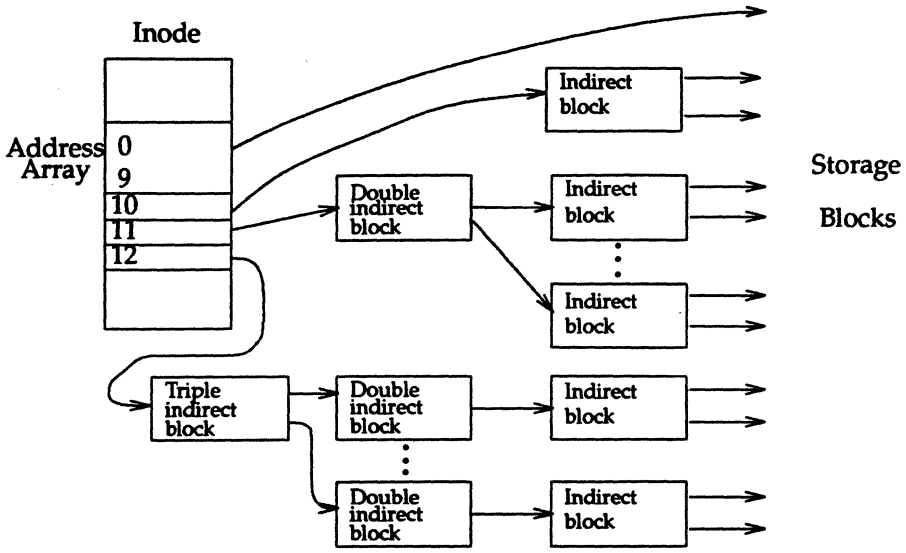
An inode contains all the information about a file except its name, which is kept in a directory. Inodes are 64 bytes long, so there are eight of them in a physical block. The length of an i-list is not fixed; it depends on the number of inodes specified when the file system is created. Specifically, an s5 inode contains:

- the type and mode of the file - the type can be regular, directory, block, character, symbolic link, or first-in first-out (FIFO), also known as named pipe; the mode is the set of read-write-execute permissions
- the number of hard links to the file
- the user-id of the owner of the file
- the group-id to which the file belongs
- the number of bytes in the file
- an array of 13 disk block addresses
- the date and time the file was last accessed
- the date and time the file was last modified
- the date and time the file was created

The array of 13 disk block addresses is the heart of the inode. The first ten are direct addresses; that is, they point directly to the first ten logical storage blocks of the contents of the file. If the file is larger than ten logical blocks, the 11th address points to an indirect block, which contains direct addresses instead of file contents; the 12th address points to a double indirect block, which contains addresses of indirect blocks. Finally, the 13th address in the array is the address of a triple indirect block, which contains addresses of double indirect blocks. Figure 6-6

illustrates this chaining of address blocks stemming from the inode.

Figure 6-6: The File System Address Chain for s5



The following table shows the number of bytes addressable by the different levels of indirection in the inode address array for s5 file systems. These numbers are calculated using the logical block size of the file system and the number of bytes (four) used to hold an address.

Logical Block Size	Maximum number of bytes addressable by Direct Blocks	Single Indirect Blocks	Double Indirect Blocks	Triple Indirect Blocks
1024 bytes	10240	256K	64M	16G
2048 bytes	20480	1M	512M	256G

The table shows the number of bytes addressable using the level of indirection in the column header plus all lower levels of addressing. For example, the table values for single indirect blocks also include bytes addressable by direct blocks; and the table values for triple indirect blocks include bytes addressable by direct blocks and single and double indirect blocks.

The theoretical maximum size of an s5 system file is the same as the size of a file addressable with triple indirection (shown in the last column of the table). In practice, however, file size is limited by the size field in the inode. This is a 32-bit field, so file sizes are limited to four gigabytes.

s5 Storage Blocks

The rest of the space allocated to the file system is occupied by storage blocks, also called data blocks. For a regular file, the storage blocks contain the contents of the file. For a directory, the storage blocks contain 16-byte entries. Each entry represents a file or subdirectory that is a member of the directory. An entry consists of two bytes for the i-number and 14 bytes for the filename of the member file or subdirectory.

s5 Free Blocks

Blocks not currently being used as inodes, as indirect address blocks, or as storage blocks are chained together in a linked list of free blocks. Each block in the list carries the address of the next block in the chain.

The ufs File System Type

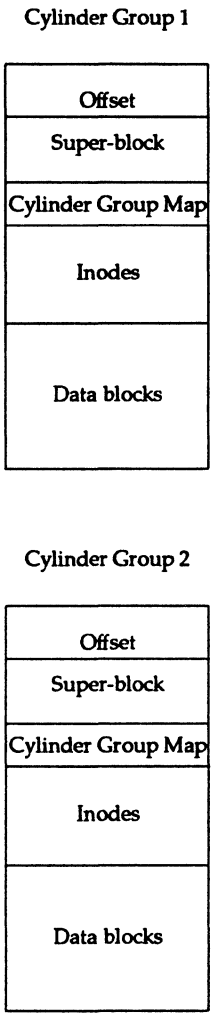
The ufs *FSType* is considerably more complex in its design than the s5 *FSType*. In addition to the four categories of addressable blocks found in s5, there are several additional information management disk areas. There is also a radically different method of allocating and managing these blocks. Of primary interest is the fact that multiple super-blocks are made during the mkfs procedure. One of the replicas is stored in each cylinder group, offset by a certain amount. For multiple platter disk drives, the offsets are calculated so that a super-block appears on each platter of the drive. So if the first platter is lost, an alternate super-block can be retrieved. For platters other than the top one in a pack, the leading blocks created by the offsets are reclaimed for data storage.

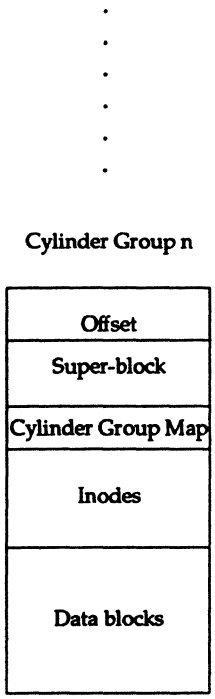
Kept with the super-block is a summary information block. This block is not replicated, but is grouped together with the first super-block, normally in cylinder group 0. This summary block is used to record changes that take place as the file system is used, and lists the number of inodes, directories, fragments, and blocks within the file system.

Another feature found in ufs is the "cylinder group map." This is a block of data found in each cylinder group that records the block usage within the cylinder. This information is kept directly following the super-block copy for that cylinder group.

To give an idea of the appearance of a typical ufs file system, the following diagram shows a series of cylinder groups in a generic ufs file system:

Figure 6-7: The UNIX View of a ufs File System





The ufs Boot Block

The boot block appears only in the first cylinder group (cylinder group 0) and is the first 8K in a partition. It is reserved for storing the procedures used in booting the system. If a file system is not to be used for booting, the boot block is left blank.

The ufs Super-Block

Much of the information about the file system is stored in the super-block. A few of the more important things it contains are:

- the size and status of the file system
- the label (file system name)
- the size of the file system in logical blocks
- the date and time of the last update
- the cylinder group size
- the number of data blocks in a cylinder group
- the summary data block

ufs Inodes

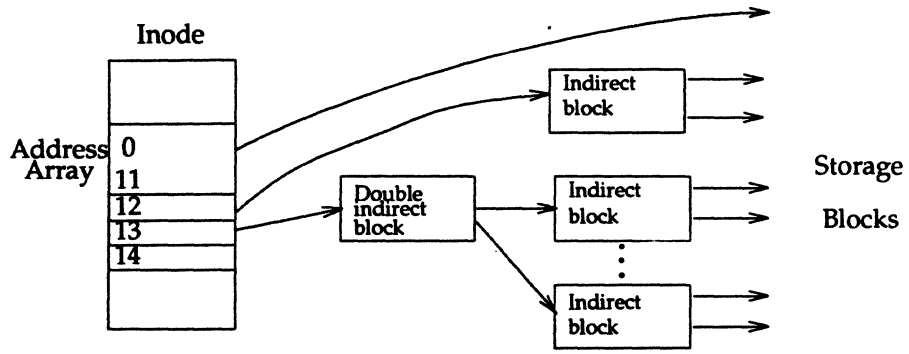
The inode information is kept in the cylinder information block. An inode contains all the information about a file except its name, which is kept in a directory. An inode is 128 bytes long. One inode is created for every 2K of storage available in the file system. This parameter can be changed when mkfs is used to create the file system, but it is fixed thereafter. A ufs inode contains:

- the type and mode of the file - the type can be regular, directory, block, character, symbolic link, or FIFO, also known as named pipe; the mode is the set of read-write-execute permissions
- the number of hard links to the file
- the user-id of the owner of the file
- the group-id to which the file belongs
- the number of bytes in the file
- an array of 15 disk block addresses
- the date and time the file was last accessed

- the date and time the file was last modified
- the date and time the file was created

The array of 15 disk addresses is the heart of the inode. The first 12 are direct addresses; that is, they point directly to the first 12 logical storage blocks of the contents of the file. If the file is larger than 12 logical blocks, the 13th address points to an indirect block, which contains direct addresses instead of file contents; the 14th address points to a double indirect block, which contains addresses of indirect blocks. The 15th address is unused. Figure 6-8 illustrates this chaining of address blocks stemming from the inode.

Figure 6-8: The File System Address Chain In a ufs File System



The following table shows the number of bytes addressable by the different levels of indirection in the inode address array for ufs file systems. These numbers are calculated using the logical block size of the file system and the number of bytes used to hold an address.

Logical Block Size	Max number of bytes addressable by Direct Blocks	Single Indirect Blocks	Double Indirect Blocks
4096 bytes	48K	4M	4G
8192 bytes	96K	16M	32G

The table shows the number of bytes addressable using the level of indirection in the column header plus all lower levels of addressing. For example, the table values for single indirect blocks also include bytes addressable by direct blocks; and the table values for indirect blocks include bytes addressable by direct blocks and single indirect blocks.

The theoretical maximum size of a ufs file system is the same as the size of a file addressable with double indirection. In practice, however, file size is limited by the size field in the inode. This is a signed 32-bit field, so file sizes are limited to two gigabytes. Because of the large size of ufs logical blocks, double indirect blocks rarely appear in ufs file systems. The result is that data retrieval in large files is much quicker than it would otherwise be.

ufs Storage Blocks

The rest of the space allocated to the file system is occupied by storage blocks, also called data blocks. The size of these storage blocks is determined at the time a file system is created, and can be either 4096 or 8192 bytes. Because of these large block sizes, and the potential for waste with small files, ufs also has a subdivision of a block called a fragment. When a file system is created the fragment size may be set to 512, 1024, 2048, or 4096 bytes. Fragments of 1024 bytes are the most common. For a regular file, the storage blocks contain the contents of the file. For a directory, the storage blocks contain entries that give the inode number and the filename. ufs filenames may be up to 255 bytes long. Each entry represents a file or subdirectory that is a member of the directory.

ufs Free Blocks

Blocks not currently being used as inodes, as indirect address blocks, or as storage blocks are marked as free in the block map kept in the cylinder group summary information block. This block also keeps track of fragments in order to prevent fragmentation from degrading disk performance excessively.

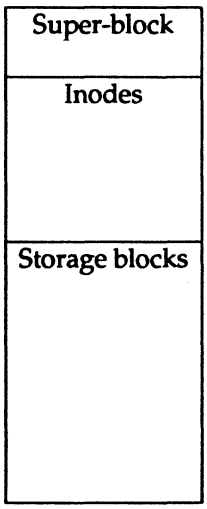
The bfs File System Type

The bfs *FSType* is a special-purpose file system. It contains all the stand-alone programs (for example, unix) and all the text files necessary for the boot procedures. See "The stand and boot Partitions" in the "Machine Management" chapter for more information.

The object of the bfs *FSType* is to allow quick and simple booting. It is for this reason that bfs was designed as a contiguous flat file system. bfs file systems can be mounted on the root directory only. Users may only create regular files; no directories or special files can be created in the bfs file system.

A bfs file system consists of three parts: the disk super-block, the inodes, and the storage or data blocks. The layout is illustrated in the following figure:

Figure 6-9: The UNIX View of a bfs File System



The bfs Super-Block

The following information is stored in the super-block:

- the magic number
- the size of the file system
  - the offset to the start of file system data (in bytes)
  - the offset to the end of file system data (in bytes)
- the sanity words

  There are four words used to promote sanity during compaction. They are used by the fsck command to recover if there has been a system crash at any time during the process of compaction. See "Compaction" in this chapter for more information about compaction.

bfs Inodes

The bfs equivalent of an inode (the bfs_dirent structure) contains all the information about a file except its name. File names are kept in the root directory, the only directory in the bfs file system. An inode is 64 bytes long. The number of inodes is defined when mkfs is used to create the file system. An inode contains:

- the inode number

  By convention this field is set to zero to indicate that the inode is available.
- the first data block
- the last data block
- the disk offset to the end-of-file (in bytes)
- the file attributes
- the type and mode of the file
- the user ID of the owner of the file

- the group ID to which the file belongs
- the number of hard links to the file
- the date and time the file was last accessed
- the date and time the file was last modified
- the date and time the file was created

bfs Storage Blocks

The remainder of the space allocated to the file system is taken up by storage blocks, also called data blocks. The size of the storage blocks is 512 bytes. The storage blocks are used to store the root directory and the regular files. For a regular file, the storage blocks contain the contents of the file. For the root directory, the storage blocks contain 16-byte entries. Each entry represents a file and consists of two bytes for the i-number and 14 bytes for the file name.

Managing Data Blocks

The data or storage blocks for a file are allocated contiguously. The data block after the last data block used in the file system is considered the next data block available to store a file. When a file is deleted, its data blocks are released; for the file system to reuse them, one of the following must be true:

- the file deleted must be the last file stored in the file system, or
- the system must detect the need for compaction and perform it.

Compaction

Compaction is a way of recovering data blocks by shifting files until the gaps left behind by deleted files are eliminated. This operation can be very expensive, but it is necessary because of the method used by `bfs` to store and delete files.

The system recognizes the need for compaction and performs it when:

- the system has reached the end of the file system and there are still free blocks available, or
- the system deletes a very large file and the file after it on disk is small and is the last file in the file system. (Small files are files of no more than ten blocks; large files are files of 500 or more blocks.)

The Relationship Between the File System and the Storage Device

In the UNIX system, file systems are kept on random-access disk devices. (A file system can be put on tape, but that is normally done only to create a backup copy in case the file system must be restored.) Before you can install a file system on a storage device, you must format the device. The material in this part of the chapter is a summary of material described in more detail in the "Storage Device Management" chapter.

Formatting the Storage Device

Before a disk can be used by the UNIX system it must be formatted into addressable sectors. A disk sector is a 512-byte portion of the storage medium that can be addressed by the disk controller. The number of sectors is a function of the size and number of surfaces of the disk device. Sectors are numbered from zero on up.

NOTE

Hard disk units on the Motorola reference platform are formatted when installed. The only time they might have to be reformatted would be after a catastrophic hardware failure. We recommend that you contact your service representative if such an event occurs.

Partitions

A partition consists of one or more sectors. Up to 16 partitions can be defined on a hard disk. On a hard disk, the `fmthard`(1M) command is used to associate the starting points of partitions with sector numbers. `prtvtoc` may be used to see the partitions assigned. `fmthard` gives the number of sectors allocated to a partition and a hex code tag that tells how the partition is to be used. Partition tags 0-8 are reserved. The list below shows how the tags may be used under `s5`.

Name	Number
UNASSIGNED	0
ROOT	2
SWAP	3
USR	4
VAR	5
STAND	6
BACKUP	7
HOME	8

The following figure is an example of the partitions for a 600-megabyte disk SCSI drive that has been configured to be the root device of a system.

Figure 6-10: Disk Partitions for a 600-Megabyte Drive

600-Megabyte Hard Disk Drive (blocks per cylinder = 324)				
Disk Partition	Use	Sector Offset	Size*	I-Nodes
m328_c0d0s0	root	648	90000	5600
m328_c0d0s1	swap	90648	70000	-
m328_c0d0s2	stand	160648	50000	496
m328_c0d0s3	usr	210648	300000	187207
m328_c0d0s4	var	510648	150000	9344
m328_c0d0s5	home	660648	20000	1248
m328_c0d0s7	entire disk	0	1195740	-

* Size in 512-byte blocks.

The table shows five file systems defined on this drive: `root`, `/usr`, `/stand`, `/var`, and `/home`. Space has also been set aside for the swap space. Tables showing the default partitioning for all supported devices are in Appendix A.

The disk partition naming convention takes the form *prefix*`_c`*X*`d`*Y*`s`*suffix*, where *prefix* uniquely defines the type of device, *X* specifies the controller number (starting from zero) of the stated device type, *Y* specifies the logical device number (starting from zero) for the device attached to the stated controller, and *suffix* specifies device dependent information. See `intro`(7) for more information.

Size Limitations

The maximum number of blocks that can be allocated to a file system is close to the total number of sectors on the disk device. This maximum may be reduced by space set aside for swapping or paging. Also, recall that under `ufs` a two gigabyte upper limit is imposed by the size field in the inode.

The maximum number of inodes that can be specified for `s5` is 65,500. The `ufs` *FSType* does not have a rigid upper limit. Rather, roughly one inode is created automatically for each 2K of data space, although this default can be overridden at the time the file system is created.

The size of a block on a disk is 512 bytes, the same as a disk sector. However, internal file I/O works with logical blocks rather than with 512-byte physical blocks. The size of a logical block is set with the `mkfs`(1M) command. `s5` uses logical block sizes of 1024K and 2048K bytes; the default is 1024K byte blocks. `ufs`, on the other hand, uses logical block sizes of 4096K and 8192K bytes; the default is 8192K byte blocks. For optimal `ufs` performance, use a 4K block size.

Because of the large block size used by `ufs`, small files could waste a lot of space. To deal with this, `ufs` has a subdivision of a block called a fragment. When a `ufs` file system is created using `mkfs`, the fragment size may be set to 512, 1024, 2048, or 4096 bytes. When a block must be fragmented, the remaining fragments are made available to other files to use for storage. The information on which fragments are in use and which are available is kept in the cylinder group summary information block.

Administering a File System

The Generic Administrative Commands

The Virtual File System architecture allows multiple file system types (*FSTypes*) to coexist in the UNIX kernel. Each *FSType* has certain characteristic features that it does not share with any other *FSTypes*. However, the file system administrative commands provide a common interface that allows the administrator to maintain file systems of differing types.

The following commands for file system administration are unified commands and can be used on multiple *FSTypes*:

- `dcopy`(1M)
- `df`(1M)
- `ff`(1M)
- `fsck`(1M)
- `fsdb`(1M)
- `fstyp`(1M)
- `labelit`(1M)
- `mkfs`(1M)
- `mount`(1M)
- `mountall`(1M)
- `ncheck`(1M)
- `umount`(1M)
- `umountall`(1M)
- `volcopy`(1M)

Most of these commands may be invoked as follows:

command `[-F` *FSType*`]` `[-V]` `[`*current_options*`]` `[-o` *specific_options*`]` *operands*

The `-F` is used to specify the *FSType* on which the command must act. The *FSType* must be specified on the command line or must be determinable from `/etc/vfstab` by matching an entry in that file with one of the *operands* specified. (See the section below for information on the `vfstab` file.)

The -V option causes the command to echo the completed command line. The echoed line will include additional information derived from the `vfstab` file. This option can be used to verify and validate the command line. It does not cause the command to execute.

current_options are options supported by the `s5`-specific module of the *command*.

The -o option is used to specify *FSType*-specific options. *specific_options* are options specified in a comma-separated list of keywords and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command.

operands are *FSType*-specific; consult the *FSType*-specific manual page of the command for a detailed description.

