Solaris Advanced User's Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806–7612–06
December 2001

011025@2471

Contents

Preface

Who Should Read This Book

This book is for advanced users of the Solaris™ 9 operating environment who want to use the command line interface to perform various system tasks.

Before Reading This Book

Your system should be installed and ready for use. If the Solaris operating environment is not installed on your system, see the installation manual specific to your system before you read this book.

Related Books

The Solaris operating environment documentation set provides access to a number of books about the Solaris software. These books are organized into the following related sets:

- *Solaris 9 System Administrator Collection*

  This set offers detailed installation and system administration information for a variety of system configurations, including larger networks of Sun workstations.

- *Solaris 9 Software Developer Collection*

  This set gives software developers the information they need to write, debug, and maintain programs on the system.
- *Solaris 9 Reference Manual Collection*

  This set contains a description of every SunOS command. This collection, often referred to as man pages, can optionally be installed as online documentation.
- *Solaris 9 User Collection*

  This set offers a detailed description of various aspects of the Solaris operating environment. These aspects include:

  - How to use SunOS™ commands
  - How to work with window environments
  - How to customize your work environment
  - How to write shell scripts
  - How to use email
  - How to work on the network

Ordering Sun Documents

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at `http://www1.fatbrain.com/documentation/sun`.

Accessing Sun Documentation Online

The docs.sun.com℠ Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Differences Between Command Line Interface and Graphical User Interface

Desktop software should already be installed on your hard disk or on an accessible server in your network. If you are unsure that you have access to the desktop software, see your system administrator, or refer to the installation manual for your specific platform.

This chapter briefly describes the differences between the command line interface and the desktop environment.

Command Line Interface

A command line interface (CLI) enables users to type commands in a terminal or console window to interact with an operating system. Users respond to a visual prompt by typing a command on a specified line, and receive a response back from the system. Users type a command or series of commands for each task they want to perform.

This book describes how to use the command line interface to perform various system tasks.

Graphical User Interfaces

A graphical user interface (GUI) uses graphics, along with a keyboard and a mouse, to provide an easy-to-use interface to a program. A GUI provides windows, pull-down

menus, buttons, scrollbars, iconic images, wizards, other icons, and the mouse to enable users to interact with the operating system or application.

The Solaris 9 operating environment supports two GUIs, the Common Desktop Environment (CDE) and the GNOME desktop.

Common Desktop Environment

The Common Desktop Environment (CDE) provides windows, workspaces, controls, menus, and the Front Panel to help you organize and manage your work. You can use the CDE GUI to organize your files and directories, read, compose and send email, access files, and manage your system.

For more information, see *Solaris Common Desktop Environment: User's Guide*.

GNOME Desktop

GNOME (GNU Network Object Model Environment) is a GUI and set of computer desktop applications. You can use the GNOME desktop, panel, applications, and tool set to customize your working environment and manage your system tasks. GNOME also provides an application set, including a word processor, a spreadsheet program, a database manager, a presentation tool, a Web browser, and an email program.

CHAPTER 2

Logging In and Using Basic SunOS Commands

This chapter describes the following topics.

- How to log in to your system
- How to log out
- How to enter commands
- How to correct typing mistakes
- How to enter long or multiple commands
- How to use command options
- How to use a password
- How to access documentation about SunOS commands

To enter commands, use a terminal or window. Use your desktop environment documentation if you need information on starting a terminal or window.

Logging In

A *standard work session* is the interval between the time you log in to the system and the time you log out. The SunOS multiuser environment requires that you identify yourself each time you want to use the system. Your *login name* (also known as a *user name* or an *account*) serves as your identity to the system and to other users on the system. Your *password* restricts use of your account to you. If you don't already have a login name and password, ask your *system administrator* to set up an account for you. After you have this information, you are ready to log in.

Before you log in to the system, your screen should look similar to the following:

hostname `console login:`

Type the login name given to you by the system administrator and press the Return key. For example, if your login name is *spanky*, type:

hostname `console login:` **`spanky`**

and press Return. Next, the system requests your password as follows:

hostname `console login:` **`spanky`**
`Password:`

Type your password at the prompt and press Return. If your account does not have a password assigned to it, the system logs you in without asking you for a password. Note that the system does not display (*echo*) your password on the screen as you type it. The failure to display your password helps to prevent others from discovering your password.

Your Login Shell

In the chapters that follow, you enter SunOS commands. When you issue a command to the system, you are actually providing information to a command interpretation program, called a *shell*. The shell program then reads the information you have provided and causes the proper action to occur within the system.

The default shell for SunOS system software is the Bourne shell. The Solaris operating environment also supports the following shells.

- GNU Bourne Again shell (`bash`)
- C shell (`csh`)
- Korn shell (`ksh`)
- TC shell (`tcsh`)
- Z shell (`zsh`)

Each of these shells has its own unique differences.

Note – You can get specific information about any SunOS command, including each of the available shells, by viewing its `man` (manual reference) page. For more information on `man` pages, see "Displaying Manual Pages With `man`" on page 26.

When you initially log in to the system (or open a new terminal or window) and you see your command prompt, it indicates that a shell program has been started for you automatically. This shell is called your *login shell*. If your login shell is not the SunOS default (the Bourne shell), it is because a different shell has been specified for you by your system administrator.

Some commands or procedures available when using one shell might not be available when using another shell. Unless stated otherwise, all commands and procedures described in this manual are available in the Bourne shell.

Logging Out

When you have finished your work session and are ready to exit the operating system, log out by typing the following:

```
$ exit
```

After a moment, the system once again displays the login prompt:

```
$ exit
hostname console login:
```

When you see the login prompt, it indicates that you have successfully logged out. The system is now ready for you or another user to log in.

Note – In the SunOS operating system, turning off your workstation or terminal does *not* necessarily log you out. Unless you log out explicitly, you might remain logged in to the system.

Keyboard Equivalents

You can often speed up operations by using a sequence of keystrokes, called *keyboard accelerators*, that duplicate the operations of the mouse and menus, and of the preconfigured keyboard keys.

The following table lists several command operations and the keyboard equivalents for both SPARC™ and IA machines.

Note – The Meta key is the <> key on SPARC keyboards and is obtained on IA keyboards by pressing Ctrl-Alt.

To use a keyboard accelerator, press and hold the first key (Meta or Ctrl-Alt together) and type the second key. For example, to cut selected text, press and hold the Meta

key and press X on a SPARC system. On an IA system, press and hold Ctrl and Alt together and press X simultaneously.

TABLE 2–1 Keyboard Accelerators

Operation	Keyboard Equivalent	Action
Again	Meta - a	Repeats the previous operation
Copy	Meta - c	Copies the selection to the clipboard
Cut	Meta - x	Cuts the selection and puts it on the clipboard
Find	Meta - f	Finds the selection to the right of the caret
Help	Help or F1	Displays a help window with context-sensitive help for the object at the pointer location
New	Meta - n	Loads a new file
Open (File)	Meta - o	Opens a file (for example, if you've highlighted a file icon in File Manager)
Open (Window)	Meta - w	Opens an icon or closes a window to an icon
Paste	Meta - v	Copies the clipboard selection to the insertion point
Print	Meta - p	Sends the file to the printer (for example, if you've highlighted a file icon in File Manager)
Props	Meta - i	Displays the property window for the application at the pointer location
Redo	Shift-Meta - p	Undoes an Undo
Save	Meta - s	Saves the current file
Stop	Stop or Esc	Stops the current operation
Undo	Meta - u	Undoes the previous operation

Command Prompt

When you log in, the screen or window displays an initial prompt. The appearance of this prompt varies depending on the shell you are using and on how your system administrator originally set it up. Because the default command prompt for SunOS system software is the dollar sign ($), this prompt is used in most of the examples presented in this manual.

If you decide later that you want to change your command prompt, see “Changing Your Command Prompt” on page 134 for instructions.

Typing Commands

When you see the command prompt, the system is waiting for you to type a command. Try typing the command `date` at the prompt, as shown in this example (type `date` and press the Return key):

```
$ date
Mon Sep 17 10:12:51 PST 2001
$
```

This command displays the current date and time. When you type the same command, but capitalized, you receive the following message.

```
$ Date
Date: Command not found.
$
```

The Solaris operating environment interprets an uppercase `D` differently than a lowercase `d`, and the `Date` command fails. Most commands in the Solaris operating environment are lowercase.

Correcting Typing Mistakes

The commands you type are not sent to the system until you press `Return`. If you type a command incorrectly, but do not press `Return`, you can correct your mistake in the following ways.

- Press the Delete or Back Space key to move back a space to the error; or
- Type Ctrl-U to erase the entire line and start over. Hold down the `Control` key and press `u`.

Try both of these methods and see how they work. The Delete/Back Space key varies on some systems. Ctrl-U should work on most systems.

Typing Multiple Commands and Long Commands

You can type more than one command on a single line. Simply place a semicolon (;) between the commands, as shown here with the `date` command and the `logname` command:

```
$ date; logname
Tue Oct 31 15:16:00 MST 2000
spooky
```

This command entry displays the current date and time (from the `date` command) and the login name of the user currently logged in to the system (from the `logname` command).

If you are typing a long command, you can use the backslash character (`\`) to continue typing on a second line. For example:

```
$ date; \
logname
Tue Oct 31 15:17:30 MST 2000
spooky
```

Although the `date` and `logname` commands are not long commands, they demonstrate the concept of continuing a set of commands on the next line. Later, when the commands you want to use are longer than the width of your screen, you will see how useful the backslash character can be.

Note – If you use a desktop window, you might not need to use the backslash character to continue typing commands on the next line. When you reach the end of a line, the commands you type wrap to the next line automatically, and the system executes all commands when you press `Return`.

Repeating Previous Commands

The Korn, Bourne Again, Cl, TC, and Z shells enable your system to keep a *history* of commands you type and are able to repeat previous commands.

Note – The Bourne shell (`sh`) does not support the `history` command.

Repeating Commands in the Bourne Again, C, TC, or Z Shell

If you use the Bourne Again, C, TC, or Z shell, type `!!` and press Return to repeat the last command you typed.

```
example%!!
date
Tue Oct 31 15:18:38 MST 2000
example%
```

You can also repeat any previously typed command by typing `!`*x*, where *x* is the desired command's corresponding number on the *history list*. To see the history list, type the `history` command and press Return. The following is an example of what you might see.

```
example% history
1  pwd
2  clear
3  ls -l
4  cd $HOME
5  logname
6  date
7  history
```

Note – The Z shell does not display the `history` command in the history list.

Another method for repeating characters from the history list is to follow the `!` with a negative number. For example, to repeat the second from the last command on the history list, type the following command.

```
example% !-2
date
Tue Oct 31 15:20:41 MST 2000
example%
```

Note – If you use this command repetition method immediately after the history command in the Z shell, increase the negative number after the `!` by one (`!-3`).

Using the previous example history list, the `date` command is repeated.

You can also type the `!` character, followed by the first few characters of a previous command to repeat that command. For example, if you had previously typed the `clear` command to clear your screen, you could type `!cl` to clear your screen again. With this method for repeating commands, however, you must use enough characters for the desired command to be unique in the history list. If you use only one letter after the `!`, the system repeats the most recent command that begins with that letter.

Repeating Commands in the Korn Shell

If you use the Korn shell, type the following command to repeat the previous command.

```
$ fc -s -
date
Tue Oct 31 15:18:38 MST 2000
$
```

You can also repeat any previously typed command by typing `fc -s` *x*, where *x* is the desired command's corresponding number on the *history list*. To see the history list, type the `fc -l` command and press Return. The following example is a sample history list.

```
$ fc -l
344  pwd
345  clear
346  ls -l
347  cd $HOME
348  logname
349  date
350  history
$
```

You can also repeat commands from the history list by following the `fc -s` command with a negative number. For example, to repeat the second from the last command on the history list, type the following command.

```
$ fc -s -2
date
Tue Oct 31 15:20:41 MST 2000
$
```

Using the previous example history list, the `date` command repeats.

You can also use the `fc -s` command with the first few characters of a previous command. For example, if you had previously typed the `date` command to display the current date, you could type `fc -s da` to display the date again. However, you must use enough characters for the desired command to be unique in the history list. If you use only one letter after the `fc -s` command, the system repeats the most recent command that begins with that letter.

Adding Command Options

Many commands have *options* that invoke special features of the command. For example, the `date` command has the option `-u`, which expresses the date in Greenwich Mean Time instead of local time:

```
$ date -u
Tue Oct 31 22:33:16 GMT 2000
$
```

Most options are expressed as a single character preceded by a dash (-). Not all commands have options. Some commands have more than one option. If you use more than one option for a command, you can either type the options separately (`-a -b`) or together (`-ab`).

Redirecting and Piping Command Output

Unless you indicate otherwise, commands normally display their results on the screen. Some special symbols allow you to *redirect* the output of a command. For example, you might want to save the output to a file rather than display it on the screen. The following example illustrates the use of the redirect symbol (>).

```
$ date > sample.file
$
```

In this example, the output from the `date` command is redirected to a new file called `sample.file`. You can display the contents of `sample.file` by typing the `more` command.

```
$ more sample.file
Tue Oct 31 15:34:45 MST 2000
$
```

As you can see, the contents of `sample.file` now contain the output from the `date` command. See Chapter 3 for information on the `more` command.

Sometimes you might want to redirect the output of one command as input to another command. A set of commands strung together in this way is called a *pipeline*. The symbol for this type of redirection is a vertical bar (|) called a *pipe*.

For example, instead of saving the output of a command to a file, you might want to direct it as input to the command for printing (`lp`) by using the pipe symbol (|). To send the output from the `date` command directly to the printer, type the following:

```
$ date | lp
request id is jetprint-46 (1 file)
$
```

This pipeline would print the results of the `date` command. See “Submitting Print Requests to the Default Printer” on page 105 for information on using the `lp` command to print files.

The command redirection examples shown here are simple, but when you learn more advanced commands, you will find many uses for piping and redirection.

Running Commands in the Background

When you type a command and press the Return key, your system runs the command, waits for the command to complete a task, and then prompts you for another command. However, some commands can take a long time to finish, and you might prefer to type other commands in the meantime. If you want to run additional commands while a previous command runs, you can run a command in the *background*.

If you know you want to run a command in the background, type an ampersand (`&`) after the command as shown in the following example.

```
$ bigjob &
[1] 7493
$
```

The number that follows is the process id. The command `bigjob` will now run in the background, and you can continue to type other commands. After the job completes, you will see a message similar to the following the next time you type another command, such as `date` in the following example.

```
$ date
Tue Oct 31 15:44:59 MST 2000
[1]    Done                 bigjob
$
```

If you plan to log off before a background job completes, use the `nohup` (no hangup) command to enable the job to complete, as shown in the following example. If you do not use the `nohup` command , the background job terminates when you log off.

```
$ nohup bigjob &
[3] 7495
$
```

Using a Password

To ensure your system's security, the Solaris operating environment requires you use a password to access your system. Changing your password several times a year helps to ensure that you are the only user with easy access to your account.

Note – If you believe someone has used your account without your permission, change your password immediately.

When you choose a password, follow these guidelines.

- Choose a password that you can remember without writing it down. A password that you cannot remember is worse than one that is too easily guessed.
- Choose a password that is at least six characters long and contains at least one number.
- Don't use your own name or initials or the name or initials of your spouse.
- Don't use the names of pets or objects common to your interests.
- Don't use all capital letters.
- If you have more than one account, don't use the same password for every account.
- Avoid using the characters Ctrl-C, Ctrl-Z, Ctrl-U, Ctrl-S, Esc, Tab, #, and @ in your password. The terminal might interpret these characters as signals rather than text characters, and this interpretation would preclude you from properly typing in your password.