The `vfstab` File

Since the generic commands work on multiple *FSTypes* (for example, `mount` can mount an `s5` or a `ufs` file system among other types), they require *FSType*-specific information that may be provided explicitly on the command line or implicitly through the file system table `/etc/vfstab`.

The file system table contains a list of default parameters for each file system. It is an ASCII file that should be maintained by the system administrator. Each record contains space separated information about a file system in the format:

special fsckdev mountp fstype fsckpass automnt mntopts

The meaning of each field is as follows:

- *special*: the block special device for local devices or the resource name for remote file systems (for example, `rfs` and `nfs`). For more information on remote file systems, see the *Network Applications Guide*.
- *fsckdev*: the character special device that corresponds to the *special*. The block special device is used if the character special device is not available. Use a "-" where there is no applicable device.
- *mountp*: the default mount directory (mount point)
- *fstype*: the type of the file system on the special device

- *fsckpass*: the pass number to be used by `ff`, `fsck`, and `ncheck` to decide whether to check the file system automatically. Use "-" to inhibit automatic checking of the file system.
- *automnt*: `yes` or `no` for whether the file system should be automatically mounted by `mountall` when the system is booted.
- *mntopts*: a list of options, separated by commas, that will be used in mounting the file system. Use "-" to indicate no options. See `mount`(1M) for a list of the available options.

Lines beginning with the # character are comments.

Listing Installed File System Types

Use the `crash`(1M) command to display a list of *FSTypes* installed in the kernel. The following will produce such a list:

```
crash <<!
vfssw
!
```

In addition to the familiar *FSTypes*, `crash` will also list certain internal *FSTypes*, such as `specfs`, that have no user interface.

Identifying the Type of an Unmounted File System

Most commands that are used in file system administration require that the *FSType* of a file system be provided on the command line or in the file system table. Most of these commands also attempt to distinguish the type of a file system by themselves, so if the administrator provides the wrong type the command may fail. However, it is important to specify the correct type because file systems may be damaged if a command fails to detect an administrator's error and an operation applicable only to one type of file system is applied to another.

Sometimes the administrator will have to try to determine the type of an unmounted file system type, either because the `vfstab` file contains outdated information, or because it contains no information at all. The command `fstyp`(1M) uses heuristics to determine the type of an unmounted file system.

`fstyp` determines and displays the file system type on `stdout`. If it cannot determine the type it echoes *unknown_fstyp(no matches)* on `stderr`.

Creating a File System

Using `mkfs`

Once a disk is formatted the next step is to define the file system. The `mkfs`(1M) command is used for this purpose. The generic format of the `mkfs`(1M) command follows:

```
mkfs [F FSType] [-V] [-m] [current_options] [-o specific_options] \
    special [operands]
```

(The above command line is shown on two lines for readability.)

`mkfs` constructs a file system by writing on the *special* file, which must be the first argument. The file system is created based on the *FSType* specified using the `-F` option, the *specific_options*, and *operands* specified on the command line.

The `-F` is used to specify the *FSType* on which the command must act. The *FSType* must be specified on the command line or must be determinable from `/etc/vfstab` by matching an entry in that file with one of the *operands* specified. (See the section below for information on the `vfstab` file.)

The `-V` option causes the command to echo the completed command line. The echoed line will include additional information derived from `/etc/vfstab`. This option can be used to verify and validate the command line. It does not cause the command to execute.

The `-m` option is used to return the command line which was used to create the file system. The file system must already exist and this option provides a means of determining the attributes used in constructing the file system. Note that file systems cannot be constructed for all *FSTypes*. Take care to specify valid *FSTypes*.

current_options are options supported by the `s5`-specific module of the *command*.

The `-o` option is used to specify *FSType*-specific options if any. *specific_options* are options specified in a comma-separated list of keywords and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command.

operands are *FSType*-specific; consult the *FSType*-specific manual page of the command for a detailed description.

Choosing Logical Block Size

Logical block size is the size of the blocks that the UNIX kernel uses to read or write files. The logical block size is usually different from the physical block size, which is the size of the smallest block that the disk controller can read or write (usually 512 bytes).

The `mkfs`(1M) command allows the administrator to specify the logical block size of the file system. By default, the logical block size is 1024 bytes (1K) for `s5` file systems and 8192 bytes (8K) for `ufs` file systems. The `root` and `usr` file systems are delivered as s5 1024-byte (1K) file systems. Besides 1K file systems, the `s5` file system also supports 2K file systems. The `ufs` file system supports 4096-byte (4K) and 8K systems.

To choose a reasonable logical block size for your system, you must consider both the performance desired and the available space. For information on disk performance, refer to the "Performance Management" chapter. For most `ufs` systems, an 8K file system with a 1K fragment size gives the best performance, while for most `s5` systems, a 1K file system provides the best performance: both offer a good balance between disk performance and use of space in primary memory and on disk. For special applications running under `s5` (such as `s5` file servers) that use large executable files or large data files, a 2K file system may be a better choice. See the "Performance Management" chapter for more information.

Using `mkfs` to Create an `s5` File System

When used to make an `s5` file system, the `mkfs` command builds a file system with a `root` directory and a `lost+found` directory. It is usually invoked in one of the following ways:

`mkfs [-F s5] [-b` *blocksize*`]` *special blocks*`[:`*inodes*`] [`*gap blocks/cyl*`]`

`mkfs [-F s5] [-b` *blocksize*`]` *special proto* `[`*gap blocks/cyl*`]`

As discussed earlier, the file system type (`s5`) need only be specified on the command line if no entry has been set up for *special* in the `vfstab`(4) file by the administrator. See the section "The `vfstab` File".

Notice that neither form of invocation names the file system that is to be created (for this function see `labelit`(1M)); instead, both forms identify the file by the filename of the special device file on which it will reside. The special device file, traditionally located in the directory `/dev`, is associated with the identifying controller and unit numbers (major and minor, respectively) for the physical device.

In the first form of invocation, the only other information that must be furnished on the `mkfs` command line is the number of 512-byte blocks the file system is to occupy. The second form lets you include that information in a prototype file that can also define a directory and file structure for the new file system, and it even allows for reading in the contents of files from an existing file system.

Both forms of invocation let you specify information about the interrecord gap and the blocks per cylinder. If this information is not given on the command line, default values are used. By default, the file system has a logical block size of 1024K bytes. With the `-b` option, you can specify a logical block size of 1024K bytes, or 2048K bytes. In the first form of invocation, even though the number of blocks in the file is required, the number of inodes may be omitted. If the number of inodes is omitted, the command uses a default value of one inode for every four logical storage blocks, rounding down to a modulo 16 value if necessary to fill the final inode block.

If you use the first form of invocation, the file system is created with a `root` directory and a `lost+found` directory. If you use a prototype file, as noted above, it may include information that allows the command to build and initialize a directory and file structure for the file system. The format of a prototype file is described on the `mkfs`(1M) manual page.

Summary: Creating and Converting `s5` File Systems

Here is a summary of the steps in creating a new file system or converting an old one to a new logical block size:

1. If the new file system is to be created on a disk partition that contains an old file system, back up the old file system. For information, see the chapters on backup and restore in this guide. To back up systems with one or more hard disks, use `cpio`(1).
2. If the new file system is to be created from an old file system, run the `labelit` command, which reports the mounted file system name and the physical volume name of the old file system (see `volcopy`(1M)). These labels are destroyed when you make the new file system, so you must

restore them.

3. If the new file system is to be created from an old file system and the new file system will have a larger logical block size, then, because of fragmentation, the new file system will allocate more disk blocks for data storage than the old. Use the `fsba`(1M) command to find out the space requirements of the old file system with the new block size.

   Use the information you get from the `fsba` command to make sure that the disk partition to be used for the new file system is large enough. Use the `prtvtoc`(1M) command to find out the size of your current disk partitions. If the new file system requires a disk repartition, see "Formatting Hard Disks and Tapes" in the "Storage Device Management" chapter.

4. Use the `mkfs`(1M) command with the `-b` option to make the new file system with the appropriate logical block size.

5. Run the `labelit` command to restore the file system and volume names.

6. Populate the new file system—for example, do a restore from a file system backup, or, if your system has two hard disks, do a `cpio` from a mounted file system. (The `volcopy`(1M) and `dd`(1M) commands copy a file system image; they cannot convert logical block size.)

Using `mkfs` to Create a `ufs` File System

When used to make a `ufs` file system, the `mkfs` command builds a file system with a `root` directory and a `lost+found` directory. The number of inodes is calculated as a function of the file system size.

The syntax for the `mkfs` command when it is used to create a `ufs` file system is the following:

```
mkfs -F ufs [-o specific_options] special size
```

The *specific_options* are a comma-separated list that allow you to control the parameters of the file system. The more important ones are as follows (for a complete list, see the `ufs`-specific `mkfs`(1M) manual page):

- `nsect` - the number of sectors per track on the disk. The default is 36.

- `ntrack` - the number of tracks per cylinder on the disk. The default is 9.
- `bsize` - the primary block size, in bytes, for files on the file system. It must be a power of two, currently selected from 4096 or 8192 (the default).
- `fragsize` - the smallest amount of disk space, in bytes, that will be allocated to a file. It must be a power of two, currently selected from the range 512 to 8192. The default is 1024.
- `cgsize` - the number of disk cylinders per cylinder group. This number must be in the range 1 to 32. The default is 16.
- `free` - the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, you must be a privileged user to allocate disk blocks. The default is 10.

A sample invocation follows:

```
mkfs -F ufs -o bsize=4096,nsect=36,ntrack=9 \
/dev/rdsk/c0d0s2 35340
```

(The above command line is shown on two lines for readability.)

Summary: Creating and Converting `ufs` File Systems

Here is a summary of the steps required to create a new file system or convert an old one to a new logical block size:

1. If the new file system is to be created on a disk partition that contains an old file system, back up the old file system. For information, see the chapters on backup and restore in this guide. To back up systems with one or more hard disks, use `cpio`(1).
2. If the new file system is to be created from an old file system, run the `labelit` command, which reports the mounted file system name and the physical volume name of the old file system (see `volcopy`(1M)). These labels are destroyed when you make the new file system, so you must restore them.
3. Use the `mkfs`(1M) command to make the new file system with the appropriate logical block size. The `mkfs`(1M) command is described in this section.

4. Run the `labelit` command to restore the file system and volume names.
5. Populate the new file system—for example, do a restore from a file system backup, or, if your system has two hard disks, do a `cpio`(1M) from a mounted file system. (The `volcopy`(1M) and `dd`(1M) commands copy a file system image; they cannot convert logical block size.)

Using `mkfs` to Create a `bfs` File System

When used to make a `bfs` file system, the `mkfs` command builds a file system with a `root` directory.

The syntax for the `mkfs` command when making a `bfs` file system is as follows:

`mkfs [-F bfs]` *special blocks* [*inodes*]

If the number of inodes is not specified on the command line, the default number of inodes is calculated as a function of the file system size.

Although any disk can have multiple boot file systems defined on it, you will not normally want more than one boot file system on one disk.

The following procedure shows how to define a new boot file system and assumes that the disk you are using is already bootable. See the section "Making New Bootable Disks" in the "Machine Management" chapter for instructions on making new bootable disks.

Defining a New Boot File System on a Bootable Disk

1. Use the `prtvtoc`(1M) command to identify the type and size of the current disk partitions on the disk. If your new `bfs` file system requires a disk repartition, see "Making New Bootable Disks" in the "Machine Management" chapter for information on partitioning a bootable disk.
2. Use the `mkfs`(1M) command to make a `bfs`file system in the appropriate partition of the disk.
3. Mount the new boot file system.
4. Populate the new file system; that is, copy into the new `bfs` file system all the required bootable programs and data files used during the boot procedure. See "The `stand` and `boot` Partitions" in the "Machine Management" chapter for information about these files.

Mounting and Unmounting File Systems

For a file system to be available to users, it must be mounted. The `root` and `/usr` file systems are always mounted as part of the boot procedure. The `/usr` file system may be on the same disk device as the `root` file system. The `mount` command that makes these two file systems available is contained in start-up shell procedures.

The `mount` command causes the mounted disk device and the mounted-on directory (the "mount point") to be associated with certain other information (such as the *FSType*, the mount options used during the mount, and the time the mount was performed) in the file `/etc/mnttab` (see `mnttab`(4)). For example, the command

```
mount -F s5 /dev/dsk/c1d0s2 /usr
```

tells the system to mount `/dev/dsk/c1d0s2` as an `s5` file system that begins at the directory `/usr`, while the command

```
mount -F ufs /dev/dsk/c1d0s2 /usr
```

tells the system to mount `/dev/dsk/c1d0s2` as a `ufs` file system that begins at the directory `/usr`.

If you try to change directories (`cd`(1)) to a directory in the `/usr` file system before the `mount` command is issued, the `cd` command will fail. Until the `mount` command completes, the system does not know about any of the directories in the `/usr` file system. True, there is a directory `/usr` (it must exist at the time the `mount` command is issued), but the file system below that remains inaccessible until the `mount` command completes.

Unmounting is frequently a first step before using other commands that operate on file systems. For example, `fsck`, which checks and repairs a file system, works on unmounted file systems. Unmounting is also an important part of the process of shutting the system down.

Maintaining a File System

Once a file system has been created and made available, it must be maintained regularly so that its performance remains satisfactory and so that it does not develop inconsistencies. The maintenance to ensure satisfactory performance will be dealt with in the rest of this section, that to ensure consistency in the next.

There are four tasks that should be part of routine maintenance if the administrator wants to be sure that the performance of the file system will be satisfactory. All of them are aimed at ensuring that disk space does not become so scarce that system performance is degraded. They are:

- monitoring the percent of disk space used
- monitoring files and directories that grow
- identifying and removing inactive files
- identifying large space users

Monitoring the Percent of Disk Space Used

Monitoring disk space may be done at any time to see how close to capacity your system is running. Until a pattern has emerged, it is advisable to check every day. To do this, use the `df`(1M) command as follows:

```
df -t
```

The `-t` option causes the total allocated blocks and files to be displayed, as well as free blocks and files. When no file systems are named, information about all mounted file systems is displayed. If information on unmounted file systems is needed the file system name must be specified. For more information on the numerous options available to `df`, see the `df`(1M) manual page.

Monitoring Files and Directories that Grow

Almost any system that is used daily has several files and directories that grow through normal use. Some examples are:

File	Use
/var/adm/wtmp	history of system logins
/var/adm/sulog	history of su commands
/var/cron/log	history of actions of /usr/sbin/cron
/var/help/HELPLOG	actions of /usr/bin/help
/var/spell/spellhist	words that spell(1) fails to match

The frequency with which you should check growing files depends on how active your system is and how critical the disk space problem is. A good way to limit the size of such files is the following:

```
tail -50 /var/adm/sulog > /var/tmp/sulog

mv /var/tmp/sulog /var/adm/sulog
```

This sequence puts the last 50 lines of /var/adm/sulog into a temporary file, and then it moves the temporary file to /var/adm/sulog, thus truncating the file to the 50 most recent entries.

Identifying and Removing Inactive Files

Part of the job of cleaning up heavily loaded file systems involves locating and removing files that have not been used recently. The find(1) command locates files that have not been accessed recently. find searches a directory tree beginning at a point named on the command line. It looks for filenames that match a given set of expressions, and when a match is found, performs a specified action on the file.

```
find /home -type f -mtime +60 -print > \
   /var/tmp/deadfiles &
```

(The above command line is shown on two lines for readability.)

Here is what the example shows:

/home	specifies the pathname where find is to start. Presumably, your machine is organized in such a way that inactive user files will not often be found in the root file system.

`-type f` tells `find` to look only for regular files, and to ignore special files, directories, and pipes.

`-mtime +60` says you are interested only in files that have not been modified in 60 days.

`-print` means that when a file is found that matches the `-type` and `-mtime` expressions, you want the pathname to be printed.

`> /var/tmp/deadfiles &`
directs the output to a temporary file and indicates that the process is to run in the background. This is a sensible precaution if your experience tells you to expect a substantial amount of output.

Identifying Large Space Users

Two commands provide useful information: `du`(1M) and `find`(1).

`du` produces a summary of the block counts for files or directories named in the command line. For example:

```
du /home
```

displays the block count for all directories in the `/home` file system. Optional arguments allow you to refine the output somewhat. For example, `du -s` may be run against each user's login to monitor individual users.

The `find` command can be used to locate specific files that exceed a given size limit.

```
find /home -size +10 -print
```

This example produces a display of the pathnames of all files (and directories) in the `/home` file system that are larger than ten (512-byte) blocks.

Quotas

The quota system is built around limits on the two principal resources of a file system: inodes and data blocks. For each of these resources, users may be assigned quotas. A quota in this case consists of two limits, known as the soft and hard limits. The hard limit represents an absolute limit on the resource, blocks or inodes, that the user may never exceed under any circumstances. Associated with the soft limit is a time limit set by the administrator. Users may exceed the soft limits assigned to them, but only for a limited amount of time - the time limit set by the administrator. This allows the user to temporarily exceed limits if needed, as long as they are back under those limits before the time limit expires. An example of such a situation might be the generation of a large file that is then printed and deleted.

In summary, for each user, you can assign quotas (soft and hard limits) for both blocks and inodes. You can also define a time limit that applies to all users on a file system indicating how long they can exceed the soft limits. There are actually two time limits: one for blocks, and one for inodes. You may define different time limits for different file systems. Also, users may have different quotas set on different file systems.

Using Quotas

Before turning quotas on for a file system for the first time:

- If the quotas are for a file system listed in `/etc/vfstab`, enter an `rq` in the `mntopts` field for that file system. If there is an `rw` in that field in the table, it should be replaced by `rq`.
- Mount the file system and `cd` to the mount point. Create a file called `quotas`. This file should be owned by `root`, and not writable by others.
- Execute `edquota -t` to change the time limits for exceeding the soft limits for blocks owned, and/or inodes owned. These limits are initially set to the values defined for `DQ_FTIMELIMIT` and `DQ_BTIMELIMIT` in `/usr/include/sys/fs/ufs_quota.h`. This is normally 1 week. If you leave either time limit (the one for exceeding the block limit or the one for exceeding the inode limit) at 0, or if you set either limit to 0, the default values will apply. You can, of course, change them to something else.

- Execute `edquota`, with or without the `-p` option, to set user quotas. Once you have set the quotas for a user, you can use the `-p` option to set the same quotas for another user. Note that because you are not limited to UIDs that are already being used, you may set quotas for future users.

Before turning on quotas on a file system, always run `quotacheck` on the file system. This will sync up the quotas with the actual state of the file system, so that if the file system has been used since the last time the quotas were turned on, all of the quotas will be updated to reflect the current state. This also provides a sanity check on the `quotas` file.

Use `quotaon` to turn quotas on, and `quotaoff` to turn them off. If you use the `-a` option with either, the command will execute the desired action on each `ufs` file system with `rq` in the `mntopts` field of its `vfstab` entry. Otherwise, you must invoke the command on each individual file system.

To report on quotas an administrator can use `repquota` or `quot` to get information on all users on a file system, or use `quota` to get information on a single user. Normal users can use `quota` to get information on their own quotas; they cannot get information on anyone else's quotas.

The Effects of Quotas on the User

The following are the major effects of the use of quotas on users:

- If a user exceeds his/her soft limit for blocks or inodes, the timer is started. If the user then reduces usage to a level under the soft limit, the timer is turned off and all is well. But if the user still has not reduced usage to an appropriate level when the timer expires, any further attempts by the user to acquire more file system resources will fail and the user will receive error messages saying that the file system is full. These messages will persist until the user has reduced usage to a level below the soft limit.
- If a user tries to exceed the hard limit at any time, the attempt will fail and the utility will indicate that it has run out of space.
- Because no warning is given when the user has exceeded the soft limit, users should be advised to run `quota` frequently. Users should be encouraged to include `quota` in their `.profile` so that it runs when they log in.

Checking a File System for Consistency

When the UNIX operating system is brought up, a consistency check of the file systems should always be done. Often this check is done automatically as part of the power-up process. Included as part of that process is a sanity check of each file system on the hard disk using `fsck -m`. The sanity check returns a code for each file system indicating whether the consistency checking and repair program, `fsck`(1M), should be run.

Use `fsck` to check file systems not mounted routinely as part of the power-up process. If you discover inconsistencies, you must take corrective action before the file systems are mounted. The remainder of this section is designed to acquaint you with the command line options of the `fsck` utility, the type of checking it does in each of its phases, and the repairs it suggests.

File system corruption, while serious, is not all that common. Most of the time a check of the file systems finds everything all right. The reason we put so much emphasis on file system checking is that if errors are allowed to go undetected, the loss can be substantial.

The `fsck` Utility

The file system check (`fsck`) utility is an interactive file system check and repair program. `fsck` uses the information contained in the file system to perform consistency checks. If an inconsistency is detected, a message describing the inconsistency is displayed. At that point you may decide whether to have `fsck` ignore the inconsistency or attempt to fix it. Reasons you might choose to have `fsck` ignore an inconsistency are that you think the problem is so severe that you want to fix it yourself, or that you plan to go back to an earlier version of the file system. Whatever your decision, you should not ignore the inconsistencies `fsck` reports. File system inconsistencies do not repair themselves. If they are ignored, they get worse.

The `fsck` administrative command is used to run the `fsck` utility to check and repair inconsistencies in a file system. With the exception of the `root` file system, a file system should be unmounted while it is being checked. If this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards if the file system checked is a critical one.

The `root` file system should be checked only when the computer is in run level S and no other activity is taking place in the machine. The system should be rebooted immediately afterwards.

The generic format of the `fsck` command follows:

`fsck` [`-F` *FSType*] [`-V`] [`-m`] [*special ...*]

`fsck` [`-F` *FSType*] [`-V`] [*current_options*] [`-o` *specific_options*] [*special ...*]

The `-F` is used to specify the *FSType* on which the command must act. The *FSType* must be specified on the command line or must be determinable from `/etc/vfstab` by matching an entry in that file with the *special* specified.

The `-V` option causes the command to echo the completed command line. The echoed line will include additional information derived from `/etc/vfstab`. This option can be used to verify and validate the command line. It does not cause the command to execute.

The `-m` is used to perform a sanity check only. This option is usually used before mounting file systems because it lets the administrator know whether the file system needs to be checked.

current_options are options supported by the `s5`-specific module of the *command*.

The `-o` option is used to specify *FSType*-specific options if any. *specific_options* are options specified in a comma-separated list of keywords and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command.

If the file system is inconsistent, the user is prompted for concurrence before each correction is attempted.

NOTE

Some corrective actions will result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output.

The default action for each correction is to wait for the user to respond `yes` or `no`. If the user does not have write permission, `fsck` defaults to a `no` action.

In the rest of this chapter we will see how to use `fsck` to check `s5`, `ufs`, and `bfs` file systems. The information is presented separately for each *FSType*. Although this results in a certain amount of repetition, it is hoped that it will avoid confusion.

Checking s5 File Systems

The following is the s5-specific format of the fsck command:

```
fsck [-F s5] [generic_options] [special...]

fsck [-F s5] [generic_options] [-y] [-n] [-p] [-sX] [-SX] [-tfile] \
[-l] [-q] [-D] [-f] [special...]
```

(The second command line is shown on two lines for readability.)

The options are as follows:

-y	Specifies a yes response for all questions. This is the normal choice when the command is being run as part of a shell procedure. It generally causes fsck to correct all errors.
-n	Specifies a no response for all questions. fsck will not write the file system.
-p	This option causes fsck to correct any innocuous inconsistencies. Unreferenced blocks, misaligned directories, incorrect link counts and bad free lists are some examples of inconsistencies that are automatically corrected.
-sX	Specifies an unconditional reconstruction of the free list. The X argument specifies the number of blocks-per-cylinder and the number of blocks to skip (rotational gap). The default values are those specified when the file system was created. The formats for some common disk drives are as follows for 1K file systems. See the "Using mkfs to Create an s5 File System" section of this chapter for more information.
-SX	Specifies a conditional reconstruction of the free list, to be done only if corruption is detected. The format of the X argument is the same as described above for the -sX option.

`-t`*file*	Specifies a scratch file for use in case the file system check requires additional memory. If this option is not specified, the process asks for a filename when more memory is needed.
`-l`	This option causes damaged files to be identified by their logical names in addition to the inode numbers.
`-q`	Specifies a "quiet" file system check. Output messages from the process are suppressed.
`-D`	Checks directories for bad blocks. This option is used to check file systems for damage after a system crash.
`-f`	Specifies that a fast file system check be done. Only Phase 1 (check blocks and sizes) and Phase 5 (check free list) are executed for a fast check. Phase 6 (reconstruct free list) is run only if necessary.
special	Names the special device file associated with a file system. If no device name is specified, `fsck` checks all file systems named in `/etc/vfstab` with a numeric `fsckpass` field.

Sample Command Use

The command line below shows `fsck` being entered to check the `usr` file system. No options are specified. The system response means that no inconsistencies were detected. The command operates in phases, some of which are run only if required or in response to a command line option. As each phase is completed, a message is displayed. At the end of the program a summary message is displayed showing the number of files (inodes) used, blocks used, and free blocks.

```
fsck -F s5 /dev/rdsk/c1d0s2

/dev/rdsk/c1d0s2
File System: usr Volume: usr

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
289 files 6522 blocks 3220 free
```

s5 File System Components Checked by fsck

Before describing the phases of `fsck` and the messages that may appear in each, we will review the components of an `s5` file system and describe the kinds of consistency checks that are applied to each.

Super-Block

Every change to the file system blocks or inodes modifies the super-block. If the CPU is halted, and the last command involving output to the file system is not a `sync` command, the super-block will almost certainly be corrupted. The super-block can be checked for inconsistencies involving:

- file system size
- inode list size
- free-block list
- free-block count
- free inode count

File System Size and Inode List Size

The number of blocks in a file system must be greater than the number of blocks used by the super-block plus the number of blocks used by the inode list. The number of inodes must be less than the maximum number allowed for the file system type. While there is no way to check these sizes precisely, `fsck` can check that they are within reasonable bounds. All other checks of the file system depend on the reasonableness of these values.

Free-Block List

The free-block list starts in the super-block and continues through the free-list blocks of the file system. Each free-list block can be checked for

- list count out of range
- block numbers out of range
- blocks already allocated within the file system

A check is made to see that all the blocks in the file system were found.

The first free-block list is in the super-block. The `fsck` program checks the list count for a value less than 0 or greater than 50. It also checks each block number to make sure it is within the range bounded by the first and last data block in the file system. Each block number is compared to a list of previously allocated blocks. If the free-list block pointer is not 0, the next free-list block is read and the process is repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks in the free-block list plus the number of blocks claimed by the inodes equals the total number of blocks in the file system. If anything is wrong with the free-block list, `fsck` can rebuild it leaving out blocks already allocated.

Free-Block Count

The super-block contains a count of the total number of free blocks within the file system. The `fsck` program compares this count to the number of blocks it finds free within the file system. If the counts do not agree, `fsck` can replace the count in the super-block by the actual free-block count.

Free Inode Count

The super-block contains a count of the number of free inodes within the file system. The `fsck` program compares this count to the number of inodes it found free within the file system. If the counts do not agree, `fsck` can replace the count in

the super-block by the actual free-inode count.

Inodes

The list of inodes is checked sequentially starting with inode 1 (there is no inode 0). Each inode is checked for inconsistencies involving:

- format and type
- link count
- duplicate blocks
- bad block numbers
- inode size

Format and Type

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes may be one of six types:

- regular
- directory
- block special
- character special
- FIFO (named-pipe)
- symbolic link

Inodes may be in one of three states: unallocated, allocated, or partially allocated. This last state means that the inode is incorrectly formatted. An inode can get into this state if, for example, bad data is written into the inode list because of a hardware failure. The only corrective action `fsck` can take is to clear the inode.

Link Count

Each inode contains a count of the number of directory entries linked to it. The `fsck` program verifies the link count of each inode by examining the entire directory structure, starting from the root directory, and calculating an actual link count for each inode.

Discrepancies between the link count stored in the inode and the actual link count as determined by `fsck` may be of three types:

- The stored count is not 0, the actual count is 0.

  This can occur if no directory entry appears for the inode. In this case `fsck` can link the disconnected file to the `lost+found` directory.

- The stored count is not 0, the actual count is not 0, but the counts are unequal.

  This can occur if a directory entry has been removed but the inode has not been updated. In this case `fsck` can replace the stored link count by the actual link count.

- The stored count is 0, the actual count is not 0.

  In this case `fsck` can change the link count of inode to the actual count.

Duplicate Blocks

Each inode contains a list of all the blocks claimed by the inode. The `fsck` program compares each block number claimed by an inode to a list of allocated blocks. If a block number claimed by an inode is on the list of allocated blocks, it is put on a list of duplicate blocks. If it is not on the list of allocated blocks, it is put on it. If this process produces a list of duplicate blocks, `fsck` makes a second pass of the inode list to find the other inode that claims each duplicate block.

NOTE

A large number of duplicate blocks in an inode may be caused by an indirect block not being written to the file system.

Although it is not possible to determine with certainty which inode is in error, in most cases the inode with the most recent modification time is correct. The `fsck` program prompts the user to clear both inodes.

Bad Block Numbers

The `fsck` program checks each block number claimed by an inode to see that its value is higher than that of the first data block and lower than that of the last block in the file system. If the block number is outside this range, it is considered a bad block number.

Bad block numbers in an inode may be caused by an indirect block not being written to the file system. The `fsck` program prompts the user to clear the inode.

NOTE: A bad block number in a file system is not the same as a bad (that is, unreadable) block on a hard disk.

Inode Size

Each inode contains a 32-bit (4-byte) size field. This field shows the number of characters in the file associated with the inode. A directory inode within the file system has the directory bit set in the inode mode word.

If the directory size is not a multiple of 16, `fsck` warns of directory misalignment and prompts for corrective action.

For a regular file, you can perform a rough check of the consistency of the size field of an inode by using the number of characters shown in the size field to calculate how many blocks should be associated with the inode, and comparing that to the actual number of blocks claimed by the inode.

Indirect Blocks

Indirect blocks are owned by an inode. Therefore, inconsistencies in an indirect block directly affect the inode that owns it. Inconsistencies that can be checked are:

- blocks already claimed by another inode
- block numbers outside the range of the file system

The consistency checks for direct blocks described in the sections "Duplicate Blocks" and "Bad Block Numbers" above are also performed for indirect blocks.

Directory Data Blocks

Directories are distinguished from regular files by an entry in the mode field of the inode. Data blocks associated with a directory contain the directory entries. Directory data blocks are checked for inconsistencies involving:

- directory inode numbers pointing to unallocated inodes
- directory inode numbers greater than the number of inodes in the file system
- incorrect directory inode numbers for "." and ". ." directories
- directories disconnected from the file system

Directory Unallocated

If a directory entry inode number points to an unallocated inode, `fsck` can remove that directory entry. This condition occurs if the data blocks containing the directory entries are modified and written out while the inode is not yet written out.

Bad Inode Number

If a directory entry inode number is pointing beyond the end of the inode list, `fsck` can remove that directory entry. This condition occurs if bad data is written into a directory data block.

Incorrect "." and ".." Entries

The directory inode number entry for "." should be the first entry in the directory data block. Its value should be equal to the inode number for the directory data block.

The directory inode number entry for ". ." should be the second entry in the directory data block. Its value should be equal to the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory). If the directory inode numbers for "." and ". ." are incorrect, `fsck` can replace them with the correct values.

Disconnected Directories

The `fsck` program checks the general connectivity of the file system. If a directory is found that is not linked to the file system, `fsck` links the directory to the `lost+found` directory of the file system. (This condition can occur when inodes are written to the file system but the corresponding directory data blocks are not.) When a file is linked to the `lost+found` directory, the owner of the file must be notified.

Regular Data Blocks

Data blocks associated with a regular file hold the file's contents. `fsck` does not attempt to check the validity of the contents of a regular file's data blocks.

Running `fsck` on an `s5` File System

The `fsck` program runs in phases. Each phase reports any errors that it detects. Figure 6-12 lists the abbreviations that are used in the `fsck` error messages. If the error is one that `fsck` can correct, the user is asked if the correction should be made. This section describes the messages that are produced by each phase.

Figure 6-11: Error Message Abbreviations in `fsck` Output

The following abbreviations that appear in the messages have the meaning indicated by the text following them:

`BLK`	block number
`DUP`	duplicate block number
`DIR`	directory name
`MTIME`	time file was last modified
`UNREF`	unreferenced
`CG`	cylinder group

The following single-letter abbreviations are replaced by the text opposite them when the message appears on your screen:

`B`	block number
`F`	file (or directory) name
`I`	inode number
`M`	file mode
`O`	user-ID of a file's owner
`S`	file size
`T`	time file was last modified
`X`	replaced by one of the following, depending on the context: link count, number of BAD, DUP, or MISSING blocks number of files
`Y`	replaced by one of the following, depending on the context: corrected link count number, number of blocks in file system
`Z`	number of free blocks

Initialization Phase

Command line syntax is checked. Before the file system check can be performed, `fsck` sets up some tables and opens some files. The `fsck` program terminates when it encounters errors during the initialization phase.

General Errors

The following three error messages may appear in any phase after initialization. While they offer the option to continue, it is generally best to regard them as fatal, end the run, and try to determine what caused the problem.

Message

```
CAN NOT SEEK: BLK B (CONTINUE?)
```

A request to move to a specified block number *B* in the file system failed. This message indicates a serious problem, probably a hardware failure.

Message

```
CAN NOT READ: BLK B (CONTINUE?)
```

A request to read a specified block number *B* in the file system failed. The message indicates a serious problem, probably a hardware failure.

Message

```
CAN NOT WRITE: BLK B (CONTINUE?)
```

A request to write a specified block number *B* in the file system failed. The disk may be write-protected.

Meaning of Yes/No Responses

An n (no) response to the CONTINUE? prompt means:

Terminate the program.
(This is the recommended response.)

A y (yes) response to the CONTINUE? prompt means:

Attempt to continue to run the file system check.
Note that the problem will often recur. Any of the three error conditions described above prevents a complete check of the file system. You should run fsck again to recheck the file system.

Phase 1: Check Blocks and Sizes

This phase checks the inode list. It reports error conditions encountered while:

- checking inode types
- setting up the zero-link-count table
- examining inode block numbers for bad or duplicate blocks
- checking inode size
- checking inode format

Types of Error Messages - Phase 1

Phase 1 produces the following types of error messages:

- informational messages
- messages with a CONTINUE? prompt
- messages with a CLEAR? prompt
- messages with a RECOVER? prompt

There is a connection between some informational messages and messages with a CONTINUE? prompt. The CONTINUE? prompt generally indicates that some limit has been reached.

Meaning of Yes/No Responses - Phase 1

An n (no) response to the CONTINUE? prompt means:

Terminate the program.

In Phase 1, a y (yes) response to the CONTINUE? prompt means:

Continue with the program.
When this error occurs a complete check of the file system is not possible. You should run `fsck` again to recheck the file system.

An n (no) response to the RECOVER? prompt means:

Recover all the blocks to which the inode points.
A no response is only appropriate if the user intends to delete the excess blocks.

An n (no) response to the CLEAR? prompt means:

Ignore the error condition.
A no response is only appropriate if the user intends to take other measures to fix the problem.

A y (yes) response to the CLEAR? prompt means:

Deallocate the inode *I* by zeroing out its contents.
This may generate the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this inode.

Phase 1 Error Messages

Message

```
UNKNOWN FILE TYPE I= I (CLEAR?)
```

The mode word of the inode *I* indicates that the inode is not a pipe, special character inode, regular inode, or directory inode. If the -p option is specified the inode will be cleared.

Message

```
LINK COUNT TABLE OVERFLOW (CONTINUE?)
```

An internal table for `fsck` containing allocated inodes with a link count of zero has no more room. If the `-p` option is specified the program will exit and `fsck` will have to be completed manually.

Message

```
B BAD I= I
```

Inode *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may also generate the `EXCESSIVE BAD BLKS` error message if inode *I* has too many block numbers outside the file system range. This error condition generates the `BAD/DUP` error message in Phases 2 and 4.

Message

```
EXCESSIVE BAD BLOCKS I= I (CONTINUE?)
```

There are too many (usually more than ten) blocks with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system associated with inode *I*. If the `-p` option is specified, the program will terminate.

Message

```
B DUP I= I
```

Inode *I* contains block number *B*, which is already claimed by the same or another inode or by a free-list. This error condition may also generate the EXCESSIVE DUP BLKS error message if inode *I* has too many block numbers claimed by the same or another inode or by a free-list. This error condition invokes Phase 1B and generates the BAD/DUP error message in Phases 2 and 4.

Message

```
EXCESSIVE DUP BLKS I= I (CONTINUE?)
```

There are too many (usually more than ten) blocks claimed by the same or another inode or by a free-list. If the -p option is specified, the program will terminate.

Message

```
DUP TABLE OVERFLOW (CONTINUE?)
```

An internal table in `fsck` containing duplicate block numbers has no more room. If the -p option is specified, the program will terminate.

Message

```
DIRECTORY MISALIGNED I= I
```

The size of a directory inode is not a multiple of 16. If the -p option is used, the directory will be recovered automatically.

Message

```
PARTIALLY ALLOCATED INODE I= I (CLEAR?)
```

I-node *I* is neither allocated nor unallocated. If the -p option is specified, the inode will be cleared.

Message

```
DIR/FILE SIZE ERROR
```

The file references more or less data than is indicated by the inode.

Message

```
DELETE OR RECOVER EXCESS DATA
```

The user has the choice of deleting or recovering the excess blocks pointed to by the inode.

Message

```
RECOVER?
```

The file references more data than is indicated by the inode. The user is given the choice of correcting the inode information. If the -p option is specified, the data will be recovered.

Message

```
DELETE?
```

The file references more data than is indicated by the inode. The user is given the choice of deleting the referenced blocks and leaving the inode data intact.

Phase 1B: Rescan for More DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the inode that previously claimed that block. When the duplicate block is found, the following informational message is printed:

Message

```
DUP I= I
```

Inode *I* contains block number *B* that is already claimed by the same or another inode or by a free-list. This error condition generates the BAD/DUP error message in Phase 2. Inodes that have overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1.

Phase 2: Check Pathnames

This phase removes directory entries pointing to bad inodes found in Phases 1 and 1B. It reports error conditions resulting from the following:

- incorrect root inode mode and status
- directory inode pointers out of range
- directory entries pointing to bad inodes

Types of Error Messages—Phase 2

Phase 2 has the following types of error messages:

- informational messages
- messages with a `FIX?` prompt
- messages with a `CONTINUE?` prompt
- messages with a `REMOVE?` prompt

Meaning of Yes/No Responses—Phase 2

An `n` (no) response to the `FIX?` prompt means:

Terminate the program because `fsck` will be unable to continue.

A `y` (yes) response to the `FIX?` prompt means:

Change the root inode type to "directory."
If the root inode data blocks are not directory blocks, a very large number of error messages are generated.

An `n` (no) response to the `CONTINUE?` prompt means:

Terminate the program.

A `y` (yes) response to the `CONTINUE?` prompt means:

Ignore the DUPS/BAD IN ROOT INODE error message and continue to run the file system check.
If the root inode is not correct, a large number of other error messages may be generated.