Changing Your Password

To change your personal password, type the `passwd` command:

```
$ passwd
passwd:  Changing password for user2
Enter login password:
New password:
Re-enter new password:
passwd (SYSTEM): passwd successfully changed for user2
$
```

1. **When the system prompts you for** `Enter login password:`**, type your current password.**

   If no password is currently assigned to your account, the system skips the `Old Password:` prompt.

   The system does not echo (display) your password on the screen and thereby prevents other users from discovering your password.

2. **When the system prompts you for** `New Password:`**, type your new password.**

   Again, the password you type does not echo on the screen.

3. **At the final prompt,** `Re-enter new password:`**, type your new password a second time.**

   Your system verifies that you typed the password you intended to type.

   If you do not type your password precisely the way you did at the previous prompt, the system refuses to change your password and responds with the following message:

```
passwd: They don't match; try again.
```
If you receive this message repeatedly, contact your system administrator to get a new password.

Note – Passwords containing fewer than six characters are not allowed. Also, a new password must differ from the old password by at least three characters.

Password Aging

If your system uses password aging (implemented with options to the `passwd` command), your password can have either a maximum, or a maximum *and* minimum lifespan. The lifespan of your password is set by your system administrator.

When your password reaches the maturity date, your system prompts you to change your password when you log in. The following message displays.

```
Your password has expired. Choose a new one.
```

The system then automatically runs the `passwd` program and prompts you for a new password.

If, for example, the *minimum* age of your password has been set for two weeks, and you try to change your password before the minimum life span has elapsed, the following message displays.

```
Sorry, less than 2 weeks since the last change.
```

For more information on `passwd`(1) and password aging, refer to the *man Pages(1): User Commands*.

Getting Help With OS Commands

This section describes various online help features. These features enable you to view reference information from your workstation or terminal.

Displaying Manual Pages With `man`

If you know the name of a command, but you are not sure what it does, the `man` command can be helpful. Type the following to find out more about this command:

```
$ man man
```

This command displays the first part of a SunOS manual reference page in the window display area. Press the space bar to see the next screen, or press the Q key to quit and return to the command prompt. Use the `man` command to see all the available options and to show the proper command syntax. Manual reference pages often provide examples that illustrate various uses of the command.

Displaying a One-line Summary With `whatis`

If you want just a one-line summary of the command's function, use the `whatis` command, as shown here:

```
$ whatis date
date (1)           -display or set the date
$
```

The number in parentheses after the command name in the previous example indicates the Reference Manual section to which this command belongs. Commands are grouped into various categories according to function. Most user commands are in section 1. By common convention, the section number is displayed in parentheses after the name of the command. You can find the printed manual reference page for a command in alphabetical order within its group.

Note – The `whatis` command is only available if your system administrator has set up a special database of command descriptions.

Keyword Lookup With `apropos`

If you know what you want to do, but you do not know which command to use, use the `apropos` command to locate commands by keyword lookup. The `apropos` command lists all commands that have one-line summaries that contain any keywords you supply. The output of the `apropos` can be lengthy, as some keywords might appear in many places.

Note – The `apropos` command is only available if your system administrator has set up a special database of command descriptions.

To view examples of `apropos` output, type one or more of the following commands.

- `apropos who`
- `apropos execute`
- `apropos apropos`

If you do type a keyword that generates an unreasonably lengthy display, press Ctrl-C to interrupt the display and return you to the command prompt. Hold down the Control key and press "c."

CHAPTER 3

Working with Files and Directories

You can use the SunOS command line to work with, organize, and manage files and directories. You type the file and directory names with SunOS commands to complete specific operations. The command line operates differently than a desktop File Manager. In a File Manager, you can display files as icons that you can click on and move, and you can select commands from menus.

This chapter describes the concepts and procedures you use to work with files and directories from the SunOS command line.

File Concepts

A *file* is the basic unit in the SunOS operating system. Almost everything is treated as a file, including the following items.

- *Documents* – These items include text files, such as letters or reports, computer source code, or any other document that you write and want to save.
- *Commands* – Most commands are *executable* file. That is, they are files you can execute to run a particular program. For example, the `date` command in the previous chapter, which executes a program that provides the current date, is an executable file.
- *Devices* – Your terminal, printer, and disk drive(s) are all treated as files.
- *Directories* – A directory is simply a file that contains other files.

The following section explains the commands for creating, listing, copying, moving, and deleting files. This section also includes information on how to list the contents of a file and how to determine the nature of a file.

Using File Commands

Each of the command descriptions in this section includes an example of how to use the command. Try the examples as you read the text.

Before You Begin

Before you experiment with files, make sure that you are in your *home directory*. Your system administrator established this directory for you at your account creation. To avoid changes to parts of your system that other users expect to remain unchanged, perform the following tasks in your home directory.

To make certain that you are in your home directory, type the `cd` (change directory) command. This command moves you to your home (default) directory. Then type the `pwd` (print working directory) command to display your current location within the file system. The directory displayed is your home directory:

```
$ cd
$ pwd
/export/home/username
```

In this example, the user's home directory is `/export/home/`*username*, where *username* is the name of the user who owns the home directory.

Creating a Test File

Use the `touch` command to create an empty file.

```
$ touch tempfile
$
```

If a file by the name you specify does not exist, the `touch` command creates an empty file.

Note – If the file already exists, `touch` updates the last file access time.

Listing Files (`ls`)

Now list the file with the `ls` command to verify that you've created it:

```
$ ls tempfile
tempfile
```

When you type the `ls` command by itself, it lists all the files in your current location. If you type the `ls` command with a specific file name, it lists only that file, if the file exists.

For more information on listing files, see the man page `ls`(1).

Copying Files (`cp`)

Use the `cp` command to copy `tempfile` to a file called `copyfile`:

```
$ cp tempfile copyfile
$
```

Now list both files. Notice that both names end with the characters "file." You can use the *wildcard* character, asterisk (*), to match any character or sequence of characters. The command `ls *file` lists both `tempfile` and `copyfile`, and any other file in this directory with a name that ends with `file`.

```
$ ls *file
copyfile    tempfile
```

Notice that `copyfile` is listed first. Files are listed in alphabetical order. Capital letters and numbers precede lowercase letters.

For detailed information on copying files, see the man page `cp`(1).

Moving and Renaming Files (`mv`)

You can move and rename files by using the same command, `mv` (move). In this example, use the `mv` command to rename `tempfile` to `emptyfile`:

```
$ mv tempfile emptyfile
$
```

Then list both files again to verify the change.

```
$ ls *file
copyfile    emptyfile
```

`tempfile` is replaced by `emptyfile`.

For more information on moving and renaming files, see the man page `mv`(1).

Deleting Files (rm)

Use the rm (remove) command to delete copyfile, and verify the result with the ls command:

```
$ rm copyfile
$ ls *file
emptyfile
```

Caution – Be careful when you use the rm command, and be particularly careful when you use rm with the wildcard character (*). You cannot recover files that you have removed with rm.

For more information on the rm(1) command, refer to the *man Pages(1): User Commands*.

Displaying File Contents (more, cat)

Use the more command to display the contents of a file. Type more and follow it with the name of the file to be displayed. The contents of the file scroll down the screen. If the file is longer than one screen, this message appears:

```
--More--(nn%)
```

where *nn* is the percentage of the file already displayed.

You can also use the cat command to display the contents of a file, but it displays the file contents rapidly without pausing. The cat (concatenate) command is more often used to join two or more files into one large file, as in this example:

```
$ cat file1 file2 file3 > bigfile
$ ls *file
bigfile
file1
file2
file3
$
```

For further information on the more(1) or cat(1) commands, refer to the *man Pages(1): User Commands*.

Displaying File Type (file)

Some files, such as binary files and executable files, are not printable and cannot be displayed on the screen. Use the file command to show the file type.

```
$ file *
myscript:       executable shell script
print.ps:       PostScript document
save.txt:       ascii text
```

Directories and Hierarchy

This section describes the directory hierarchy that the Solaris operating environment uses to manage and organize files.

Directory Hierarchy

Files are grouped into directories, and directories are organized in a hierarchy. At the top of the hierarchy is the "root" directory, symbolized by "/".