An `n` (no) response to the `REMOVE?` prompt means:

Ignore the error condition.
A no response is only appropriate if the user intends to take other measures to fix the problem.

A `y` (yes) response to the `REMOVE?` prompt means:

Remove duplicate or unallocated blocks.

Phase 2 Error Messages

Message

```
ROOT INODE UNALLOCATED. TERMINATING
```

The root inode (usually inode number 2) of the file system has no allocate mode bits. This error message indicates a serious problem that causes the program to stop. Call your service representative.

Message

```
ROOT INODE NOT DIRECTORY (FIX?)
```

The root inode (usually inode number 2) of the file system is not directory inode type. If the -p option is specified, the program will terminate.

Message

```
DUPS/BAD IN ROOT INODE (CONTINUE?)
```

Phase 1 or 1B found duplicate blocks or bad blocks in the root inode (usually inode number 2) of the file system. If the -p option is specified, the program will terminate.

Message

```
I OUT OF RANGE I= I NAME= F (REMOVE?)
```

A directory entry *F* has an inode number *I* that is greater than the end of the inode list. If the -p option is specified, the inode will be removed automatically.

Message

```
UNALLOCATED I= I OWNER= O MODE= M SIZE= S MTIME= T NAME= F (REMOVE?)
```

A directory entry *F* has an inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and filename *F* are printed. If the file system is not mounted and the -n option was not specified, the entry is removed automatically if the inode it points to is character size 0. The entry is removed if the -p option is specified.

Message

```
DUP/BAD I= I OWNER= O MODE= M SIZE= S MTIME= T DIR= F (REMOVE?)
```

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory entry *F*, directory inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed. If the -p option is specified, the duplicate/bad blocks are removed.

Message

```
DUP/BAD I= I OWNER= O MODE= M SIZE= S MTIME= T FILE= F (REMOVE?)
```

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and filename *F* are printed. If the -p option is specified, the duplicate/bad blocks are removed.

```
BAD BLK B IN DIR I= I OWNER= O MODE= M SIZE= S MTIME= T
```

This message only occurs when the -D option is used. A physically damaged block was found in directory inode *I*. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ". ." entries, and embedded slashes in the name field. This error message means that the user should at a later time either remove the directory inode if the entire block looks bad, or change (or remove) those directory entries that look bad.

Phase 3: Check Connectivity

This phase checks the directories examined in Phase 2. It reports error conditions resulting from

- unreferenced directories
- missing or full `lost+found` directories

Types of Error Messages—Phase 3

Phase 3 has the following types of error messages:

- informational messages
- messages with a `RECONNECT?` prompt

Meaning of Yes/No Responses—Phase 3

An `n` (no) response to the `RECONNECT?` prompt means:

> Ignore the error condition.
> This response generates `UNREF` error messages in Phase 4.
> A no response is only appropriate if the user intends to take other measures to fix the problem.

A `y` (yes) response to the `RECONNECT?` prompt means:

> Reconnect directory inode *I* to the file system in the directory for lost files (usually the `lost+found` directory).
> This may generate `lost+found` error messages if there are problems connecting directory inode *I* to the `lost+found` directory. If the link is

successful, a CONNECTED informational message appears.

Phase 3 Error Messages

Message

```
UNREF DIR I= I OWNER= O MODE= M SIZE= S MTIME= T (RECONNECT?)
```

The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. The `fsck` program forces the reconnection of a nonempty directory.

Message

```
SORRY. NO lost+found DIRECTORY
```

There is no `lost+found` directory in the root directory of the file system; `fsck` ignores the request to link a directory to the `lost+found` directory. This generates the UNREF error message in Phase 4. The access modes of the `lost+found` directory may be incorrect.

Message

```
SORRY. NO SPACE IN lost+found DIRECTORY
```

There is no space to add another entry to the `lost+found` directory in the root directory of the file system; `fsck` ignores the request to link a directory to the `lost+found` directory. This generates the UNREF error message in Phase 4. Clear out unnecessary entries in the `lost+found` directory or make it larger.

Message

```
DIR I= I1 CONNECTED. PARENT WAS I= I2
```

This is an advisory message indicating a directory inode *I1* was successfully connected to the `lost+found` directory. The parent inode *I2* of the directory inode *I1* is replaced by the inode number of the `lost+found` directory.

Phase 4: Check Reference Counts

This phase checks the link count information obtained in Phases 2 and 3. It reports error conditions resulting from:

- unreferenced files
- a missing or full `lost+found` directory
- incorrect link counts for files, directories, or special files
- unreferenced files and directories
- bad or duplicate blocks in files and directories
- incorrect total free-inode counts

Types of Error Messages—Phase 4

Phase 4 has the following types of error messages:

- informational messages
- messages with a `RECONNECT?` prompt
- messages with a `CLEAR?` prompt
- messages with an `ADJUST?` prompt
- messages with a `FIX?` prompt

Meaning of Yes/No Responses—Phase 4

An n (no) response to the RECONNECT? prompt means:

Ignore this error condition.
This response generates a CLEAR error message later in the phase.

A y (yes) response to the RECONNECT? prompt means:

Reconnect inode *I* to the file system in the directory for lost files (usually the `lost+found` directory).
This can generate a `lost+found` error message in this phase if there are problems connecting inode *I* to the `lost+found` directory.

An n (no) response to the CLEAR? prompt means:

Ignore the error condition.
This response is only appropriate if the user intends to take other measures to fix the problem.

A y (yes) response to the CLEAR? prompt means:

Deallocate the inode by zeroing out its contents.

An n (no) response to the ADJUST? prompt means:
Ignore the error condition.
This response is only appropriate if the user intends to take other measures to fix the problem.

A y (yes) response to the ADJUST? prompt means:
Replace the link count of file inode *I* with *Y*.

An n (no) response to the FIX? prompt means:

Ignore the error condition.
This response is only appropriate if the user intends to take other measures to fix the problem.

A y (yes) response to the FIX? prompt means:

Replace the count in super-block by the actual count.

Phase 4 Error Messages

Message

```
UNREF FILE I= I OWNER= O MODE= M SIZE= S MTIME= T (RECONNECT?)
```

Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the `-n` option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty files are not cleared. If the `-p` option is specified, the inode is reconnected.

Message

```
SORRY. NO lost+found DIRECTORY
```

There is no `lost+found` directory in the root directory of the file system; `fsck` ignores the request to link a file to the `lost+found` directory. This generates the `CLEAR` error message later in the phase. The access modes of the `lost+found` directory may be incorrect.

Message

```
SORRY. NO SPACE IN lost+found DIRECTORY
```

There is no space to add another entry to the `lost+found` directory in the root directory of the file system; `fsck` ignores the request to link a file to the `lost+found` directory. This generates the `CLEAR` error message later in the phase. Check the size and contents of the `lost+found` directory.

Message

```
(CLEAR)
```

The inode mentioned in the UNREF error message immediately preceding cannot be reconnected.

Message

```
LINK COUNT FILE I= I OWNER= O MODE= M SIZE= S MTIME= T COUNT= X SHOULD BE Y (ADJUST?)
```

The link count for file inode *I* is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed. If the -p option is specified, the link count is adjusted.

Message

```
LINK COUNT DIR I= I OWNER= O MODE= M SIZE= S MTIME= T COUNT= X SHOULD BE Y (ADJUST?)
```

The link count for directory inode *I* is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the -p option is specified, the link count is adjusted.

Message

```
UNREF FILE I= I OWNER= O MODE= M SIZE= S MTIME= T (CLEAR?)
```

File inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the -n option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty directories are not cleared. If the -p option is specified, the file is cleared if it can not be reconnected.

Message

```
UNREF DIR I= I OWNER= O MODE= M SIZE= S MTIME= T (CLEAR?)
```

Directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the -n option is omitted and the file system is not mounted, empty directories are cleared automatically. Nonempty directories are not cleared. If the -p option is specified, the directory is cleared if it can not be reconnected.

Message

```
BAD/DUP FILE I= I OWNER= O MODE= M SIZE= S MTIME= T (CLEAR?)
```

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the -p option is specified, the file is cleared.

Message

```
BAD/DUP DIR I= I OWNER= O MODE= M SIZE= S MTIME= T (CLEAR?)
```

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the -p option is specified, the directory is cleared.

Message

```
FREE INODE COUNT WRONG IN SUPERBLK (FIX?)
```

The actual count of the free inodes does not match the count in the super-block of the file system. If the `-q` or `-p` option is specified, the count in the super-block will be fixed automatically.

Phase 5: Check Free List

This phase checks the free-block list. It reports error conditions resulting from:

- bad blocks in the free-block list
- a bad free-block count
- duplicate blocks in the free-block list
- unused blocks from the file system that are not in the free-block list
- an incorrect total free-block count

Types of Error Messages - Phase 5

Phase 5 has the following types of error messages:

- informational messages
- messages that have a `CONTINUE?` prompt
- messages that have a `FIX?` prompt
- messages that have a `SALVAGE?` prompt

Meaning of Yes/No Responses - Phase 5

An `n` (no) response to the `CONTINUE?` prompt means:

Terminate the program.

A y (yes) response to the CONTINUE? prompt means:

Ignore the rest of the free-block list and continue the execution of fsck. This generates the BAD BLKS IN FREE LIST error message later in this phase.

An n (no) response to the FIX? prompt means:

Ignore the error condition.
This response is only appropriate if the user intends to take other measures to fix the problem.

A y (yes) response to the FIX? prompt means:

Replace the count in the super-block by the actual count.

An n (no) response to the SALVAGE? prompt means:

Ignore the error condition.
This response is only appropriate if the user intends to take other measures to fix the problem.

A y (yes) response to the SALVAGE? prompt means:

Replace the actual free-block list by a new free-block list.
The new free-block list will be ordered according to the gap and cylinder specifications of the -s or -S option to reduce the time spent waiting for the disk to rotate into position.

Phase 5 Error Messages

Message

```
EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE?)
```

The free-block list contains too many blocks with a value less than the first data block in the file system or greater than the last block in the file system. If the -p option is specified the program terminates.

Message

```
EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE?)
```

The free-block list contains too many blocks claimed by inodes or earlier parts of the free block list. If the `-p` option is specified the program terminates.

Message

```
BAD FREEBLK COUNT
```

The count of free blocks in a free-list block is greater than 50 or less than 0. This condition generates the `BAD FREE LIST` message later in the phase.

Message

```
X BAD BLKS IN FREE LIST
```

X blocks in the free-block list have a block number lower than the first data block in the file system or greater than the last block in the file system. This condition generates the `BAD FREE LIST` message later in the phase.

Message

```
X DUP BLKS IN FREE LIST
```

X blocks claimed by inodes or earlier parts of the free-list block were found in the free-block list. This condition generates the `BAD FREE LIST` message later in the phase.

Message

```
X BLK(S) MISSING
```

X blocks unused by the file system were not found in the free-block list. This condition generates the `BAD FREE LIST` message later in the phase.

Message

```
FREE BLK COUNT WRONG IN SUPERBLOCK (FIX?)
```

The actual count of free blocks does not match the count of free blocks in the super-block of the file system. If the `-p` option was specified, the free block count in the super-block is fixed automatically.

Message

```
BAD FREE LIST (SALVAGE?)
```

This message is always preceded by one or more of the Phase 5 informational messages. If the `-q` or `-p` option was specified, the free-block list will be salvaged automatically.

Phase 6: Salvage Free List

This phase reconstructs the free-block list. It may display an advisory message about the blocks-to-skip or blocks-per-cylinder values.

Phase 6 Error Messages

Message

```
DEFAULT FREE-BLOCK LIST SPACING ASSUMED
```

This is an advisory message indicating that the blocks-to-skip (gap) is greater than the blocks-per-cylinder, the blocks-to-skip is less than 1, the blocks-per-cylinder is less than 1, or the blocks-per-cylinder is greater than 1000. The default values of 1 blocks-to-skip and 324 blocks-per-cylinder are used.

NOTE

Because the default values used may not be accurate for your system, be careful to specify correct values with the `-s` option on the command line. See the `fsck`(1M) and `mkfs`(1M) manual pages for further details.

Cleanup Phase

Once a file system has been checked, a few cleanup functions are performed. The cleanup phase displays advisory messages about the file system and the status of the file system.

Cleanup Phase Messages

Message

```
X files Y blocks Z free
```

This is an advisory message indicating that the file system checked contained *X* files using *Y* blocks, and that there are *Z* blocks free in the file system.

Message

```
***** FILE SYSTEM WAS MODIFIED *****
```

This is an advisory message indicating that the file system was modified by `fsck`.

Checking `ufs` File Systems

This section describes the use of `fsck` with `ufs` file systems. It assumes that you are running `fsck` interactively, and that all possible errors can be encountered. When an inconsistency is discovered in this mode, `fsck` reports the inconsistency for you to choose a corrective action. (You can also run `fsck` automatically using the `-y`, `-n`, or `-o p` options.)

`ufs` File System Components Checked by `fsck`

Before describing the phases of `fsck` and the messages that may appear in each, we will discuss the kinds of consistency checks applied to each component of a `ufs` file system.

Super-Block

The most commonly corrupted item in a file system is the summary information associated with the super-block. This is because the super-block is modified with every change to the blocks or inodes of the file system, and is usually corrupted after an unclean halt. The super-block is checked for inconsistencies involving:

- file system size
- number of inodes
- free block count
- free inode count

File System Size

The file system size must be larger than the number of blocks used by the super-block plus the number of blocks used by the list of inodes. While there is no way to check these sizes precisely, `fsck` can check that they are within reasonable bounds. All other file system checks require that these sizes be correct. If `fsck` detects corruption in the static parameters of the default super-block, `fsck` requests the operator to specify the location of an alternate super-block.

Free Block List

`fsck` checks that all the blocks marked as free in the cylinder group block maps are not claimed by any files. When all the blocks have been initially accounted for, `fsck` checks that the number of free blocks plus the number of blocks claimed by the inodes equals the total number of blocks in the file system. If anything is wrong with the block allocation maps, `fsck` will rebuild them based on the list it has computed of allocated blocks.

Free Block Count

The summary information associated with the super-block contains a count of the total number of free blocks within the file system. `fsck` compares this count to the number of free blocks it finds within the file system. If the two counts do not agree, then `fsck` replaces the incorrect count in the summary information by the actual free block count.

Free Inode Count

The summary information contains a count of the total number of free inodes within the file system. `fsck` compares this count to the number of free inodes it finds within the file system. If the two counts do not agree, then `fsck` replaces the incorrect count in the summary information by the actual free inode count.

Inodes

The list of inodes in the file system is checked sequentially, starting with inode 2 (inode 0 marks unused inodes; inode 1 is saved for future generations) and progressing through the last inode in the file system. Each inode is checked for inconsistencies involving:

- format and type

- link count
- duplicate blocks
- bad block numbers
- inode size

Format and Type

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes may be one of six types: regular, directory, symbolic link, special block, special character, or named-pipe. Inodes may be in one of three allocation states: unallocated, allocated, or neither unallocated nor allocated. This last state means that the inode is incorrectly formatted. An inode can get into this state if bad data is written into the inode list. The only possible corrective action `fsck` can take is to clear the inode.

Link Count

Each inode counts the total number of directory entries linked to the inode. `fsck` verifies the link count of each inode by starting at the root of the file system, and descending through the directory structure. The actual link count for each inode is calculated during the descent.

If the stored link count is non-zero and the actual link count is zero, then no directory entry appears for the inode. If this happens, `fsck` will place the disconnected file in the `lost+found` directory. If the stored and actual link counts are non-zero and unequal, a directory entry may have been added or removed without the inode being updated. If this happens, `fsck` replaces the incorrect stored link count by the actual link count.

Each inode contains a list, or pointers to lists (indirect blocks), of all the blocks claimed by the inode. Because indirect blocks are owned by an inode, inconsistencies in an indirect block directly affect the inode that owns it.

Duplicate Blocks

`fsck` compares each block number claimed by an inode against a list of already allocated blocks. If another inode already claims a block number, then the block number is added to a list of duplicate blocks. Otherwise, the list of allocated blocks is updated to include the block number.

If there are any duplicate blocks, `fsck` performs a partial second pass over the inode list to find the inode of the duplicated block. The second pass is needed because without examining the files associated with these inodes for correct content, not enough information is available to determine which inode is corrupted and should be cleared. If this condition does arise, then the inode with the earliest modify time is usually incorrect, and should be cleared. If this happens, `fsck` prompts the operator to clear both inodes. The operator must decide which one should be kept and which one should be cleared.

Bad Block Numbers

`fsck` checks the range of each block number claimed by an inode. If the block number is lower than the first data block in the file system, or greater than the last data block, then the block number is a bad block number. Many bad blocks in an inode are usually caused by an indirect block that was not written to the file system, a condition which can only occur if there has been a hardware failure. If an inode contains bad block numbers, `fsck` prompts the operator to clear it.

Inode Size

Each inode contains a count of the number of data blocks that it contains. The number of actual data blocks is the sum of the allocated data blocks and the indirect blocks. `fsck` computes the actual number of data blocks and compares that block count against the actual number of blocks the inode claims. If an inode contains an incorrect count `fsck` prompts the operator to fix it.

Each inode contains a thirty-two bit size field. The size is the number of data bytes in the file associated with the inode. The consistency of the byte size field is roughly checked by computing from the size field the maximum number of blocks that should be associated with the inode, and comparing that expected block count against the actual number of blocks the inode claims.

Data Associated with an Inode

An inode can directly or indirectly reference three kinds of data blocks. All referenced blocks must be the same kind. The three types of data blocks are: plain data blocks, symbolic link data blocks, and directory data blocks. Plain data blocks contain the information stored in a file; symbolic link data blocks contain the path name stored in a link. Directory data blocks contain directory entries. `fsck` can only check the validity of directory data blocks.

Directory Data Blocks

Directory data blocks are checked for inconsistencies involving:

- directory inode numbers pointing to unallocated inodes
- directory inode numbers that are greater than the number of inodes in the file system
- incorrect directory inode numbers for "." and ". ."
- directories that are not attached to the file system

Directory Unallocated

If the inode number in a directory data block references an unallocated inode, then `fsck` will remove that directory entry.

Bad Inode Number

If a directory entry inode number references outside the inode list, then `fsck` will remove that directory entry. This condition occurs if bad data are written into a directory data block.

Incorrect "." and ".." Entries

The directory inode number entry for "." must be the first entry in the directory data block. The inode number for "." must reference itself; that is, it must equal the inode number for the directory data block. The directory inode number entry for ". ." must be the second entry in the directory data block. Its value must equal the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory). If the directory inode numbers are incorrect, `fsck` will replace them with the correct values. If there are multiple hard links to a directory, the first one encountered is considered the real parent to which ". ." should point; `fsck` recommends deletion for the subsequently discovered names.

Disconnected Directories

`fsck` checks the general connectivity of the file system. If directories are not linked into the file system, then `fsck` links the directory back into the file system in the `lost+found` directory.

Running `fsck` on a `ufs` File System

`fsck` is a multi-pass file system check program. Each file system pass invokes a different phase of the `fsck` program. After initialization, `fsck` performs successive passes over each file system, checking blocks and sizes, path names, connectivity, reference counts, and the map of free blocks (possibly rebuilding it), and performs some cleanup.

At boot time `fsck` is normally run with the `-y` option, non-interactively. (`fsck` can also be run interactively by the administrator at any time.) `fsck` can also be run non-interactively to "preen" the file systems after an unclean halt. While preening a file system, it will only fix corruptions that are expected to result from an unclean halt. These actions are a subset of the actions that `fsck` takes when it is running interactively. When an inconsistency is detected, `fsck` generates an error message. If a response is required, `fsck` prints a prompt and waits for a response. Most errors encountered during preening are fatal. Note the response taken for those that are expected. This section explains the meaning of each error message, the possible responses, and the related error conditions.