In the following example, Figure 3–1, each directory in the file system contains many subdirectories. The / character distinguishes directory levels with the . The directory / (root) contains the subdirectories `/usr`, `/bin`, `/export/home` and `/lib`, among others subdirectories. The subdirectory `/export/home` contains `user1`, `user2`, and `user3`.

When you initiate commands, you specify directories and files by including the names of the directories that contain the files or directories you want to work with. The names of directories and the files beneath them, combined with slash separators, constitute a *path name*. For example, the path name for the `user3` directory in the following illustration is `/export/home/user3`.

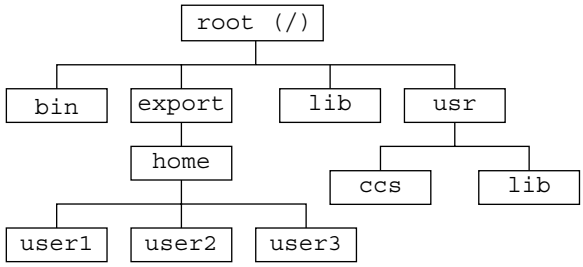

FIGURE 3–1 File System Hierarchy

All subdirectory names and file names within a directory must be unique. However, names within different directories can be the same. For example, the directory `/usr`

contains the subdirectory /usr/lib. No conflict occurs between /usr/lib and /lib because the path names are different.

Path names for files work exactly like path names for directories. The path name of a file describes that file's place within the file-system hierarchy. For example, if the /export/home/user2 directory contains a file called report5, the path name for this file is /export/home/user2/report5. This example shows that the file report5 is within the directory user2, which is within the directory home, which is within the root (/) directory.

Printing the Working Directory (pwd)

The pwd command (print working directory) displays where you are in the file system hierarchy.

```
$ pwd
/home/user1
```

Your output might be different from the example, as your directory structure is different. Your working directory is your current location within the file-system hierarchy.

Your Home Directory

Every user has a *home* directory. When you first open a terminal or a window, your initial location is your home directory. Your system administrator creates your home directory for you at your account creation.

Changing the Working Directory (cd)

The cd (change directory) command enables you to move around within the file-system hierarchy.

```
$ cd /usr/lib
$ pwd
/usr/lib
```

When you type the cd command by itself, you return to your home directory. For example, if your home directory was /export/home/user1:

```
$ cd
$ pwd
/home/user1
```

Although your home directory is `/export/home/user1`, the `cd` command returns you to the`/home/user1` because home directories are mounted on the `/home` directory by the automounter.

In the C, Korn, TC, and Z shells, the tilde (~) is used as a shortcut for specifying your home directory. For example, you would type the following to change to the subdirectory `music` within your home directory:

```
example% cd ~/music
```

You can also use this shortcut to specify another user's home directory. For example:

```
example% cd ~username
```

where *username* is another user's login name, you would change to that user's home directory.

Note – The Bourne and Bourne Again shells do not support the ~ shortcut.

If you use the Bourne shell, your system administrator might have configured the system so that you can type `$home` to specify your home directory. With this configuration, you type the following command to change your working directory to the subdirectory `music` in your home directory.

```
$ cd $home/music
```

The directory immediately "above" a subdirectory is called the *parent directory*. In the preceding example, `/home` is the parent directory of `/home/user1`. The symbol `..` ("dot-dot") represents the parent directory. The command `cd ..` changes the working directory to the parent directory, as in this example:

```
$ pwd
/home/user1
$ cd ..
$ pwd
/home
```

If your current working directory is `/home/user1` and you want to work with some files in `/home/user2`, type the following command.

```
$ pwd
/home/user1
$ cd ../user2
$ pwd
/home/user2
```

`../user2` tells the system to look in the parent directory for `user2`. As you can see, this command is much easier than typing the entire path name `/home/user2`.

Creating a Directory (`mkdir`)

To create a new directory, type the `mkdir` command and follow it with the name of the new directory.

```
$ mkdir veggies
$ cd veggies
$ mkdir broccoli
$ cd broccoli
$ pwd
/home/user2/veggies/broccoli
```

Relative Path Names

The full path name of a directory or a file begins with a slash (`/`) and describes the entire directory structure between that file (or directory) and the root directory. However, you can often use a shorter name that defines the file or directory *relative* to the current working directory.

When you are in a parent directory, you can move to a subdirectory by using only the directory name and not the full path name. In the previous example, the command `cd veggies` uses the relative path name of the directory `veggies`. If the current working directory is `/home/user2`, the full path name of this directory is `/home/user2/veggies`.

Create several different subdirectories, and then move around within this directory structure. Use both full path names and relative path names, and confirm your location with the `pwd` command.

Moving and Renaming Directories

You rename a directory by moving it to a different name. Use the `mv` command to rename directories.

```
$ pwd
/home/user2/veggies
$ ls
broccoli
$ mv broccoli carrots
$ ls
carrots
```

You can also use `mv` to move a directory to a location within another directory.

```
$ pwd
/home/user2/veggies
$ ls
```

```
carrots
$ mv carrots ../veggies2
$ ls ../veggies2
carrots
```

In this example, the directory `carrots` is moved from `veggies` to `veggies2` with the `mv` command.

Copying Directories

Use the `cp -r` command to copy directories and the files they contain:

```
$ cp -r veggies veggies3
$
```

The command in the previous example copies all files and subdirectories within the directory `veggies` to a new directory `veggies3`. This is a *recursive* copy, as designated by the `-r` option. If you attempt to copy a directory without using this option, the system displays an error message.

Removing Directories (`rmdir`)

To remove an empty directory, use the `rmdir` command as follows:

```
$ rmdir veggies3
$
```

If the directory still contains files or subdirectories, the `rmdir` command does not remove the directory.

To remove a directory and all its contents, including any subdirectories and files, use the `rm` command with the recursive option, `-r`.

```
$ rm -r veggies3
$
```

Caution – Directories that are removed with the `rmdir` command *cannot* be recovered, *nor* can directories and their contents removed with the `rm -r` command.

Viewing Differences Between Files (`diff`)

Use the `diff` command to view differences between similar files. The following command scans each line in `leftfile` and `rightfile` to check for differences.

```
$ diff leftfile rightfile
```

When the `diff` utility finds a line or lines that differ, `diff` determines if the difference is the result of an addition, a deletion, or a change to the line, and how many lines are affected. `diff` tells you the respective line number or numbers in each file, followed by the relevant text from each file.

If the difference is the result of an addition, `diff` displays a line with the following format.

l`[,`*l*`] a` *r*`[,`*r*`]`

In the previous example, *l* is a line number in `leftfile` and *r* is a line number in `rightfile`.

If the difference is the result of a deletion, `diff` uses a `d` in place of `a`. If the difference is the result of a change on the line, `diff` uses a `c`.

The relevant text from both files immediately follow the line number information. Text from `leftfile` is preceded by a left angle bracket (`<`). Text from `rightfile` is preceded by a right angle bracket (`>`).

This example shows two sample files, followed by their `diff` output.

```
$ cat sched.7.15
Week of 7/15

Day:  Time:        Action Item:          Details:

T     10:00        Hardware mtg.         every other week
W     1:30         Software mtg.
T     3:00         Docs. mtg.
F     1:00         Interview
$ cat sched.7.22
```

```
Week of 7/22

Day:  Time:        Action Item:          Details:

M     8:30         Staff mtg.            all day
T     10:00        Hardware mtg.         every other week
W     1:30         Software mtg.
T     3:00         Docs. mtg.
$ diff sched.7.15 sched.7.22
1c1
< Week of 7/15
---
> Week of 7/22
4a5
> M     8:30         Staff mtg.            all day
8d8
< F     1:00         Interview
```

If the two files to be compared are identical, `diff` does not display output.

For more information on the `diff`(1) command, refer to the *man Pages(1): User Commands*.

Comparing Three Different Files (`diff3`)

To compare three different versions of a file, use the `diff3` command.

`$ diff3` *file1 file2 file3*

`diff3` compares three versions of a file and publishes the differing ranges of text that are flagged with these codes:

==== all three files differ

====1 *file1* is different

====2 *file2* is different

====3 *file3* is different

Using `bdiff` on Large Files

If you are comparing large files, use `bdiff` instead of `diff`. Use the `diff` command syntax with `bdiff`.

```
$ bdiff leftfile rightfile
```

Use `bdiff` instead of `diff` for files longer than 3500 lines.

Searching for Files (`find`)

The `find` command searches for files that meet conditions you specify, starting from a directory you name. For example, you might search for file names that match a certain pattern or that have been modified within a specified time frame.

Unlike most commands, `find` options are several characters long. You must specify the starting directory before your desired options.

```
$ find directory options
```

In the previous example, *directory* is the name of the starting directory and *options* represents the options for the `find` command.

Each option describes a criterion for selecting a file. A file must meet all criteria to be selected. The more options you apply, the narrower the field becomes. The `-print` option indicates that you want the system to display the results.

The `-name` *filename* option tells `find` to select files that match *filename*. Here *filename* is taken to be the rightmost component of a file's full path name. For example, the rightmost component of the file `/usr/bin/calendar` is `calendar`. This portion of a file's name is often called the *base name*.

For example, to see which files within the current directory and its subdirectories end in `s`, type the following command.

```
$ find . -name '*s' -print
./programs
./programs/graphics
./programs/graphics/gks
./src/gks
$
```

The following table describes other options of the `find` command.

TABLE 3–1 `find` Options

Option	Description
`-name` *filename*	Selects files with a rightmost component that matches *filename*. Surround *filename* with single quotes if it includes filename substitution patterns.
`-user` *userid*	Selects files that are owned by *userid*. *userid* can be either a login name or user ID number.
`-group` *group*	Selects files that belong to *group*.

TABLE 3–1 `find` Options *(Continued)*

Option	Description
`-m -time` *n*	Selects files that have been modified within *n* days.
`-newer` *checkfile*	Selects files that have been modified more recently than *checkfile*.

You can specify an order of options by combining options within escaped parentheses (for example, \(*options*\)). Within escaped parentheses, you can use the `-o` flag between options to indicate that `find` should select files that qualify under either category, rather than just those files that qualify under both categories.

```
$ find . \( -name AAA -o -name BBB \) -print
./AAA
./BBB
```

In the previous example, the `find` command searches in the `.` directory for all files that are named `AAA`, then looks for all files named `BBB`. `find` then displays the results of both searches.

You can invert the sense of an option by including an escaped exclamation point before the option. `find` then selects files for which the option does *not* apply:

```
$ find . \!-name BBB -print
./AAA
```

You can also use `find` to apply commands to the files it selects with the following options.

`-exec` *command* `'{}' \;`

You terminate this option with an escaped semicolon (`\;`). The quoted braces are replaced with the file names that `find` selects.

You can use `find` to automatically remove temporary work files. If you name your temporary files consistently, you can use `find` to search for and remove these files. For example, if you name your temporary files `junk` or `dummy`, this command finds and removes the files.

```
$ find . \( -name junk -o -name dummy \) -exec rm '{}' \;
```

For more information on searching for files, see the man page `find`(1).

File and Directory Security

File permissions help to protect files and directories from unauthorized reading and writing. Often you will have files you want to allow others to read but not change. In other situations, you might want to share executable files or programs. File permissions enable you to control access to your files.

The following list describes the three basic file and directory permission types.

- r – *read* permission. A file must be readable in order for you to examine or copy it. A directory must be readable in order for you to list its contents.
- w – *write* permission. A file must be writable in order for you to modify it, remove it, or rename it. A directory must be writable in order for you to add or delete files in it.
- x – *execute* permission. A file with executable permissions is one you can run, such as a program. A directory must be executable in order for you to gain access to any of its subdirectories.

You can set permissions for three categories of users.

- User – The file owner
- Group – Other users within the same group as the user, such as all staff members of a particular division. The system administrator establishes and maintains groups.
- Others – All users.

Displaying Permissions and Status (`ls -l`)

Use the `-l` with the `ls` command to display a long listing of files and directories in alphabetical order.

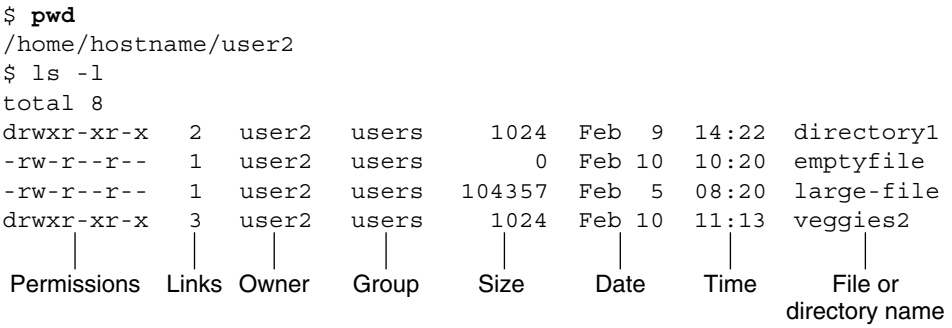

FIGURE 3–2 Displaying Permissions and Status

The first character on the line indicates the file type. A dash (`-`) indicates an ordinary file, a `d` indicates a directory, and other characters can indicate other special file types.

The next nine characters indicate the permissions for the file or directory. The nine characters consist of three groups of three, showing the permissions for the owner, the owner's group, and the world, respectively. The permissions for `emptyfile` are `rw-r--r--`, indicating that the owner can read and write this file, everyone can read it, and no one can execute it. The permissions for the directory `veggies2` are `rwxr-xr-x`, indicating that everyone has read and execute permissions, but only the owner can write to it.

In addition to file permissions, the display shows the following information:

- Number of links to this file or directory
- Name of the owner (`user2` in this case)
- Name of the group owner (`users` in this case)
- Number of bytes (characters) in the file
- Date and time the file or directory was last updated
- Name of the file or directory

  Use the `cd` command to move to your home directory, and try the `ls -l` command.

  Now type the following command, where *dirname* is the name of an actual directory in your file system.

  ```
  $ ls -l dirname
  ```

When you give the name of a directory, the `ls -l` command prints information on all the files and directories in that directory.

Listing Hidden Files (`ls -a`)

Some files are not listed by the `ls` command. These files have names that begin with the character `.` (called "dot"), such as `.cshrc`, `.login` and `.profile`. Use the `ls -a` command to list these dot files:

```
$ ls -a
.
..
.cshrc
.login
.profile
emptyfile
```

Notice that the files beginning with `.` are listed before the other files. The file `.` is the reference for the current directory, and the file `..` is the reference for the parent directory.

In general, system utilities use files that begin with `.` and the user cannot modify these files. Some exceptions to this rule do exist.

Changing Permissions (`chmod`)

Use the `chmod` command to change permissions for a file or directory. You must be the owner of a file or directory, or have root access, to change its permissions. The general form of the `chmod` command is:

`chmod` *permissions name*

In this example, *permissions* indicates the permissions to be changed and *name* is the name of the affected file or directory.

You can specify the permissions in several ways. Here is one of the forms that is easy to use:

1. Use one or more letters to indicate the type of users.
   - `u` (for the *user*)
   - `g` (for *group*)
   - `o` (for *others*)
   - `a` (for *all* three of the previous categories.))
2. Indicate whether the permissions are to be added (`+`) or removed (`-`).
3. Use one or more letters to indicate the permissions.
   - `r` (for *read*)
   - `w` (for *write*)
   - `x` (for *execute*)

In the following example, write permission is added to the directory `carrots` for users who belong to the same group (thus, *permissions* is `g+w` and *name* is `carrots`).

```
$ cd veggies2
$ ls -l
drwxr-xr-x   2 user2    users        512 Nov  1 09:11 carrots
$ chmod g+w carrots
$ ls -l
drwxrwxr-x   2 user2    users        512 Nov  1 09:11 carrots
$
```

The `chmod` **`g+w`** `carrots` command in the previous example gives the group write permission on the file `carrots`. The hyphen (`-`) in the set of permissions for group is changed to a `w`.