The error conditions are organized by the phase of the `fsck` program in which they can occur. The error conditions that may occur in more than one phase are discussed under initialization.

Initialization Phase

Before a file system check can be performed, certain tables have to be set up and certain files opened. The messages in this section relate to error conditions resulting from command line options, memory requests, the opening of files, the status of files, file system size checks, and the creation of the scratch file.

Message

```
cannot alloc NNN bytes for blockmap
cannot alloc NNN bytes for freemap
cannot alloc NNN bytes for statemap
cannot alloc NNN bytes for lncntp
```

`fsck`'s request for memory for its virtual memory tables failed. This should never happen. When it does, `fsck` terminates. This is a serious system failure and should be handled immediately. Contact your service representative or another qualified person.

Message

```
Can't open checklist file: F
```

The file system checklist or default file *F* (usually `/etc/vfstab`) cannot be opened for reading. When this occurs, `fsck` terminates. Check the access modes of *F*.

Message

```
Can't stat root
```

`fsck`'s request for statistics about the `root` directory failed. This should never happen. When it does, `fsck` terminates. Contact your service representative or another qualified person.

Message

```
Can't stat F
Can't make sense out of name F
```

`fsck`'s request for statistics about the file system *F* failed. When running interactively, it ignores this file system and continues checking the next file system given. Check the access modes of *F*.

Message

```
Can't open F
```

`fsck`'s attempt to open the file system *F* failed. When running interactively, it ignores this file system and continues checking the next file system given. Check the access modes of *F*.

Message

```
F: (NO WRITE)
```

Either the `-n` flag was specified or `fsck`'s attempt to open the file system *F* for writing failed. When `fsck` is running interactively, all the diagnostics are printed out, but `fsck` does not attempt to fix anything.

Message

```
file is not a block or character device; OK
```

The user has given `fsck` the name of a regular file by mistake. Check the type of the file specified.

Possible responses to the `OK` prompt are:

y (yes)	Ignore this error condition.
n (no)	Ignore this file system and continue checking the next file system given.

Message

```
UNDEFINED OPTIMIZATION IN SUPERBLOCK (SET TO DEFAULT)
```

The super-block optimization parameter is neither `OPT_TIME` nor `OPT_SPACE`.

Possible responses to the `SET TO DEFAULT` prompt are:

`y` (yes) Set the super-block to request optimization to minimize running time of the system. (If optimization to minimize disk space use is desired, it can be set using `tunefs` (1M).)

`n` (no) Ignore this error condition.

Message

```
IMPOSSIBLE MINFREE=D IN SUPERBLOCK (SET TO DEFAULT)
```

The super-block minimum space percentage is greater than 99 percent or less than 0 percent.

Possible responses to the `SET TO DEFAULT` prompt are:

`y` (yes) Set the `minfree` parameter to 10 percent. (If some other percentage is desired, it can be set using `tunefs` (1M).)

`n` (no) Ignore this error condition.

Message

```
MAGIC NUMBER WRONG
NCG OUT OF RANGE
CPG OUT OF RANGE
NCYL DOES NOT JIVE WITH NCG*CPG
SIZE PREPOSTEROUSLY LARGE
```

(continued on next page)

```
TRASHED VALUES IN SUPER BLOCK
```

followed by the message:

```
F: BAD SUPER BLOCK: B
USE -b OPTION TO FSCK TO SPECIFY LOCATION OF AN ALTERNATE
SUPER-BLOCK TO SUPPLY NEEDED INFORMATION; SEE fsck(1M).
```

The super-block has been corrupted. An alternative super-block must be selected from among the available copies. Choose an alternative super-block by calculating its offset or call your service representative or another qualified person. Specifying block 32 is a good first choice.

Message

```
INTERNAL INCONSISTENCY: M
```

`fsck` has had an internal panic, whose message is *M*. This should never happen. If it does, contact your service representative or another qualified person.

Message

```
CAN NOT SEEK: BLK B (CONTINUE)
```

`fsck`'s request to move to a specified block number *B* in the file system failed. This should never happen. If it does, contact your service representative or another qualified person.

Possible responses to the `CONTINUE` prompt are:

y (yes) Attempt to continue to run the file system check. (Note that the problem will often persist.) This error condition prevents a complete check of the file system. A second run of `fsck` should be made to recheck the file system. If the block was part of the virtual memory buffer cache, `fsck` will terminate with the message:

```
Fatal I/O error
```

n (no) Terminate the program.

Message

```
CAN NOT READ: BLK B (CONTINUE)
```

`fsck`'s request to read a specified block number *B* in the file system failed. This should never happen. If it does, contact your service representative or another qualified person.

Possible responses to the `CONTINUE` prompt are:

y (yes) Attempt to continue to run the file system check. `fsck` will retry the read and print out the message:

```
THE FOLLOWING SECTORS COULD NOT BE READ: N
```

where *N* indicates the sectors that could not be read. If `fsck` ever tries to write back one of the blocks on which the read failed it will print the message:

```
WRITING ZERO'ED BLOCK N TO DISK
```

where *N* indicates the sector that was written with zero's. If the disk is experiencing hardware problems, the problem will persist. This error condition prevents a complete check of the file system. A second run of `fsck` should be made to recheck the file system. If the block was part of the virtual memory buffer cache, `fsck` will terminate with the message:

```
Fatal I/O error
```

n (no) Terminate the program.

Message

```
CAN NOT WRITE: BLK B (CONTINUE)
```

`fsck`'s request to write a specified block number *B* in the file system failed. The disk is write-protected; check the write-protect lock on the drive. If that is not the problem, contact your service representative or another qualified person.

Possible responses to the CONTINUE prompt are:

y (yes) Attempt to continue to run the file system check. The write operation will be retried. Sectors that could not be written will be indicated by the message:

```
THE FOLLOWING SECTORS COULD NOT BE WRITTEN: N
```

where *N* indicates the sectors that could not be written. If the disk is experiencing hardware problems, the problem will persist. This error condition prevents a complete check of the file system. A second run of `fsck` should be made to recheck this file system. If the block was part of the virtual memory buffer cache, `fsck` will terminate with the message:

```
Fatal I/O error
```

n (no) Terminate the program.

Message

```
bad inode number DDD to ginode
```

An internal error was caused by an attempt to read non-existent inode *DDD*. This error causes `fsck` to exit. If this occurs, contact your service representative or another qualified person.

Phase 1: Check Blocks and Sizes

This phase checks the inode list. It reports error conditions encountered while:

- checking inode types
- setting up the zero-link-count table
- examining inode block numbers for bad or duplicate blocks
- checking inode size
- checking inode format

All the errors in this phase except `INCORRECT BLOCK COUNT` and `PARTIALLY TRUNCATED INODE` are fatal if the file system is being preened.

Phase 1 Error Messages

Message

```
UNKNOWN FILE TYPE I=I (CLEAR)
```

The mode word of the inode *I* indicates that the inode is not a special block inode, special character inode, socket inode, regular inode, symbolic link, FIFO file, or directory inode.

Possible responses to the `CLEAR` prompt are:

y (yes)	De-allocate inode *I* by zeroing out its contents. This will always generate the `UNALLOCATED` error message in Phase 2 for each directory entry pointing to this inode.
n (no)	Ignore this error condition.

Message

```
PARTIALLY TRUNCATED INODE I=I (SALVAGE)
```

`fsck` has found inode *I* whose size is shorter than the number of blocks allocated to it. This condition should only occur if the system crashes while truncating a file. During preening the file system, `fsck` completes the truncation to the specified size.

Possible responses to the `SALVAGE` prompt are:

`y` (yes)	Complete the truncation to the size specified in the inode.
`n` (no)	Ignore this error condition.

Message

```
LINK COUNT TABLE OVERFLOW (CONTINUE)
```

An internal table for `fsck` containing allocated inodes with a link count of zero has no more room.

Possible responses to the `CONTINUE` prompt are:

`y` (yes)	Continue with the program. This error condition prevents a complete check of the file system. A second run of `fsck` should be made to recheck the file system. If another allocated inode with a zero link count is found, the error message is repeated.
`n` (no)	Terminate the program.

Message

```
B BAD I=I
```

Inode *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may also generate the `EXCESSIVE BAD BLKS` error message if inode *I* has too many block numbers outside the file system range. This error condition generates the `BAD/DUP` error message in Phases 2 and 4.

Message

```
EXCESSIVE BAD BLKS I=I (CONTINUE)
```

There are too many (usually more than ten) blocks with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system associated with inode *I*.

Possible responses to the `CONTINUE` prompt are:

`y` (yes)	Ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition prevents a complete check of the file system. A second run of `fsck` should be made to recheck this file system.
`n` (no)	Terminate the program.

Message

```
BAD STATE DDD TO BLKERR
```

An internal error has scrambled `fsck`'s state map to have the impossible value *DDD*. `fsck` exits immediately. If this occurs, contact your service representative or another qualified person.

Message

```
B DUP I=I
```

Inode *I* contains block number *B* that is already claimed by another inode. This error condition may also generate the `EXCESSIVE DUP BLKS` error message if inode *I* has too many block numbers claimed by other inodes. This error condition invokes Phase 1B and generates the `BAD/DUP` error message in Phases 2 and 4.

Message

```
BAD MODE: MAKE IT A FILE?
```

This message is generated when the status of a given inode is set to all ones, indicating file system damage. This message does not indicate disk damage, unless it appears repeatedly after `fsck -y` has been run. A response of `y` causes `fsck` to reinitialize the inode to a reasonable value.

Message

```
EXCESSIVE DUP BLKS I=I (CONTINUE)
```

There are too many (usually more than ten) blocks claimed by other inodes.

Possible responses to the `CONTINUE` prompt are:

`y` (yes)	Ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition prevents a complete check of the file system. A second run of `fsck` should be made to recheck the file system.
`n` (no)	Terminate the program.

Message

```
DUP TABLE OVERFLOW (CONTINUE)
```

An internal table in `fsck` containing duplicate block numbers has no more room.

Possible responses to the `CONTINUE` prompt are:

`y` (yes)	Continue with the program. This error condition prevents a complete check of the file system. A second run of `fsck` should be made to recheck the file system. If another duplicate block is found, this error message will repeat.
`n` (no)	Terminate the program.

Message

```
PARTIALLY ALLOCATED INODE I=I (CLEAR)
```

Inode *I* is neither allocated nor unallocated.

Possible responses to the `CLEAR` prompt are:

`y` (yes)	De-allocate inode *I* by zeroing out its contents.
`n` (no)	Ignore this error condition.

Message

```
INCORRECT BLOCK COUNT I=I (X should be Y) (CORRECT)
```

The block count for inode *I* is *X* blocks, but should be *Y* blocks. During preening, the count is corrected.

Possible responses to the CORRECT prompt are:

y (yes)	Replace the block count of inode *I* by *Y*.
n (no)	Ignore this error condition.

Phase 1B: Rescan for More DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the inode that previously claimed that block. When the duplicate block is found, the following informational message appears:

Message

```
B DUP I=I
```

Inode *I* contains block number *B* that is already claimed by another inode. This error condition generates the BAD/DUP error message in Phase 2. You can determine which inodes have overlapping blocks by examining this error condition and the DUP error condition in Phase 1.

Phase 2: Check Pathnames

This phase removes directory entries pointing to bad inodes found in Phases 1 and 1B. It reports error conditions resulting from:

- incorrect root inode mode and status
- directory inode pointers out of range
- directory entries pointing to bad inodes
- directory integrity checks

All errors in this phase are fatal if the file system is being preened, except for directories not being a multiple of the block size and extraneous hard links.

Phase 2 Error Messages

Message

```
ROOT INODE UNALLOCATED (ALLOCATE)
```

The root inode (usually inode number 2) has no allocate mode bits. This should never happen.

Possible responses to the `ALLOCATE` prompt are:

y (yes) Allocate inode 2 as the root inode. The files and directories usually found in the root will be recovered in Phase 3 and put into the `lost+found` directory. If the attempt to allocate the root fails, `fsck` will exit with the message

```
CANNOT ALLOCATE ROOT INODE
```

n (no) Terminate the program.

Message

```
ROOT INODE NOT DIRECTORY (REALLOCATE)
```

The root inode (usually inode number 2) of the file system is not a directory inode.

Possible responses to the `REALLOCATE` prompt are:

y (yes) — Clear the existing contents of the root inode and reallocate it. The files and directories usually found in the root will be recovered in Phase 3 and put into the `lost+found` directory. If the attempt to allocate the root fails, `fsck` will exit with the message:

```
CANNOT ALLOCATE ROOT INODE
```

n (no) — `fsck` will then prompt with `FIX`

Possible responses to the `FIX` prompt are:

y (yes) — Change the type of the root inode to directory. If the root inode's data blocks are not directory blocks, many error messages will be generated.

n (no) — Terminate the program.

Message

```
DUPS/BAD IN ROOT INODE (REALLOCATE)
```

Phase 1 or Phase 1B has found duplicate blocks or bad blocks in the root inode (usually inode number 2) of the file system.

Possible responses to the REALLOCATE prompt are:

y (yes) Clear the existing contents of the root inode and reallocate it. The files and directories usually found in the root will be recovered in Phase 3 and put into the `lost+found` directory. If the attempt to allocate the root fails, `fsck` will exit with the message:

```
CANNOT ALLOCATE ROOT INODE
```

n (no) `fsck` will then prompt with CONTINUE.

Possible responses to the CONTINUE prompt are:

y (yes) Ignore the DUPS/BAD error condition in the root inode and try to continue running the file system check. If the root inode is not correct, this may generate many other error messages.

n (no) Terminate the program.

Message

```
NAME TOO LONG F
```

An excessively long path name has been found. This usually indicates loops in the file system name space. This can occur if a privileged user has made circular links to directories. These links must be removed.

Message

```
I OUT OF RANGE I=I NAME=F (REMOVE)
```

A directory entry *F* has an inode number *I* that is greater than the end of the inode list.

Possible responses to the REMOVE prompt are:

y (yes)	Remove the directory entry *F*.
n (no)	Ignore this error condition.

Message

```
UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T TYPE=F (REMOVE)
```

A directory or file entry *F* points to an unallocated inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and name *F* are printed.

Possible responses to the REMOVE prompt are:

y (yes)	Remove the directory entry *F*.
n (no)	Ignore this error condition.

Message

```
DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T TYPE=F (REMOVE)
```

Phase 1 or Phase 1B has found duplicate blocks or bad blocks associated with directory or file entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

y (yes) Remove the directory entry *F*.

n (no) Ignore this error condition.

Message

```
ZERO LENGTH DIRECTORY I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
(REMOVE)
```

A directory entry *F* has a size *S* that is zero. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

y (yes) Remove the directory entry *F*; this will generate the BAD/DUP error message in Phase 4.

n (no) Ignore this error condition.

Message

```
DIRECTORY TOO SHORT I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (FIX)
```

A directory *F* has been found whose size *S* is less than the minimum size directory. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the FIX prompt are:

y (yes) Increase the size of the directory to the minimum directory size.

n (no) Ignore this directory.

Message

```
DIRECTORY F LENGTH S NOT MULTIPLE OF B (ADJUST)
```

A directory *F* has been found with size *S* that is not a multiple of the directory block size *B*.

Possible responses to the ADJUST prompt are:

y (yes)	Round up the length to the appropriate block size. During preening, the file system only a warning is printed and the directory is adjusted.
n (no)	Ignore the error condition.

Message

```
DIRECTORY CORRUPTED I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
(SALVAGE)
```

A directory with an inconsistent internal state has been found.

Possible responses to the SALVAGE prompt are:

y (yes)	Throw away all entries up to the next directory boundary (usually a 512-byte boundary). This drastic action can throw away up to 42 entries, and should be taken only after other recovery efforts have failed.
n (no)	Skip to the next directory boundary and resume reading, but do not modify the directory.

Message

```
BAD INODE NUMBER FOR '.' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
(FIX)
```

A directory *I* has been found whose inode number for "." does not equal *I*.

Possible responses to the `FIX` prompt are:

y (yes) Change the inode number for "." to be equal to *I*.

n (no) Leave the inode number for "." unchanged.

Message

```
MISSING '.' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (FIX)
```

A directory *I* has been found whose first entry is unallocated.

Possible responses to the `FIX` prompt are:

y (yes) Build an entry for "." with inode number equal to *I*.

n (no) Leave the directory unchanged.

Message

```
MISSING '.' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
CANNOT FIX, FIRST ENTRY IN DIRECTORY CONTAINS F
```

A directory *I* has been found whose first entry is *F*. `fsck` cannot resolve this problem. The file system should be mounted and entry *F* moved elsewhere. The file system should then be unmounted and `fsck` should be run again.

Message

```
MISSING '.' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
CANNOT FIX, INSUFFICIENT SPACE TO ADD '.'
```

A directory *I* has been found whose first entry is not ".". This should never happen. `fsck` cannot resolve the problem. If this occurs, contact your service representative or another qualified person.

Message

```
EXTRA '.' ENTRY I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (FIX)
```

A directory *I* has been found that has more than one entry for ".".

Possible responses to the `FIX` prompt are:

y (yes) Remove the extra entry for ".".

n (no) Leave the directory unchanged.

Message

```
BAD INODE NUMBER FOR '..' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
(FIX)
```

A directory *I* has been found whose inode number for ".." does not equal the parent of *I*.

Possible responses to the `FIX` prompt are:

y (yes) Change the inode number for ". ." to be equal to the parent of *I*. (Note that " . ." in the root inode points to itself.)

n (no) Leave the inode number for ". ." unchanged.

Message

```
MISSING '..' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (FIX)
```

A directory *I* has been found whose second entry is unallocated.

Possible responses to the `FIX` prompt are:

y (yes) Build an entry for ". ." with inode number equal to the parent of *I*. (Note that ". ." in the root inode points to itself.)

n (no) Leave the directory unchanged.

Message

```
MISSING '..' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
CANNOT FIX, SECOND ENTRY IN DIRECTORY CONTAINS F
```

A directory *I* has been found whose second entry is *F*. `fsck` cannot resolve this problem. The file system should be mounted and entry *F* moved elsewhere. The file system should then be unmounted and `fsck` should be run again.

Message

```
MISSING '..' I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F
CANNOT FIX, INSUFFICIENT SPACE TO ADD '..'
```

A directory *I* has been found whose second entry is not ". ." (the parent directory). `fsck` cannot resolve this problem. The file system should be mounted and the second entry in the directory moved elsewhere. The file system should then be unmounted and `fsck` should be run again.

Message

```
EXTRA '..' ENTRY I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (FIX)
```

A directory *I* has been found that has more than one entry for ". ." (the parent directory).

Possible responses to the `FIX` prompt are:

y (yes)	Remove the extra entry for . . (the parent directory).
n (no)	Leave the directory unchanged.

Message

```
N IS AN EXTRANEOUS HARD LINK TO A DIRECTORY D (REMOVE)
```

`fsck` has found a hard link *N* to a directory *D*. During preening, the extraneous links are ignored.

Possible responses to the `REMOVE` prompt are:

y (yes)	Delete the extraneous entry *N*.
n (no)	Ignore the error condition.

Message

```
BAD INODE S TO DESCEND
```

An internal error has caused an impossible state *S* to be passed to the routine that descends the file system directory structure. `fsck` exits. If this occurs, contact your service representative or another qualified person.

Message

```
BAD RETURN STATE S FROM DESCEND
```

An internal error has caused an impossible state *S* to be returned from the routine that descends the file system directory structure. `fsck` exits. If you encounter this error, contact your service representative or another qualified person.

Message

```
BAD STATE S FOR ROOT INODE
```

An internal error has caused an impossible state *S* to be assigned to the root inode. `fsck` exits. If this occurs, contact your service representative or another qualified person.