To make this same directory unreadable and unexecutable by other users outside your group type the following commands.

```
$ ls -l
drwxrwxr-x   2 user2    users        512 Nov  1 09:11 carrots
$ chmod o-rx carrots
$ ls -l
drwxrwx---   2 user2    users        512 Nov  1 09:11 carrots
$
```

Now, the `r` (for read) and the `x` (for execute) in the set of permissions for other users are both changed to hyphens (`-`).

When you create a new file, the system automatically assigns the following permissions.

```
-rw-r--r--
```

When you create a new directory, the system automatically assigns the following permissions.

```
drwxr-xr-x
```

For example, to make a new file `turnip` executable by its owner (`user2`), type the following command.

```
$ ls -l turnip
-rw-r--r--   1 user2    users        124 Nov  1 09:14 turnip
$ chmod u+x turnip
$ ls -l turnip
-rwxr--r--   1 user2    users        124 Nov  1 09:14 turnip
$
```

If you want to change permissions for all categories of users, use the `-a` option of the `ls` command. To make a new file `garlic` executable by everyone, type the following command.

```
$ ls -l garlic
-rw-r--r--   1 user2    users        704 Nov  1 09:16 garlic
$ chmod a+x garlic
$ ls -l garlic
-rwxr-xr-x   1 user2    users        704 Nov  1 09:16 garlic
$
```

The `x` in the output of the `ls -l` command indicates `garlic` is executable by everyone.

You can also use the `*` wildcard character to change permissions for groups of files and directories. For example, to change the permissions for all the files in the current directory `veggies` so that the files can be written by you alone, type the following command.

```
$ pwd
/home/user2/veggies
$ ls -l
-rwxrwxrwx   1 user2    users       5618 Nov  1 09:18 beets
-rwxrwxrwx   1 user2    users       1777 Nov  1 09:18 corn
-rwxrwxrwx   1 user2    users       3424 Nov  1 09:18 garlic
-rwxrwxrwx   1 user2    users      65536 Nov  1 09:18 onions
$ chmod go-w *
$ ls -l
total 152
-rwxr-xr-x   1 user2    users       5618 Nov  1 09:18 beets
-rwxr-xr-x   1 user2    users       1777 Nov  1 09:18 corn
-rwxr-xr-x   1 user2    users       3424 Nov  1 09:18 garlic
-rwxr-xr-x   1 user2    users      65536 Nov  1 09:18 onions
$
```

Note – Perform this `chmod` operation on the current directory only.

Setting Absolute Permissions

In the previous section, you used the `chmod` command to change file permissions relative to their current settings. You can also set the permissions for a file or directory *absolutely* by using numeric codes with the `chmod` command.

The syntax for this usage of the `chmod` command is:

`chmod` *numcode name*

In this example, *numcode* is the numeric code and *name* is the name of the file or directory for which you are changing permissions.

The complete numeric code consists of three numbers. One number is used for each of the three categories: user, group, and others. For example, the following command sets

absolute read, write, and execute permissions for the user and the group, and execute permissions only for others.

```
$ chmod 771 garlic
```

Table 3–2 illustrates how the the code `771` describes the permissions for `garlic`.

TABLE 3–2 Permissions for `garlic`

Permission	User	Group	Others
Read	4	4	0
Write	2	2	0
Execute	1	1	1
Total	7	7	1

Each of the columns in Table 3–2 represents one of the categories: user, group, and others. To set read permissions, add 4 to the appropriate column. To set write permissions, add 2. To add execute permissions, add 1. The total in all three columns in the last row of the table is the complete numeric code.

The following is another example of using numeric codes to set absolute permissions, with the inclusion of the `ls -l` command to demonstrate the results.

```
$ ls -l onions
-rwxr-xr-x   1 user2    users      65536 Nov  1 09:18 onions
$ chmod 755 onions
$ ls -l onions
-rwxr-xr-x   1 user2    users      65536 Nov  1 09:18 onions
$
```

The **`chmod 755 onions`** command sets the permissions for the file `onions` so that the user can read, write, and execute, group members can read and execute, and others can read and execute. Table 3–3 describes the numeric code that is used to set the permissions for `onions`.

TABLE 3–3 Permissions for `onions`

Permission	User	Group	Others
Read	4	4	4
Write	2	0	0
Execute	1	1	1
Total	7	5	5

To provide read, write, and execute permissions for the file `cabbage` to yourself, your group, and all other users, type the following command.

```
$ ls -l cabbage
-rw-r--r--   1 user2    users         75 Nov  1 09:28 cabbage
$ chmod 777 cabbage
$ ls -l cabbage
-rwxrwxrwx   1 user2    users         75 Nov  1 09:28 cabbage
$
```

Table 3–4 describes the numeric code that is used to set permissions in the previous example.

TABLE 3–4 Permissions for `cabbage`

Permission	User	Group	Others
Read	4	4	4
Write	2	2	2
Execute	1	1	1
Total	7	7	7

The numeric code `777` represents the maximum level of permissions you can provide.

Similar to changing relative permissions, you can also use the wildcard character `*` to set absolute permissions for all in the files in the current directory. For example, suppose you want to set absolute permissions for all files in the current directory as follows:

- Owner – Read, write, and execute permissions
- Group – Read and write permissions
- Others – Execute permissions

To set these permissions, type the following commands.

```
$ pwd
/home/user2/veggies
$ ls -l
-rwxrwxrwx   1 user2    users       5618 Nov  1 09:18 beets
-rwxrwxrwx   1 user2    users       1777 Nov  1 09:18 corn
-rwxrwxrwx   1 user2    users       3424 Nov  1 09:18 garlic
-rwxrwxrwx   1 user2    users      65536 Nov  1 09:18 onions
$ chmod 751 *
$ ls -l
-rwxr-x--x   1 user2    users       5618 Nov  1 09:18 beets
-rwxr-x--x   1 user2    users       1777 Nov  1 09:18 corn
-rwxr-x--x   1 user2    users       3424 Nov  1 09:18 garlic
-rwxr-x--x   1 user2    users      65536 Nov  1 09:18 onions
$
```

The `pwd` command is included in this example to illustrate that the directory on which you perform this operation must be the current directory. The `ls -l` command is shown only to illustrate the changes in permissions. When setting absolute permissions, you do not need to know what the permissions are currently.

For more information on the `chmod`(1) command, refer to the *man Pages(1): User Commands*.

CHAPTER 4

Searching Files

This chapter describes how to search directories and files for keywords and strings by using the `grep` command.

Searching for Patterns With `grep`

To search for a particular character string in a file, use the `grep` command. The basic syntax of the `grep` command is:

```
$ grep string file
```

In this example, *string* is the word or phrase you want to find, and *file* is the file to be searched.

Note – A *string* is one or more characters. A single letter is a string, as is a word or a sentence. Strings can include blank spaces, punctuation, and invisible (control) characters.

For example, to find Edgar Allan Poe's telephone extension, type `grep`, all or part of his name, and the file containing the information:

```
$ grep Poe extensions
Edgar Allan Poe     x72836
$
```

Note that more than one line might match the pattern you give.

```
$ grep Allan extensions
David Allan         x76438
Edgar Allan Poe     x72836
```

```
$ grep Al extensions
Louisa May Alcott   x74236
David Allan         x76438
Edgar Allan Poe     x72836
$
```

`grep` is case sensitive; that is, you must match the pattern with respect to uppercase and lowercase letters:

```
$ grep allan extensions
$ grep Allan extensions
David Allan         x76438
Edgar Allan Poe     x72836
$
```

Note that `grep` failed in the first try because none of the entries began with a lowercase `a`.

`grep` as a Filter

You can use the `grep` command as a filter with other commands, enabling you to filter out unnecessary information from the command output. To use `grep` as a filter, you must pipe the output of the command through `grep`. The symbol for pipe is "|".