Phase 3: Check Connectivity

This phase checks the directories examined in Phase 2. It reports error conditions resulting from:

- unreferenced directories
- missing or full `lost+found` directories

Phase 3 Error Messages

Message

```
UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)
```

The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. During preening, the directory is reconnected if its size is non-zero; otherwise it is cleared.

Possible responses to the RECONNECT prompt are:

y (yes) Reconnect directory inode *I* to the file system in the directory for lost files (usually the `lost+found` directory). This may generate the `lost+found` error messages if there are problems connecting directory inode *I* to the `lost+found` directory. It may also generate the CONNECTED error message if the link is successful.

n (no) Ignore this error condition. This generates the UNREF error message in Phase 4.

Message

```
NO lost+found DIRECTORY (CREATE)
```

There is no `lost+found` directory in the root directory of the file system; During preening, `fsck` tries to create a `lost+found` directory.

Possible responses to the CREATE prompt are:

y (yes) Create a `lost+found` directory in the root of the file system. This may produce the message:

```
NO SPACE LEFT IN / (EXPAND)
```

See below for the possible responses. Inability to create a `lost+found` directory generates the message:

```
SORRY. CANNOT CREATE lost+found DIRECTORY
```

and aborts the attempt to link up the lost inode. This in turn generates the UNREF error message in Phase 4.

n (no) Abort the attempt to link up the lost inode. This generates the UNREF error message in Phase 4.

Message

```
lost+found IS NOT A DIRECTORY (REALLOCATE)
```

The entry for `lost+found` is not a directory.

Possible responses to the REALLOCATE prompt are:

y (yes) Allocate a directory inode, and change `lost+found` to reference it. The previous inode referenced by the `lost+found` directory is not cleared. Thus it will either be reclaimed as an UNREF'ed inode or have its link count ADJUST'ed later in this phase. Inability to create a `lost+found` directory generates the message:

```
SORRY. CANNOT CREATE lost+found DIRECTORY
```

and aborts the attempt to link up the lost inode. This in turn generates the UNREF error message in Phase 4.

n (no) Abort the attempt to link up the lost inode. This generates the UNREF error message in Phase 4.

Message

```
NO SPACE LEFT IN /lost+found (EXPAND)
```

There is no space to add another entry to the `lost+found` directory in the root directory of the file system. During preening, the `lost+found` directory is expanded.

Possible responses to the EXPAND prompt are:

y (yes) Expand the `lost+found` directory to make room for the new entry. If the attempted expansion fails, `fsck` prints the message:

```
SORRY. NO SPACE IN lost+found DIRECTORY
```

and aborts the attempt to link up the lost inode. This in turn generates the UNREF error message in Phase 4. Clear out unnecessary entries in the `lost+found` directory. This error is fatal if the file system is being preened.

n (no) Abort the attempt to link up the lost inode. This generates the UNREF error message in Phase 4.

Message

```
DIR I=I1 CONNECTED. PARENT WAS I=I2
```

This is an advisory message indicating that a directory inode *I1* was successfully connected to the `lost+found` directory. The parent inode *I2* of the directory inode *I1* is replaced by the inode number of the `lost+found` directory.

Message

```
DIRECTORY F LENGTH S NOT MULTIPLE OF B (ADJUST)
```

A directory *F* has been found with size *S* that is not a multiple of the directory block size *B*. (Note that this may reoccur if the error condition is not corrected in Phase 2).

Possible responses to the `ADJUST` prompt are:

y (yes)	Round up the length to the appropriate block size. During preening, only a warning is printed and the directory is adjusted.
n (no)	Ignore the error condition.

Message

```
BAD INODE S TO DESCEND
```

An internal error has caused an impossible state *S* to be passed to the routine that descends the file system directory structure. `fsck` exits. If this occurs, contact your service representative or another qualified person.

Phase 4: Check Reference Counts

This phase checks the link count information obtained in Phases 2 and 3. It reports error conditions resulting from:

- unreferenced files
- missing or full `lost+found` directory
- incorrect link counts for files, directories, symbolic links, or special files

- unreferenced files, symbolic links, and directories
- bad or duplicate blocks in files, symbolic links, and directories

All errors in this phase, except running out of space in the `lost+found` directory, are correctable if the file system is being preened.

Phase 4 Error Messages

Message

```
UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)
```

Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. During preening, the file is cleared if either its size or its link count is zero; otherwise it is reconnected.

Possible responses to the `RECONNECT` prompt are:

y (yes)	Reconnect inode *I* to the file system in the directory for lost files (usually the `lost+found` directory). This may generate the `lost+found` error message in Phase 4 if there are problems connecting inode *I* to the `lost+found` directory.
n (no)	Ignore this error condition. This will always invoke the `CLEAR` error condition in Phase 4.

Message

```
(CLEAR)
```

The inode mentioned in the error message immediately preceding cannot be reconnected. This message cannot appear if the file system is being preened, because lack of space to reconnect files is a fatal error.

Possible responses to the CLEAR prompt are:

y (yes) De-allocate the inode by zeroing out its contents.

n (no) Ignore this error condition.

Message

```
NO lost+found DIRECTORY (CREATE)
```

There is no `lost+found` directory in the root directory of the file system. During preening, `fsck` tries to create a `lost+found` directory.

Possible responses to the CREATE prompt are:

y (yes) Create a `lost+found` directory in the root of the file system. This may generate the message:

```
NO SPACE LEFT IN / (EXPAND)
```

See below for the possible responses. Inability to create a `lost+found` directory generates the message:

```
SORRY. CANNOT CREATE lost+found DIRECTORY
```

and aborts the attempt to link up the lost inode. This in turn generates the UNREF error message.

n (no) Abort the attempt to link up the lost inode. This generates the UNREF error message.

Message

```
lost+found IS NOT A DIRECTORY (REALLOCATE)
```

The entry for `lost+found` is not a directory.

Possible responses to the `REALLOCATE` prompt are:

y (yes) — Allocate a directory inode and change the `lost+found` directory to reference it. The previous inode reference by the `lost+found` directory is not cleared. Thus it will either be reclaimed as an `UNREF`ed inode or have its link count `ADJUST`ed later in this phase. Inability to create a `lost+found` directory generates the message:

```
SORRY. CANNOT CREATE lost+found DIRECTORY
```

and aborts the attempt to link up the lost inode. This generates the `UNREF` error message.

n (no) — Abort the attempt to link up the lost inode. This generates the `UNREF` error message.

Message

```
NO SPACE LEFT IN /lost+found (EXPAND)
```

There is no space to add another entry to the `lost+found` directory in the root directory of the file system. During preening, the `lost+found` directory is expanded.

Possible responses to the EXPAND prompt are:

y (yes) Expand the `lost+found` directory to make room for the new entry. If the attempted expansion fails, `fsck` prints the message:

```
SORRY. NO SPACE IN lost+found DIRECTORY
```

and aborts the attempt to link up the lost inode. This generates the UNREF error message. Clear out unnecessary entries in the `lost+found` directory. This error is fatal if the file system is being preened.

n (no) Abort the attempt to link up the lost inode. This generates the UNREF error message.

Message

```
LINK COUNT TYPE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X
SHOULD BE Y (ADJUST)
```

The link count for inode *I* is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed. During preening, the link count is adjusted unless the number of references is increasing, a condition that should never occur unless precipitated by a hardware failure. When the number of references is increasing during preening, `fsck` exits with the message:

```
LINK COUNT INCREASING
```

Possible responses to the ADJUST prompt are:

y (yes) Replace the link count of file inode *I* by *Y*.

n (no) Ignore this error condition.

Message

```
UNREF TYPE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
```

Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. Since this is a file that was not connected because its size or link count was zero, it is cleared during preening.

Possible responses to the CLEAR prompt are:

y (yes) De-allocate inode *I* by zeroing out its contents.

n (no) Ignore this error condition.

Message

```
BAD/DUP TYPE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
```

Phase 1 or Phase 1B has found duplicate blocks or bad blocks associated with inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This message cannot appear when the file system is being preened, because it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

y (yes)	De-allocate inode *I* by zeroing out its contents.
n (no)	Ignore this error condition.

Phase 5: Check Cylinder Groups

This phase checks the free block and used inode maps. It reports error conditions resulting from:

- allocated blocks in the free block maps
- free blocks missing from free block maps
- incorrect total free block count
- free inodes in the used inode maps
- allocated inodes missing from used inode maps
- incorrect total used inode count

Phase 5 Error Messages

Message

```
CG C: BAD MAGIC NUMBER
```

The magic number of cylinder group C is wrong. This usually indicates that the cylinder group maps have been destroyed. When running interactively, the cylinder group is marked as needing reconstruction. This error is fatal if the file system is being preened.

Message

```
BLK(S) MISSING IN BIT MAPS (SALVAGE)
```

A cylinder group block map is missing some free blocks. During preening the maps are reconstructed.

Possible responses to the SALVAGE prompt are:

y (yes)	Reconstruct the free block map.
n (no)	Ignore this error condition.

Message

```
SUMMARY INFORMATION BAD (SALVAGE)
```

The summary information was found to be incorrect. During preening, the summary information is recomputed.

Possible responses to the SALVAGE prompt are:

y (yes)	Reconstruct the summary information.
n (no)	Ignore this error condition.

Message

```
FREE BLK COUNT(S) WRONG IN SUPERBLOCK (SALVAGE)
```

The super-block free block information was found to be incorrect. During preening, the super-block free block information is recomputed.

Possible responses to the SALVAGE prompt are:

y (yes)	Reconstruct the super-block free block information.
n (no)	Ignore this error condition.

Cleanup Phase

After a file system has been checked, a few cleanup functions are performed. The cleanup phase displays advisory messages about the status of the file system.

Message

```
V files, W used, X free (Y frags, Z blocks)
```

This is an advisory message indicating that the file system checked contains *V* files using *W* fragment sized blocks, and that there are *X* fragment sized blocks free in the file system. The numbers in parentheses break the free count down into *Y* free fragments and *Z* free full sized blocks.

Message

```
***** FILE SYSTEM WAS MODIFIED *****
```

This is an advisory message indicating that the file system was modified by `fsck`. If this file system is mounted or is the current root file system, you should reboot. If the file system is mounted you may need to unmount it and run `fsck` again; otherwise the work done by `fsck` may be undone by the in-core copies of tables.

Checking `bfs` File Systems

All I/O for `bfs` file systems is synchronous; therefore, `bfs` file systems will not get corrupted even if proper shutdown procedures are not observed.

Corruption of a `bfs` file system is likely to occur only if the system crashes during the process of compaction. See the section "Compaction" earlier in this chapter for a description of compaction.

`fsck` checks the sanity words stored in the `bfs` super-block to see if compaction was in process before the system crashed. If it was, `fsck` completes the compaction of the file system.

7 Machine Management

DRAFT COPY
February 1, 1992
File: Cmachine

DRAFT COPY
February 1, 1992
File: Cmachine

DRAFT COPY
February 1, 1992
File: Cmachine

Introduction

This chapter tells you how to perform tasks that affect the way in which your computer operates or that provide information on the current state of your computer.

Some of the tasks described in this chapter can be performed through the `sysadm` system administration interface. Others require execution of shell-level commands, and some require knowledge of firmware-level operations available on your computer. The operations described in this chapter, and the ways that you can perform them, are:

Figure 7-1: Machine Management Tasks

Task	Interface
Displaying system configuration information	sysadm/shell
Reboot system	sysadm/shell
Shutdown system	sysadm/shell
Displaying information about the users on your system	sysadm/shell

System Administration Interface

To access the system administration menu for machine management, log in as `root` and type:

```
sysadm machine
```

The following menu will appear on your screen:

```
        Machine Configuration Display and Powerdown

configuration  - System Configuration Display
reboot         - Stops All Running Programs and Reboots Machine
shutdown       - Stops All Running Programs and Halts Machine
whos on        - Displays List of Users Logged onto Machine
```

If you prefer not to use the menus, you can perform the same tasks by executing shell-level commands instead. The following table shows the shell commands that correspond to the tasks listed on the menu.

Figure 7-2: Machine Tasks and Shell Commands

Task to Be Performed	`sysadm` Task	Shell Command
Displaying configuration info	`summary`	`prtconf`(1)
Rebooting your system	`reboot`	`shutdown`(1M)
Displaying system name	`system`	`uname`(1)
Displaying who is logged on	`whos on`	`who`(1)

Each task listed above is explained fully later in this chapter. Details on each command can be found in the *System Administrator's Reference Manual*.

The remainder of this chapter provides background information and describes how to perform all the tasks mentioned in Figure 7-1 through the shell or from firmware mode, as appropriate.

If you are not an experienced administrator, you are encouraged to perform these tasks, where possible, through the `sysadm` interface.

The next section provides some guidelines that should be followed whenever you perform machine management tasks.

An Overview of Machine Management

The task of managing a computer involves many responsibilities. Typically, the actions you take when performing machine management tasks will affect the operation of the computer as a whole, and every user on the computer.

In addition to the information in this chapter, you will also want to consult the hardware manual that accompanies your computer. This hardware manual will provide a detailed discussion of machine management and descriptions of firmware commands that are available on your computer.

Many administrative tasks require the system to be shut down to a system state other than multi-user state (see "System States" later in this chapter for a description).

In many system states, conventional users cannot access the system. Therefore, you should try to perform tasks that require a change in system state at a time when you will interfere least with users' work. Sometimes situations arise that require the system to be taken down with little or no warning to the users. Try to give users as much advance notice as possible about events affecting the use of the machine. When you must take the system out of service, tell them when it will become available again.

Use the news (see `news`(1)) and the message of the day (contained in the file `/etc/motd`) to keep users informed about changes in hardware, software, policies, and procedures.

Follow the guidelines below whenever you do a task that requires the system to leave multi-user state.

1. Schedule tasks that will affect service for periods of low system use. Inform users of the times that the system will be unavailable.
2. Check to see who is logged in before taking any actions that would affect active users. The `whodo`(1M) and `who`(1) commands can be used to see who is on the system.
3. If the system is in use, warn the users about changes in system states or pending maintenance actions. If you must interrupt service immediately, broadcast a warning to all users' screens with the `wall`(1M) command. Give users a reasonable amount of time to stop working and log off before you take the system down.

Keep a record of your administrative activities in a system log notebook (if you have one). This log can prove invaluable in recovering your system should you make a serious error.

The remainder of this chapter provides:

- a brief overview of the boot procedure that tells you how the operating system is loaded from disk into memory and executed
- an overview of system states that tells you what happens after the machine is booted and it enters the default system state, how to change the default system state, and how to change system states after powerup
- detailed procedures for the tasks listed in Figure 7-1

In previous releases of UNIX System V, it was assumed that a System V (`s5`) file system was defined on the root partition, and that the programs and data files needed during booting (and in firmware mode) reside in the root file system.

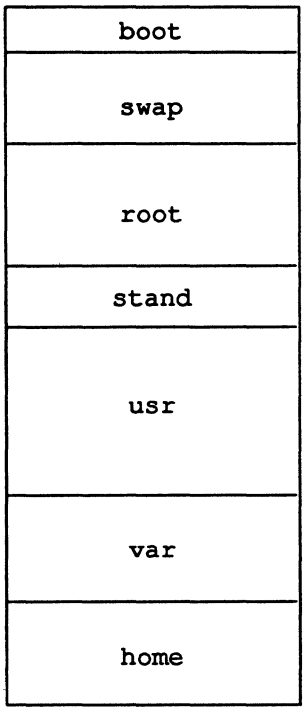
With UNIX System V/68 or V/88 Release 4, the boot procedure no longer depends on the file system type of the root file system. This file system independent boot procedure is implemented through the use of a new file system type, the boot file system type (`bfs`).

The following section explains the location and contents of the boot file system. It also explains the location and contents of the boot partition, a separate disk partition (not a file system) that holds the programs used to load the bootable programs found in the boot file system.

The `stand` and `boot` Partitions

Figure 7-3 shows the general layout of a single disk formatted into the default partitions. The disk layout shown in the figure is a generalized single-disk layout; the layout on your machine may be different. (Refer to the installation instructions accompanying the release for the default partitioning.)

Figure 7-3: Generalized Hard Disk Default Partitioning



The two partitions of interest to the boot procedure are the `boot` and `stand` partitions.

The `boot` partition is not a file system, but a special area of the disk that contains the boot programs. The following is the boot process as executed on the supported Delta Series and DeltaSERVER platforms:

- Turn your machine on, and the firmware prompts you for the name of the program to boot or automatically loads the `boot` program from the boot partition into a special area of memory.

- `boot` loads the bootable operating system `/stand/unix`.

The bootable operating system `unix` is found in the `stand` partition. This partition has a file system defined on it that contains all the bootable programs and data files used during the boot procedure. It can be identified using the `prtvtoc`(1M) command as the partition with the tag of 6 (indicating `V_STAND`). The file system defined on this partition is mounted by default as `/stand`.

The contents of `/stand` include the following:

`unix` — This is the bootable operating system. When the boot program is loaded, it searches for and loads this program into memory, and then passes control to it. Once the bootable operating system is running, various system daemons are started, and the system enters one of the `init` states (see the description of `/etc/inittab` in this chapter). Note that the filename `unix` is linked to `/stand/unix` for compatibility with earlier releases; you can enter `unix` or `/stand/unix` at any prompt requiring the name of the bootable operating system.

`system` — This is the system configuration file; it contains a description of the hardware and software modules that must be included in the bootable operating system (`/stand/unix`) for correct configuration of the system. If the time stamp of the `system` file is newer than the time stamp of the `/stand/unix` file, a reconfiguration of the bootable operating system is necessary before the boot program can pass control to it. Note that the filename `system` is linked to `/stand/system` for compatibility with earlier releases; you can enter `system` or `/stand/system` at any prompt requiring the name of the system configuration file.

`mUNIX` — This is a version of the UNIX operating system that is run only during the configuration of a new bootable operating system (`unix`).

`mini_system` — This is the system configuration file for the `mUNIX` program.

`EDTP_*` — Programs beginning with EDTP_ probe for the existence of peripheral boards. Normally, `boot` will run each one of these programs before loading `/stand/unix`.

`noprobe`	This file, if it exists, tells `boot` to load `/stand/unix` without first running the `EDTP_*` programs.
`noautoconfig`	This file, if it exists, tells the startup scripts and `rc6(1M)` to proceed without reconfiguring the kernel, even if it would normally be needed. See `buildsys(1M)` and `shutdown(1M)`.
`edt_data`	The Equipped Device Table (EDT) file has hardware data specific to the peripheral boards that may be configured into the computer. `boot` and the `EDTP_*` programs read it while building the in-core `EDT` before loading `/stand/unix`.

See "Configuring the UNIX Operating System" in the "Performance Management" chapter for more information on the operating system files in `/stand`.

Operations on the `boot` Partition

The only operation performed on the `boot` partition is the loading of the boot programs; no file system is defined on this partition.

The boot programs are loaded using the `dinit`(1M) command. The boot programs are found in the root file system under the `/usr/lib` directory. The command is used as follows:

```
# dinit -b /usr/lib/boot ddef file
/dev/rdsk/prefix_cXdYsZ
```

where the *prefix* is the device type, `c`*X* is the controller number of that device, `d`*Y* is the device number for the controller, and `s`*Z* is the slice number. See `intro`(7) for complete lists of controllers, devices, and slices.

The *ddef file* and /dev/rdsk/*prefix*_cXdYsZ used in the example above may vary depending on the number and types of disks you have connected to your computer. See `dinit`(1M) for a complete description of this command.

This command is normally not used unless you are manually repartitioning your hard disks or creating a new bootable disk (see "Making New Bootable Disks," in this chapter). In the delivered system, a boot partition is defined on the first integral hard disk. This partition already contains the `boot` program.

Operations on the `stand` Partition

In the delivered system, the `stand` partition contains a file system of type `bfs`; this boot file system is mounted as `/stand` by default.