The following example displays files that end in "`.ps`" and were created in the month of September.

```
$ ls -l *.ps | grep Sep
```

The first part of this command line produces a list of files ending in `.ps`.

```
ls -l *.ps
```

```
$ ls -l *.ps
-rw-r--r--   1 user2    users     833233 Jun 29 16:22 buttons.ps
-rw-r--r--   1 user2    users      39245 Sep 27 09:38 changes.ps
-rw-r--r--   1 user2    users     608368 Mar  2  2000 clock.ps
-rw-r--r--   1 user2    users     827114 Sep 13 16:49 commands.ps
$
```

The second part of the command line pipes that list through `grep`, looking for the pattern `Sep`.

```
| grep Sep
```

The search provides the following results.

```
$ ls -l *.ps | grep Sep
-rw-r--r--   1 user2    users      39245 Sep 27 09:38 changes.ps
-rw-r--r--   1 user2    users     827114 Sep 13 16:49 commands.ps
$
```

grep With Multiword Strings

To find a pattern that is more than one word long, enclose the string with single or double quotation marks.

```
$ grep "Louisa May" extensions
Louisa May Alcott     x74236
$
```

The grep command can search for a string in groups of files. When it finds a pattern that matches in more than one file, it prints the name of the file, followed by a colon, then the line matching the pattern.

```
$ grep ar *
actors:Humphrey Bogart
alaska:Alaska is the largest state in the United States.
wilde:book.  Books are well written or badly written.
$
```

Searching for Lines Without a Certain String

To search for all the lines of a file that *do not* contain a certain string, use the -v option to grep. The following example shows how to search through all the files in the current directory for lines that do not contain the letter e.

```
$ ls
actors    alaska    hinterland    tutors    wilde
$ grep -v e *
actors:Mon Mar 14 10:00 PST 1936
wilde:That is all.
$
```

Using Regular Expressions With grep

You can also use the grep command to search for targets that are defined as patterns by using *regular expressions*. Regular expressions consist of letters and numbers, in addition to characters with special meaning to grep. These special characters, called *metacharacters*, also have special meaning to the system. When you use regular expressions with the grep command, you need to tell your system to ignore the special meaning of these metacharacters by *escaping* them. When you use a grep regular expression at the command prompt, surround the regular expression with quotes. Escape metacharacters (such as & ! . * $? and \) with a backslash (\). See "Searching for Metacharacters" on page 54 for more information on escaping metacharacters.

- A caret (^) metacharacter indicates the beginning of the line. The following command finds any line in the file list that starts with the letter b.

```
$ grep '^b' list
```

- A dollar-sign ($) metacharacter indicates the end of the line. The following command displays any line in which `b` is the last character on the line.

  ```
  $ grep 'b$' list
  ```

  The following command displays any line in the file `list` where `b` is the *only* character on the line.

  ```
  $ grep '^b$' list
  ```

- Within a regular expression, dot (`.`) finds any single character. The following command matches any three-character string with "an" as the first two characters, including "any," "and," "management," and "plan" (because spaces count, too).

  ```
  $ grep 'an.' list
  ```

- When an asterisk (`*`) follows a character, `grep` interprets the asterisk as "zero or more instances of that character." When the asterisk follows a regular expression, `grep` interprets the asterisk as "zero or more instances of characters matching the pattern."

  Because it includes zero occurrences, the asterisk can create a confusing command output. If you want to find all words with the letters "qu" in them, type the following command.

  ```
  $ grep 'qu*' list
  ```

  However, if you want to find all words containing the letter "n," type the following command.

  ```
  $ grep 'nn*' list
  ```

  If you want to find all words containing the pattern "nn," type the following command.

  ```
  $ grep 'nnn*' list
  ```

- To match zero or more occurrences of *any* character in `list`, type the following command.

  ```
  $ grep .* list
  ```

Searching for Metacharacters

To use the `grep` command to search for metacharacters such as `& ! . * ?` and `\`, precede the metacharacter with a backslash (`\`). The backslash tells `grep` to ignore (*escape*) the metacharacter.

For example, the following expression matches lines that start with a period, and is useful when searching for `nroff` or `troff` formatting requests (which begin with a period).

```
$ grep ^\.
```

Table 4–1 lists common search pattern elements you can use with `grep`.

TABLE 4–1 `grep` Search Pattern Elements

Character	Matches
^	The beginning of a text line
$	The end of a text line
.	Any single character
[...]	Any single character in the bracketed list or range
[^...]	Any character not in the list or range
*	Zero or more occurrences of the preceding character or regular expression
.*	Zero or more occurrences of any single character
\	The escape of special meaning of next character

Note that you can also use these search characters in `vi` text editor searches.

Single or Double Quotes on Command Lines

As shown earlier, you use quotation marks to surround text that you want to be interpreted as one word. For example, type the following to use `grep` to search all files for the phrase “dang it, boys”:

```
$ grep "dang it, boys" *
```

You can also use single quotation marks (`'`) to group multiword phrases into single units. Single quotation marks also make sure that certain that the system interprets metacharacters, such as `$`, literally.

Note – The `history` metacharacter `!` is always interpreted as a metacharacter, even inside quotation marks, unless you escape it with a backslash.

Escape characters such as `& ! $ ? . ;` and `\` when you want `grep` to interpret these characters as ordinary typographical characters.

For example, if you type the following command, the system displays *all* the lines in the file `list`.

```
$ grep $ list
```

However, if you type the following command, the system displays only those lines that contain the "`$`" character.

```
$ grep '\$' list
```

For more information on the `grep`(1) command, refer to the *man Pages(1): User Commands*.

CHAPTER 5

Managing Processes and Disk Usage

This chapter describes how to list the processes that run on your machine, how to kill unwanted processes, and how to display the amount of space that is being used on your disk.

Processes and PIDs

After your system interprets each command, the system creates an independent *process* with a unique process identification number (PID) to perform the command. The system uses the PID to track the current status of each process.

What Commands Are Running Now (`ps`)

Use the `ps` command to see what processes are currently running.The `ps` command shows the *process identification number* (listed under `PID`) for each process you own, which is created after you type a command. This command also shows you the *terminal* from which it was started (`TTY`), the *cpu time* it has used so far (`TIME`), and the *command* it is performing (`COMMAND`).

If you add the `-l` option to the `ps` command, the system displays other process information, including the *state* of each running process (listed under `S`). The following list defines the codes used to describe processes.

- `O` – Process is running on a processor
- `S` – Sleeping: Process is waiting for an event to complete
- `R` - Runnable: Process is on run queue
- `I` - Idle: Process is being create.

- `Z` – Zombie state: Process terminated and parent not waiting
- `T` – Traced: Process stopped by a signal because parent is tracing it
- `X` – SXBRK state: Process is waiting for more primary memory.

Note that while `ps` is running, the status of an individual process can change. Since the `ps` command gives you only a snapshot of what's going on at the moment you issue the command, the output is only accurate for a split second after you type the command.

The `ps`(1) command has more options than those covered here. Refer to the *man Pages(1): User Commands*.

Terminating Processes (`pkill`)

Most window environments have a tool for managing processes. See the tool's online help if you need information on using it.

You can use the `pgrep` and `pkill` commands to identify and stop command processes that you no longer want to run. These commands are useful when you mistakenly start a process that takes a long time to run.

To terminate a process:

1. **Type `pgrep` to find out the PID(s) for the process(es).**
2. **Type `pkill` followed by the PID(s).**

The following example illustrates how to find all the processes with a specific name (`xterm`) and terminate the `xterm` process that was started last.

```
$ pgrep xterm
17818
17828
17758
18210
$ pkill -n 18210
$
```

If you need to forcibly terminate a process, use the `-9` option to the `pkill` command.

```
$ pkill -9 -n xterm
```

Managing Disk Storage

Because space on the disk is a limited resource, make sure that you know the amount of space currently in use.