The boot file system is a flat file system, with only one directory. You can copy and move regular files to and from a boot file system, but cannot make directories or create other special files in the `bfs` file system. You can, however, use file system commands such as `mount`(1M), `umount`(1M), and `fsck`(1M).

It is recommended that you use this file system for boot- and configuration-related files only.

If you want to, you can use `mkfs` to create `bfs`-type file systems on other disk partitions or on other disks (to make them bootable). See the section in this chapter "Making New Bootable Disks," the "File System Administration" chapter, and the `mkfs`(1M) manual page for more information. Having multiple boot file systems on one or more disks is particularly useful in an operating system development environment.

You must define `bfs`-type file systems only on hard disk partitions with a `tag` of 6. Do not make `bfs`-type file systems on any other devices. Similarly, do not make a file system of any other type than `bfs` in a partition with a `tag` of 6. Doing so may render your machine unbootable. See the references cited in the above paragraph, and the `fmthard`(1M) and `prtvtoc`(1M) manual pages in the *System Administrator's Reference Manual* for more information.

Any disk can contain `stand` and `boot` partitions. It is possible to boot the system from any disk that has the `stand`, `boot`, and `root` partitions.

A disk can also have multiple `stand` partitions with a boot file system defined on each one. Each partition can contain a version of the bootable programs. The `next part` boot loader command (described below) can be used to switch partitions from which you can request a manual boot from firmware mode.

When you are in firmware mode, however, you can access all `stand` partitions on a disk. When you enter firmware mode, a prompt similar to the following is displayed:

```
COLD Start
1) Continue System Start Up
2) Select Alternate Boot Device
3) Go to System Debugger
4) Initiate Service Call
5) Display System Test Errors
6) Dump Memory to Tape
Enter Menu #:
```

Enter 3, followed by a (RETURN). The following prompt is displayed:

```
141-Diag
```

for M68000 family of processors or

```
181-Diag>
```

for M88000 family of processors.

Enter **bo x y**, followed by a (RETURN). `x` and `y` are the controller and device numbers for the disk. See the *MVME181BUG 181Bug Debugging Package User's Manual* for more details. The following prompt is displayed:

```
Enter name of boot file (or 'q' to quit):
```

Use the `next part` command to switch the working `stand` partition, or just press (RETURN) to display the directory of the current `stand` partition and its contents.

The file `/etc/vfstab` contains the pathname of the `stand` partition to be mounted during single- and multi-user states and is used during configuration of a new operating system on powerup or reboot. This should always be the one defined on the lowest numbered partition on your default boot disk (usually partition 2).

Boot Scenarios

There are several reasons that a system boot takes place:

- Power is newly applied to the machine. In this case, the boot program looks for the `unix` and `system` files in the first `stand` partition on the default boot disk. If `unix` has the same or a later timestamp than `system`, then the boot program loads and executes `unix`; if `system` is newer, the boot program will execute `mUNIX`, and then `buildsys`(1M) to configure a new `unix`. When the new `unix` is built, the system boots the new `unix` and the system comes up in the state defined by the `initdefault` entry in `/etc/inittab` [see `inittab`(4)].
- An explicit request is made to enter firmware mode or reboot mode while the UNIX system is running (e.g., `init 5` or `6` or `shutdown -i5` or `-i6`). In this case, all system activity is stopped (all processes are killed, etc.). The system checks to see if a new `unix` needs to be configured (as above), and if so, configures one. Then, the system unmounts all file systems and boots `unix`. The system comes down to the firmware mode.
- A reboot is requested from firmware. From the firmware prompt, the user can specify the name of any executable (such as `unix`) or text file (such as `system`) in `/stand`. If an executable is specified, the boot program loads it and branches to it. If a text file is specified, the boot program starts `mUNIX`, and then executes `buildsys`(1M) to configure a new `unix`, using the text file specified as the `system` file for the new `unix`. When the new `unix` is built, the system loads and executes it, and comes up in the state defined by the `initdefault` entry in `/etc/inittab`.
- A crash occurs and the machine automatically reboots. In this case, the procedure is the same as for the first case, above.

See the section, "Configuring the UNIX Operating System," in the "Performance Management" chapter for a description of the configuration process.

The /boot Directory

There is a directory called /boot in the root file system that is not to be confused with the stand and boot partitions. Files in the /boot directory are used during the configuration of a new unix. See "Configuring the UNIX Operating System" in the "Performance Management" chapter for more details.

System States

Once powered up, the UNIX system operates in one system state or another. Having different system states allows you to perform administrative and operational functions with the confidence that the computer is operating with a selected group of active processes.

There are many system states, but first a note of clarification. Many terms are used to identify the particular operating state of the system. The system state is also called the "init state," "run state," "run level," or "run mode." In this guide, the term system state is used to identify the system's operating state.

There are several ways that you can change from one system state to another:

- with the `shutdown` command
- with the `powerdown` command
- with the `init` command

The first two commands call the `init` command during their execution. (Refer to the *System Administrator's Reference Manual* for complete information on using and changing system states with these three commands.) Figure 7-4 shows the factory-configured states in which your system can operate.

Figure 7-4: System States

System State	Description
0	State 0 is the powerdown state. Shut the machine down so it is safe to remove the power. Have the machine remove its own power (if possible).
1	State 1 is referred to as the administrative state. In state 1, file systems required for multi-user operations are mounted, and logins requiring access to multi-user file systems can be used. Other multi-user services are unavailable. Note that going to state 1 is only meaningful when the system is coming up from firmware or state s(S). When going to state 1 from state 2, no services are stopped, no processes are killed, and the system continues operating just as in state 2.
s, or S	State s (or S) is referred to as the single-user state. It is used to install or remove software utilities, run file system backups or restores, and check file systems. All multi-user file systems are unmounted and the system can only be accessed through the console. Logins requiring access to multi-user file systems cannot be used.
2	State 2 is the normal operating state for the system (also called multi-user state) as delivered. File systems are mounted, and multi-user services are started.
3	State 3 is used to start Remote File Sharing (RFS), connect your computer to an RFS network, mount remote resources, and offer your resources automatically. Known as the RFS state.
4	State 4 is the user-defined system state. This state is not used in the delivered system.
5	State 5 is the firmware state. This state is used to run special firmware commands and diagnostics. An example of the former is executing `/stand/unix` to reboot the system.

Figure 7-4: System States (continued)

6	State 6 is used to halt and reboot the operating system to the state defined by the `initdefault` entry in the `/etc/inittab` file.
a, b, or c	States a, b, or c are used to process `/etc/inittab` file entries. These are pseudo-states that may be used to run certain commands without changing the current system state. They are supported by `init` but unused in the delivered system.
Q or q	State Q (or q) is used to re-examine the `/etc/inittab` file.

As delivered, the system enters the multi-user state on powerup. File systems are mounted, daemons started, and system services (such as `lp` and `uucp`) are made available. These activities are performed by the `rc2` script [see `rc2`(1M)].

Not all activities, however, should be performed in multi-user state. For example, you should never change your system's configuration while users are accessing it because much data may be lost. By changing the system to single-user state, you can be sure that you are the only person logged on to the system and, therefore, that it is safe to perform critical system administration tasks, such as changing the system configuration.

Entering the Multi-User State During Power Up

When you power up or reboot your system, it enters the default system state that appears in the `initdefault` line of the `/etc/inittab` file. Normally, this line contains a `2` in the second field, indicating that the system is to be put into multi-user state by default. The following is an example `initdefault` entry:

```
is:2:initdefault:
```

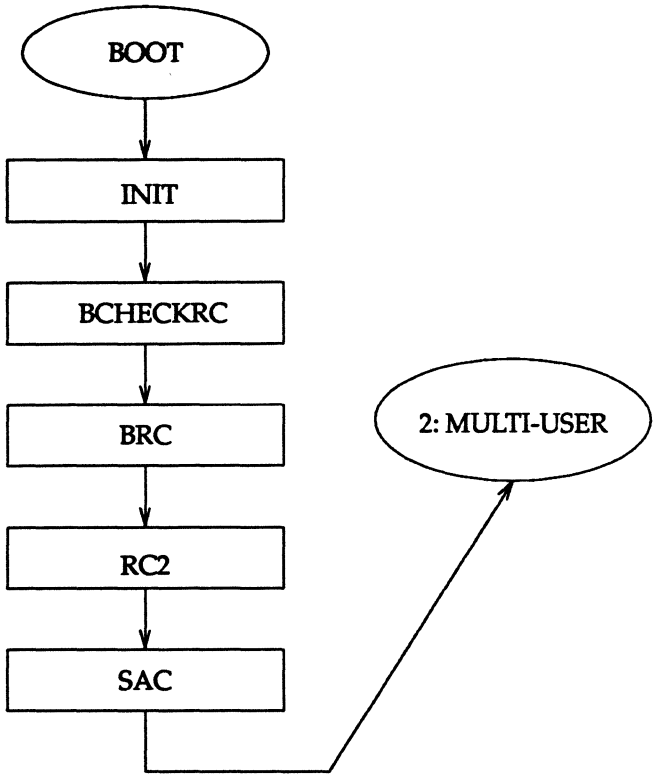
You can change the default run state of the computer by changing the second field of the `initdefault` line in your `/etc/inittab` file. However, do not change the second field to a `1` or an `S` as you may render the system unbootable. Use a lowercase `s` when you want the default system state to be single-user.

Once the machine boots and control passes to the `unix` program, the following events occur as the system goes to multi-user state:

1. Early system initializations are started by `init`.
2. File systems are checked and mounted by `bcheckrc`.
3. Other startup functions are performed by `brc`.
4. The system state change is prepared by the `rc2` procedure.
5. The system is made public via the spawning of the service access facility (`ttymon` and `sac`).

Figure 7-5 shows the most important events that occur as `unix` comes up in multi-user state.

Figure 7-5: A Look at System Initialization



Early Initialization

After the operating system is loaded into core memory via the boot programs (see the section "The Boot Procedure," in this chapter) the `init` process is created. It immediately scans `/etc/inittab` for entries of the type `sysinit`. A sample listing of these entries is shown in Figure 7-6.

Figure 7-6: Sample `sysinit` Entries in an `/etc/inittab` File

```
ap::sysinit:/sbin/autopush -f /etc/iu.ap
fs::sysinit:/sbin/bcheckrc </dev/console >/dev/console 2>&1
ck::sysinit:/sbin/setclk </dev/console >/dev/console 2>&1
xdc::sysinit:/sbin/sh -c 'if [ -x /etc/rc.d/0xdc ] ; then /etc/rc.d/0xdc ;
fi' >/dev/console 2>&1
ac::sysinit:/sbin/ckmunix </dev/console >/dev/console 2>&1
```

These entries are executed in the order in which they are listed in the file. Once the first entry is executed, communication between the system and the console is established. Subsequent entries define the standard input and the standard output as `/dev/console`.

Preparing the System State Change

Now the system must be placed in a particular system state. First, `init` scans each entry in the `/etc/inittab` file for the value `initdefault` in the third field (the "action" field). Including the value `initdefault` in the third field of an entry means that the default system state is defined in the second field of that entry. For example, in the first line of Figure 7-7, the 2 in the second field, followed by the value `initdefault` in the third field, means that the default system state for this system is system state 2 (multi-user state).

Once `init` has identified the default system state as system state 2, it searches the table for all entries that specify processes for system state 2. Typical system state 2 entries are shown in Figure 7-7.

Figure 7-7: System State 2 Processes

```
is:2:initdefault:
s2:23:wait:/sbin/rc2 >/dev/console 2>&1 </dev/console
sc:234:respawn:/usr/lib/saf/sac -t 300
co:1234:respawn:/usr/lib/saf/ttymon -g -p "Console Login: " -m ldterm
-d /dev/console -l console
ct:234:respawn:/usr/lib/saf/ttymon -g -m ldterm -d /dev/contty -l
contty
he:234:respawn:/usr/sbin/hdelogger
```

The processes defined in these entries are executed before the system enters multi-user system state during powerup (see "System State Directories" later in this chapter). The `rc2` script accomplishes (among other things) the following:

- sets up and mounts the file systems.
- Starts the `cron` daemon.
- Displays the current system hardware configuration.
- Makes `uucp` available for use (if installed).
- Makes the LP print service available for use (if installed).

Next, `init` searches the `/etc/inittab` file for system state 2 processes and executes them in the order found in the file. In this example, `init` performs the following functions:

- Starts the service access controller (`sac`) for the ports.
- Starts the tty monitor (`ttymon`) for the console.
- Starts the error logging daemon (`hdelogger`) for the hard disk.

When this is complete, the full multi-user environment is established and the system is in system state 2. The system is now available for users to log in as shown by the `login:` prompt that appears on the user's terminals.

Changing System States After Power Up

There may be times when you want to change system states after you have powered up your computer. For example, you may want to perform system diagnostics, which require you to have your computer in the firmware state (system state 5).

For most administrative duties, you will want to change the system state from multi-user to system states `s, 1, or 5`. The general procedure used to change system states once the computer is powered up is as follows:

1. Log in as `root`. You will be prompted for the root password. After you have entered it, the root prompt (#) will appear.
2. Check to see if any users are on the system by typing:

   ```
   who -H
   ```

   A typical response might be:

   ```
   NAME       LINE         TIME
   root       console      Aug 31 10:28
   marcus     term/12      Aug 31 09:43
   ```

   If there are any users logged in to the system, go to step 3; if not, go to step 4.
3. To notify users that the system is about to be shut down, type `wall`, followed (on one or more separate lines) by a message announcing the shutdown. To end your message, press RETURN and then type CTRL-d, as shown in the following example:

```
# wall
The system will be coming down in 5 minutes.
Please log off.
CTRL-d


Broadcast Message from root (console) on unix Wed Oct 5 07:30:27...
The system will be coming down in 5 minutes.
Please log off.
```

4. Execute the `shutdown` or `init` command with an argument that specifies the desired system state. (See Figure 7-4 for a list of available arguments.)

   The `init` process searches the `/etc/inittab` file for processes that match the new system state and executes them in the order that they appear in the file.

   Key procedures (such as `/sbin/shutdown`, `/sbin/rc0`, `/sbin/rc2`, and `/sbin/rc3`) are run to initialize the new state.

   The system then enters the new state. If the new system state is either state s (single-user) or state 5 (firmware), sensitive administrative procedures can then be performed.

Changing to Single-User State (System State s)

Some administrative functions, such as backing up the hard disk, can be done only when the system is in the single-user state. The normal way to go to single-user state is by running the `shutdown` command. This command executes all the files in the `/sbin/rc0.d` and `/sbin/shutdown.d` directories by invoking the `/sbin/rc0` procedure. The `shutdown` command accomplishes, among other things, the following:

- Closes all open files and stops all user processes.
- Stops all daemons and services.

- Writes all system buffers out to the disk.
- Unmounts all file systems necessary for multi-user operations, but not needed in single-user state (such as `/home`).

There are two ways to change to single-user state:

1. You can run the `shutdown -is` command (recommended).
2. You can run the `init s` command.

After these programs complete the change to single-user state, you get the message:

```
INIT: SINGLE USER MODE

Type Ctrl-d to proceed with normal startup
(or give root password for system maintenance):
```

Type the root password, press RETURN and you will get the single-user prompt (#). You are now ready to perform tasks that should be done only in the single-user state.

Changing to Multi-User State (System State 2)

System state 2 is the normal operating state of the system when it is not connected to a network. In this system state, several users can be logged in at once and use the system's resources. To change to multi-user state, run `init 2`.

This procedure executes the `/sbin/rc2` script which runs processes in the `/sbin/rc2.d` directory. Running `init 2` also initiates the Service Access Facility which manages access to your system through ports and other communication devices. (See the "Service Access" chapter for details.)

The `/var` file system must be mounted before executing `init 2`.

Changing to RFS State (System State 3)

Before you can perform administrative tasks associated with the Remote File Sharing (RFS) utilities, you must change the system to system state 3 (defined as the RFS state). To change to the RFS state, run `init 3`.

This procedure executes both the `/sbin/rc2` and `/sbin/rc3` scripts. These scripts run processes in all directories associated with system states 2 and 3, respectively. Running `init 3` also initiates the Service Access Facility which manages access to your system through ports and other communication devices. (See the "Service Access" chapter for details.)

In addition to the scripts in `/sbin/rc2.d` (multi-user state directory), scripts in `/sbin/rc3.d` (the RFS state directory) are run to do the following, if configured:

- Start Remote File Sharing and connect your computer to a Remote File Sharing network.
- Advertise your resources to remote computers.
- Mount remote resources on your computer.

If you are in a Remote File Sharing environment, you may want to change your `initdefault` entry in `/etc/inittab` from 2 to 3, so that you automatically come up in Remote File Sharing state when you reboot your computer.

The `/var` file system must be mounted before executing `init 3`.

Changing to Firmware State and Reboot States (System States 5 and 6)

Firmware state (defined as system state 5) is used both to access diagnostics and programs that reside in `/stand`.

Rebooting the operating system (defined as system state 6) kills active processes and forces the machine into a condition similar to that during powerup. After you install a software package that requires reinitialization of the system for proper operation, reboot the computer.

To go to the firmware state or to reboot your computer after reconfiguring it, run the `shutdown` command with the `-i5` or the `-i6` option. System states 5 and 6 are similar because both can exist only when the computer is under firmware control. The difference between the two states is one of duration. Your computer will stay in state 5 as long as you desire; it will stay in state 6 only long enough to reconfigure the operating system (if necessary) and to restart the initialization of it (that is, to reboot the system).

Figure 7-8: System State 5 and 6 Processes (from the Sample `/etc/inittab` File)

```
fl:056:wait:/sbin/led -f    # start green LED flashing
s0:5:wait:/sbin/rc0 firmware >/dev/console 2>&1 </dev/console
s6:6:wait:/sbin/rc6 reboot >/dev/console 2>&1 < /dev/console
```

The `rc0` script runs processes in all directories associated with system state `0`. The `rc6` script runs processes in all directories associated with system state `6`.

Turning the System Off

To turn off your system, use the `shutdown -i0` command. (There may also be a switch on the cabinet of your computer that allows you to power down the machine. See the installation guide for your computer.) The following entries in the `/etc/inittab` file apply to powering the system down:

Figure 7-9: System State 0 Processes (from the Sample `/etc/inittab` File)

```
fl:056:wait:/sbin/led -f    # start green LED flashing
s0:0:wait:/sbin/rc0 off >/dev/console 2>&1 </dev/console
```

The `rc0` procedure is called to clean up and stop all user processes, daemons, and other services, to unmount the file systems and bring the machine to a halted state.

System State Directories

System states 0, 2, and 3 each have a directory of files that are executed in transitions to and from that state. These directories are `/sbin/rc0.d`, `/sbin/rc2.d`, and `/sbin/rc3.d`, respectively. Most files in these directories are linked to files in the `/sbin/init.d` directory. Typically, their purpose is to start and stop various system services or daemons.

The system state files are named according to the following conventions:

SNNname

or

KNNname

Each filename consists of three parts:

`S` or `K`
: The first letter specifies whether the process should be started (`S`) or killed (`K`) on entering the new system state.

NN
: The next two characters are a number from 00 to 99. They show the order in which the files will be started (S00, S01, S02, and so on) or killed (K00, K01, K02, and so on).

name
: The rest of the filename is the name of the file in the `/sbin/init.d` directory to which this file is linked.

For example, the `/sbin/init.d/cron` shell script is linked to the `/etc/rc2.d/S75cron` and `/etc/rc0.d/K70cron` files. The `/sbin/init.d/cron` script will execute `/usr/bin/cron` when run with the `start` option; it will kill the `cron` process when run with the `stop` option.

When you run `init 2`, `init` runs the scripts in `/etc/rc.2`, one of which is `S75cron`. The script `S75cron` is executed with the `start` option as follows:

```
sh S75cron start
```

Similarly, when you run `init 0`, `/sbin/rc0.d/K70cron` is executed with the `stop` option:

```
sh K70cron stop
```

Running either of these scripts is the same as running `/sbin/init.d/cron` with the appropriate `start/stop` option.

Because these files are shell scripts, you can read them to see what they do. You can also change the files, although it is preferable to create your own versions because the delivered scripts may change in future releases.

Follow these rules when creating your own scripts:

- Place the file containing your script in the `/sbin/init.d` directory.
- Link the script [see `ln`(1)] to files in appropriate system state directories, using the naming convention described above.

Changing to Firmware Mode

You must change to firmware mode to run any of the firmware programs that came with your machine. Firmware procedures include the use of the firmware-resident commands and the use of the bootable programs provided as part of the Essential Utilities. They might perform such functions as running diagnostics on system software or changing the firmware password. See the hardware guide provided with your computer for a full description of the available firmware commands and instructions for running them.

To change to firmware mode, you must be logged in as `root` in the / directory and follow these steps.

Step 1 Type:

```
shutdown -i5
```

A message appears on the console confirming that a shutdown has started. A message is also automatically broadcast that informs users of the shutdown and directs them to log off or risk their files being damaged.

Step 2 If other users are logged in, the system waits for a grace period of 60 seconds before asking you if you want to continue with the shutdown; answer `y` (for yes) at this prompt.

After you answer this question, the shutdown process starts and the system changes to firmware mode. The following screen shows an example of the system output displayed after you enter the `shutdown` command with other users logged in:

```
Shutdown started.    Thu May 16 17:21:32 EDT 1989
Broadcast Message from root  (console)  Thu May 16 17:21:34 EDT 1989
THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged.
Do you want to continue (y or n): y (RETURN)

INIT: New run level: 5
The system is coming down.  Please wait.
System services are now being stopped.
The system is down.

NOTICE: Return to Firmware requested (0)
```

The system will stop. To enter firmware mode, reset the system. When you see a prompt similar to:

```
Copyright Motorola Inc. 1988, 1989, All Rights Reserved

MVME181 Debugger/Diagnostics Release Version 4.0 - 12/07/89
```

Press the **BREAK** key or some other key. The following is displayed.

```
COLD Start
1) Continue System Start Up
2) Select Alternate Boot Device
3) Go to System Debugger
4) Initiate Service Call
5) Display System Test Errors
6) Dump Memory to Tape
Enter Menu #:
```

See the hardware guide for your computer for specific instructions on running firmware programs.

See the "Returning from Firmware" section in this chapter for information on returning to an operating system state from firmware.

Powering Down Your Machine

The powerdown procedure differs depending on whether the system is in multi-user or single-user state.

From Multi-User State

If your system is in multi-user state, turn off your computer by running the `shutdown -i0` command. This command flushes the system buffers, closes any open files, stops all user processes and daemons currently running, unmounts file systems, and then removes power from the computer.

Do not pull the plug until the powerdown procedure is completely finished and the message "System secured for powering down." is displayed.

1. Before powering down your system, check to see who is logged on at the time by executing `who -H`, as shown in the example below.

```
$ who -H
NAME       LINE         TIME
rht        term/12      Aug 31 10:28
mark       term/14      Aug 31 09:43
root       console      Aug 31 08:32
$
```

   If there are any users logged in on the system, go to step 2; if not, go to step 3.

2. To notify any users that the system is about to be shut down, type `wall` followed by a message announcing the shutdown. To end your message, type CTRL-d as shown in the following example.

```
$ wall
The system will be coming down in 5 minutes.
Please log off. CTRL-d
Broadcast Message from root (console) on unix Wed Feb 26 07:30:27...
The system will be coming down in 5 minutes.
Please log off.
$
```

3. If there are no users logged on, or after you have notified all users who are logged on, power down the system by executing the following command as root from the / directory:

   ```
   shutdown -i0
   ```

   The next screen shows an example of the system output displayed at this time.

```
Shutdown started.    Thu May 16 17:10:57 EDT 1989

Broadcast Message from root (console)  Thu May 16 17:10:59
THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged.
Do you want to continue (y or n): y RETURN

INIT: New run level: 0
The system is coming down.  Please wait.
System services are now being stopped.
The system is down.
```

To protect your system from being turned off by an unauthorized user, assign a password to the `powerdown` command (see "Assigning Special Administrative Passwords" in the "Security" chapter).

From Single-User State

Do not pull the plug until the powerdown procedure is completely finished and the message "System secured for powering down." is displayed.

If your system is in single-user state, turn off your computer by running the `shutdown` command as follows:

```
shutdown -y -i0 -g0
```

where the arguments are defined as:

`-y`	answer yes to all questions asked by `shutdown`
`-i0`	change the system to state 0 (off)
`-g0`	allow a grace period of 0 seconds (for you to log off)

Console messages will appear as shown below.

```
Shutdown started.    Mon Jul  3 12:17:57 EDT 1989

INIT: New run level: 0
The system is coming down.  Please wait.
System services are now being stopped.
The system is down.

NOTICE: System Halt Requested (0)

NOTICE: System secured for powering down.
```

Shortly after the last message in the screen above appears, all system services are stopped.

Rebooting Your System

This procedure halts the system and either reboots from the bootable operating system currently on the hard disk, `/stand/unix`, or configures a new bootable operating system (if necessary) and reboots from the new `/stand/unix`.

This method forces a configuration of a new bootable operating system, if required because of software modifications to the system. (See "Configuring the UNIX Operating System," in the "Performance Management" chapter, for more about configuring a new bootable operating system.) If a new bootable operating system is created, it is written to `/stand/unix`, and the system is rebooted using the new bootable operating system.

To halt and reboot the system from the hard disk, you must be logged in as root. Enter:

```
shutdown -i6
```

A message appears on the console confirming that a shutdown has started. A message is also automatically broadcast that informs users of the shutdown and directs them to log off or risk their files being damaged. The system then waits for a grace period of 60 seconds before asking you if you want to continue with the shutdown; answer `y` (for yes) at this prompt.

The messages shown below assume that no reconfiguration of the operating system is necessary. Should a reconfiguration take place, the messages will look different after the change to the new run level. See "Configuring the UNIX Operating System," in the "Performance Management" chapter.

```
Shutdown started.  Mon Nov 21  10:32:02 EST 1988
Broadcast Message from root (console) Mon Nov 21 10:32:02
THE SYSTEM IS BEING SHUTDOWN NOW ! ! !
Log off now or risk your files being damaged.
Do you want to continue (y or n): y (RETURN)

INIT: New run level: 6
The system is coming down.  Please wait.
System services are now being stopped.
The system is down.
The system is being restarted.

NOTICE: System Reboot Requested
```

At this point, the system is stopped and messages similar to the following are displayed:

```
Copyright Motorola Inc. 1988, 1989, All Rights Reserved

MVME181 Debugger/Diagnostics Release Version 4.0 - 12/07/89

COLD Start
```

The system will proceed to do diagnostics unless the BREAK or some other key is pressed on the console. After the diagnostics are run, messages similar to the following are displayed if the autoboot controller and device have been specified to the firmware:

```
Auto boot in progress ... To abort hit <BREAK>

Booted from VME328, Controller 6, Drive 0

Loaded: Operating System

Volume: $00000000

IPL Loaded at: $00400000

System VR4.0 M88K Boot Loader
```

The system is placed in the state defined by the `initdefault` entry in `/etc/inittab`. If this entry specifies multi-user state, you receive the prompt:

```
Console Login:
```

Note that it may take 5 minutes or longer, depending on your machine model and equipment, to get to the `Console Login:` prompt after the system is reset.

After you receive the `Console Login:` prompt, you may log in to your rebooted system.

Displaying Summary Configuration Information

To print system configuration information, type:

```
prtconf
```

The system will display information about memory and peripheral configuration, such as the information shown in the following example:

```
MOTOROLA M88K SYSTEM CONFIGURATION:

Memory size: 8 Megabytes
System Peripherals:

        Device Name

        UART
        MVME37X
        MVME350
        MVME328
        FPU
```

The device and subdevice names displayed with this command are the Motorola peripheral names found in the Equipped Device Table (EDT) [see `edt_data`(4)].

Displaying System Name and Operating System Release Number

To display your system name and the number of your UNIX operating system release, use the `uname` command with the `-s` and `-r` options. For example:

```
$ uname -sr
UNIX SYSTEM V/88 4.0
```

In this example, the name of the system is UNIX SYSTEM V/88 and the release number of the UNIX system being run is `4.0`. See `uname`(1) for a list of other information you can display and change with the `uname` command.

DRAFT COPY
January 26, 1992
File: machine

Displaying Who Is Logged on to Your Machine

Before taking any action that would affect a system user, check to see who is logged on to the system. The `who` command displays a list of users logged on to your machine, along with the ID, terminal number, and login time of each user. Using the `-H` option supplies you with headers to the information. For example:

```
$ who -H
NAME        LINE         TIME
root        console      Aug 31 08:32
opns        term/13      Aug 30 21:28
mrdh        term/12      Aug 31 09:43
$
```

The `who` command has a number of options that allow you access to more information than the previous example shows. The following list describes some of these options. For a complete listing and a more detailed explanation of each, see `who`(1) in the *User's Reference Manual*.

`-u` On each line, show the number of hours and minutes since activity last occurred. A dot (.) indicates that the terminal has been active in the last minute and is therefore "current." The information is included as an additional field on the default display for the `who` command.

`-T` Show whether someone else can write to that terminal. A plus sign (+) appears if the terminal is writable by anyone; a minus sign (-) appears if it is not. `root` can write to all terminal lines. If a bad line is encountered, a ? is printed.

`-l` Show lines on which the system is waiting for someone to log in.

`-q` Show a "quick" listing, containing only the user login for those who are logged on to the system and the total number of users logged on.

`-b`	Show date and time of the last reboot.
`-r`	Show the current system state of the init process.
`-a`	Run `who` with all options turned on.

Returning from Firmware

You can bring the system back from the firmware mode by executing `unix` from the hard disk.

Step 1 In firmware mode, enter `1` followed by (RETURN) at the following prompt: DS I CO Enter menu #

Step 2 The system will perform diagnostics and boot the operating system. After the sanity of the root file system is checked, file system checks are performed as necessary, the system configuration is printed out, and the system is placed in the operating state defined by the `initdefault` entry in `/etc/inittab`. If this state is the multi-user state, you will eventually observe the prompt:

```
Console Login:
```

Note that it may take 5 minutes or longer, depending on your machine model and equipment, to get to the `Console Login:` prompt.

After you receive the `Console Login:` prompt, you may log in to your system.

Making New Bootable Disks

Each time you turn on your computer, reboot, or manually request a boot from firmware mode, the programs and data files used to boot the computer are read from the hard disk or the tape and loaded into memory for execution. For normal operations, you need to know little more. You know where the programs reside (in `/stand`) and you know how to reconfigure some of these programs (like the bootable operating system) should the configuration of your computer change.

This section shows you how to make additional bootable hard disks, should the need arise. For example, you may want to make another bootable hard disk for development purposes.

After making a new bootable disk, you can bring your machine down to the firmware level (`shutdown -i5`) and request a boot from the new bootable disk.

Making a New Bootable Hard Disk

This procedure should be used by experienced system administrators only. Users unfamiliar with disk partitioning, multi-disk operations, and the operation of the system in both firmware mode and during the boot process should not attempt this procedure.

During the process of installing the system software on your computer, the disk attached to the first hard disk controller is made a bootable disk and becomes the default hard disk for the boot process. This disk has a pathname of the form `/dev/rdsk/m328_c0d2s7`, for the character special file (raw device), and `/dev/dsk/m328_c0d2s7`, for the block special file (block device).

Whenever you power up or reboot your computer, the disk described above is the disk from which the boot programs, associated data files, and the bootable operating system are taken. The computer loads the boot program found in the `boot` partition of the default hard disk. The boot program will then load the bootable operating system found in the boot file system defined on the first `stand` partition on the default hard disk. (In most delivered systems, the default hard disk is the first integral hard disk connected to the first disk controller.)

When the system is in firmware mode, you can request a reboot from any bootable hard disk connected to your system, regardless of which device contains the root file system.

If your computer has more than one hard disk, you can make any disk a bootable disk.

A bootable disk must have at least two partitions defined on it:

- It must have a `boot` partition. This partition holds the `boot` program that loads and executes the bootable operating system.
- It must have a `stand` partition that has a boot file system (`bfs`) defined on it. This boot file system contains all the bootable programs for your computer, as well as all data files needed by these programs. The boot file system is like other file systems in that it can be mounted and unmounted under any directory. By default it is mounted as `/stand`.

You can make any disk connected to your computer bootable, and either boot that disk from firmware, or make it the default boot disk. However, you should be aware that maintaining two or more bootable devices, depending on your intentions, may involve constantly mirroring changes made in one boot file system to other boot file systems. If you make changes to your machine's configuration, such as adding a new hardware or software module, changing tunables, etc., you may want to copy the bootable operating system to all boot file systems, so that each contains a valid bootable operating system for your machine's current configuration.

For example, suppose you have a two-disk system, with boot file systems and boot partitions on both disks (Disk A and Disk B). Assume that the boot file system on Disk A is mounted as `/stand` and the boot file system on Disk B is mounted as `/stand2`. Disk A is the disk booted on powerup and on a `shutdown -i6`. You install a new software driver, which entails changing the `/stand/system` file on Disk A, and reboot the machine.

When the machine reboots, the firmware will detect a difference in the `/stand/system` file on Disk A, and will cause the configuration of a new bootable operating system (`/stand/unix`) on Disk A. After the configuration completes, the machine boots from Disk A.

Now you have two different bootable operating systems on the two disks; the one on Disk A knows about the new software driver you installed, while the one on Disk B does not.

A similar situation arises whenever you make other kinds of changes to the configuration of your system, such as the addition or removal of expansion boards, device drivers, and the like. It is a good idea to copy the bootable operating system to other boot file systems whenever you make changes to the configuration of your system.

In cases like this, request a boot from each defined boot file system on each disk. To do this, request a boot of the `/stand/system` file from firmware and then choose the appropriate disk.

Note that the boot program examines the Volume Table of Contents (VTOC) on each disk to find the boot file system. The boot file system used depends on which disk you request at the firmware prompt (on powerup and reboot, the default boot disk is assumed).

Also note that the file systems mounted when the system comes up in multi-user state are specified in `/etc/vfstab`.

The following procedure shows how to make a hard disk bootable.

Step 1 — If the hard disk you want to make bootable has data on it, perform a full backup of the hard disk. You will want to reload this data after you make the disk bootable. See the "Backup Service" chapter for backup instructions.

Step 2 — Use the `prtvtoc`(1M) command to list the partitions currently on the disk. In order to make the disk bootable, the `stand` partition must exist on the disk and must have a minimum size of 10044 512-byte sectors. The command looks like the following example:

```
prtvtoc /dev/rdsk/prefix_cXdYsZ
```

where the *prefix* is the device type, *X* is the controller number of that device, *Y* is the device number for the controller, and *Z* is the slice number. See `intro`(7) for complete lists of controllers, devices, and slices. See `prtvtoc`(1M) to identify the partitions on the disk from the output.

Step 3 — If Step 2 reveals that the `boot` and `stand` partitions do not exist, or do not meet the minimum sizes specified in Step 2, you must repartition the hard disk.

At a minimum, the disk must be formatted to contain `boot` and `stand` partitions, as well as a partition (partition 7) that contains the whole disk. Other partitions can be defined on remaining space on the disk as needed, including creating additional `stand` partitions.

You can partition a disk using either a restore procedure in the hardware guide for your computer, the `fmthard`(1M) command, or the `sysadm devices` menu.

See the "Storage Device Management" chapter for a chart of default hard disk partitions for various disk drives. If your disk is not in the chart, select the one that most closely matches yours. Then use the values you find in the chart for the `boot` and `stand` partitions to repartition your disk.

Step 4 Make a boot file system on the `stand` partition using `mkfs`(1M). The command looks like the following:

```
mkfs -F bfs /dev/dsk/prefix_cXdYsZ 12000+ [#inodes]
```

where the *prefix* is the device type, `c`*X* is the controller number of that device, `d`*Y* is the device number for the controller, and `s`*Z* is the slice number. See `intro`(7) for complete lists of controllers, devices, and slices. The *#inodes* parameter is optional; it specifies the maximum number of files permitted in the boot file system. If omitted, the system will choose a number of files based on the number of blocks specified. Under most circumstances, this parameter may be omitted. See `mkfs`(1M) and the "File System Administration" chapter for complete information on making a boot file system.

Step 5 Mount the new `stand` partition as `/mnt`:

```
mount -F bfs /dev/dsk/prefix_cXdYsZ /mnt
```

where the *prefix* is the device type, `c`*X* is the controller number of that device, `d`*Y* is the device number for the controller, and `s`*Z* is the slice number. See `intro`(7) for complete lists of controllers, devices, and slices.

Replace the question marks with appropriate controller, drive, and partition numbers, as above.

Step 6 Copy the contents of `/stand` on the old bootable disk to `/mnt`. Use any copy method you like, but the following is recommended:

```
cd /stand
find . -type f -print | cpio -pumv /mnt
```

Note that you may have to raise the allowable file size limit using the `ulimit` shell command [see `sh`(1)] if `/stand/unix` is larger than the current maximum file size limit.

Step 7 Use `umount`(1M) to unmount the boot file system from `/mnt`, as follows:

```
umount /mnt
```

Step 8 Copy the boot programs from the old bootable disk to the new `boot` partition on the new bootable disk using the `dinit`(1M) command, as follows:

```
dinit -b /usr/lib/boot ddeffile
/dev/rdsk/prefix_cXdYsZ
```

The partition number specified must be 7.

Step 9 Make any other file systems desired on remaining disk partitions. If you performed a backup in Step 1, restore data from the backup to a file system on the new bootable disk. See the "Restore Service" chapter for descriptions of restore methods.

Step 10 Create mount points for the new partitions and edit `/etc/vfstab` and `/etc/boot_tab` to match the new file system layout on your machine. These files determine the file systems that are mounted at boot time and where they are mounted. Use `prtvtoc`(1M) to list the partitions/file systems on all the disks, and compare this output to the above files. Then make changes so that the file systems you expect to be mounted at boot time are specified in these files.

You have now defined another bootable disk for your system. Right now, the only way to boot from this new disk is to explicitly request a boot from the disk from firmware.

Quick Reference to Machine Management

- Changing to firmware mode:

  ```
  shutdown -i5
  ```

  shuts the system down. The `-i5` option places you in firmware mode. When asked at what intervals warning messages should be given, answer `n` (for no) to shutdown the system with only a short delay.

- Powering down your machine from multi-user state:

  ```
  shutdown -i0
  ```

  shuts down the system. Before executing `shutdown`(1M), use `who`(1) to check who is logged on and `wall`(1M) to notify those users of your intentions to power down the system.

- Powering down your machine from single-user state:

  ```
  shutdown -y -i0 -g0
  ```

  where the `-y` option assumes "yes" to all questions, `-i0` shuts down the system to state 0 (meaning off), and `-g0` defines the grace period as 0 seconds. You can use `-y` and `-g0` in this case since you are the only user on the machine in single-user state.

- Rebooting your system:

  ```
  shutdown -i6
  ```

  where `-i6` shuts down the system to state 6, meaning stop the system and reboot.

- Displaying summary configuration information:

  ```
  prtconf
  ```

  produces a display which includes memory and peripheral configuration information.

- Displaying system name and operating system release number:

  ```
  uname -sr
  ```

  displays your system name (e.g., `unix`) and UNIX operating system release number.

- Displaying who is logged on to your machine:

  ```
  who -H
  ```

  produces a display of users logged on to your machine and shows the ID, terminal number, and sign-on time of each user. The `-H` option adds field headers to the display.