Displaying Disk Usage (`df -k`)

The `df -k` command displays the amount of space currently in use on each disk mounted (directly accessible) to your system. For the capacity of each disk mounted on your system, the amount of space available, and the percentage of space in use, use the `df -k` command.

```
$ df -k
```

If the output of the `df -k` command indicates your file systems are at or above 90 percent capacity, clear unnecessary files. Clear the files by moving them to a disk or tape, using `cp` to copy files and `mv` to move them, or removing files with the `rm` command. Perform these kinds of "housekeeping" chores only on files that you own.

Displaying Directory Usage (`du`)

You can use `du` to display the usage of a directory and all its subdirectories in units of 512 bytes or characters.

`du` shows you the disk usage in each subdirectory. For a list of subdirectories in a file system, change the directory to the path name that is associated with that file system. Then run the following pipeline:

```
$ du | sort -r -n
```

This pipeline, which uses the *reverse* (`-r`) and *numeric* (`-n`) options of the `sort` command, identifies large directories. Use `ls -l` to examine the size (in bytes) and modification times of files within each directory. Old files and text files larger 100 KBytes might warrant storage offline.

CHAPTER 6

Using the vi Editor

vi (pronounced “vee-eye,” short for visual display editor) is the standard SunOS text editor. vi is not window based and can be used on any kind of terminal to edit a wide range of file types.

You can type and edit text with vi, but it is not a word processor. vi does not process formatted text in the familiar manner of a commercial word processor. To produce formatted printouts, vi relies on a typesetting emulation program, such as nroff, troff, or ditroff. These programs enable you to format vi text by inserting codes that are then interpreted by the emulator.

vi uses a variety of commands, many of which have functions that overlap. This chapter provides an overview of the most essential vi commands. As you begin to use vi, you will find that it is an extremely powerful text editor, and proficiency happens with practice.

Note – The view utility is a read-only version of vi. When you open a file with view, you can use vi commands, but you cannot accidentally change the file by saving your changes.

Starting vi

The following sections describe how to start vi, type text in a file, save (write) the file, and quit vi.

Creating a File

Start `vi` and edit the file `paint` as shown in this example:

```
$ vi paint
```

If `paint` already exists, `vi` opens the existing file. If this is a new file, `vi` creates it. For the purposes of this example, `paint` should be a new file.

The `vi` editing screen appears in a moment:

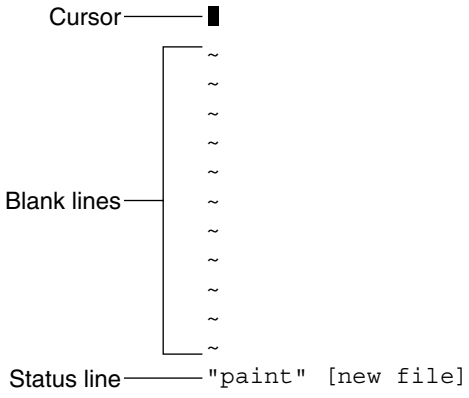

FIGURE 6–1 `vi` Editing Screen

The cursor appears in the upper left corner of the screen. Blank lines are indicated by a vertical series of tildes (~).

Note that you can also start `vi` without specifying a file name by just typing `vi`. You can then name the file later when you exit `vi`.

Status Line

The last line of the screen, called the *status line*, shows the name of the file and the number of lines and characters in the file. When you create a new file, as is the case with the example, the status line indicates that it is a new file.

Two Modes of `vi`

Two modes of operation in `vi` are entry mode and command mode. You use *entry mode* to type text into a file, while *command mode* is used to type commands that perform specific `vi` functions. Command mode is the default mode for `vi`.

Because `vi` doesn't indicate which mode you're currently in, distinguishing between command mode and entry mode is probably the single greatest cause of confusion among new `vi` users. However, if you remember just a few basic concepts from the beginning, you should be able to avoid most of the usual "vi stress."

When you first open a `vi` file, it's always in command mode. Before you can type text in the file, you must type one of the `vi` entry commands, such as `i` ("insert"), to insert text *at* the current cursor location, or `a` ("append"), to insert text *after* the current cursor location. These and other `vi` entry commands are covered in greater detail later in this chapter.

Whenever you want to return `vi` to command mode, press `Esc`. If you're not sure which mode `vi` is presently in, simply press `Esc` to make sure it's in command mode and continue from there. If you press `Esc` while `vi` is already in command mode, the system beeps and the screen flashes, but no harm is done.

Entry Mode

To type text in the sample file `paint`, type the `vi` "insert" command `i`. This command removes `vi` from command mode and puts it into entry mode.

Now type a few short lines of text, ending every line with a Return. Characters you type appear to the left of the cursor and push any existing characters to the right. For the moment, you can correct your mistakes by backspacing and retyping a line before you press Return. For information on editing text in `vi`, see "Changing Text" on page 70.

When you finish typing text in `paint`, press Esc to return to command mode. The cursor moves back onto the last character you typed. Now you can type more `vi` commands.

If `vi` seems to act unpredictably, make sure that you are not in "Caps Lock" mode, which would cause your entries to be all capital letters. On some systems, the F1 key (usually next to the Esc key) acts as the Caps Lock. Pressing this key instead of Esc is a common error.

Note – Occasionally you might need to instruct `vi` to clear or redraw the screen to eliminate, for example, extraneous system messages. To redraw the screen, enter command mode and press Ctrl-L.

Command Mode

When you open a file with `vi`, you are in command mode. In this mode, you can type commands to implement a wide range of functions. Most `vi` commands consist of one or two letters and an optional number. Usually, uppercase and lowercase versions of commands perform related but different functions. As an example, typing `a` appends the file to the right of the cursor, while typing `A` appends the file at the *end* of the line.

Most `vi` commands do not require that you press Return to execute them. Commands beginning with a colon (:), however, do require that you press Return after the command. Some discussions of the `vi` editor refer to commands that are preceded with a colon as a third, and uniquely separate mode of `vi`, *last-line mode*. This mode is so named because when you type the colon while in command mode, the colon and the remainder of what is typed appear on the bottom line of the screen. For the purpose of this discussion, however, all `vi` commands are initiated from command mode.

Commands that are preceded with a colon are actually *ex* commands. `vi` and `ex` are two separate interfaces to the same text-editing program. While `vi` is a screen-oriented interface, `ex` is a line-oriented interface. The full set of `ex` commands is available from within `vi`. When you press the colon, you are actually switching to the line-oriented, `ex` interface. This switch enables you to perform many file manipulation commands without ever leaving `vi`. See "Using `ex` Commands" on page 74, in this chapter, for further information.

Ending a Session

When you edit a file in `vi`, your changes are not made directly to the file. Instead, they are applied to a copy of the file that `vi` creates in a temporary memory space that is called the *buffer*. The permanent disk copy of the file is modified only when you *write* (save) the contents of the buffer.

This arrangement is both positive and negative. A positive feature is that you can quit a file and discard all the changes that you have made during an editing session, leaving the disk copy intact. The negative feature is that you could lose the (unsaved)

contents of the work buffer if the system crashes. People on remote terminals that are connected by phone lines are especially vulnerable to unplanned interruptions.

The best policy is to save your work frequently, especially when you are making substantive changes.

Caution – Although it's possible to run multiple, simultaneous `vi` sessions on one file, it is not a good idea. Great confusion could result when you try to determine which changes have been written to the file and which changes have been overwritten from a simultaneous session.

Saving Changes and Quitting `vi`

`vi` is rich in substantively synonymous commands that control the save of the buffer contents to a file and the exit from `vi`. These commands give you the option of saving, saving-and-quitting, or quitting-without-saving.

Saving

Save the contents of the buffer (write the buffer to the file on disk) by typing:

```
:w
```

Press Return.

Saving and Quitting

Save and quit by typing:

```
:wq
```

Press Return. Alternatively, type `ZZ`.

Note that the command `ZZ` is neither preceded by a colon nor followed by `Return`.

Quitting Without Saving

When you've made no changes to a file and want to quit, type:

```
:q
```

Press Return. If you have made changes, `vi` does not let you quit with `:q`. Instead, it displays the following message.

```
No write since last change (:quit! overrides)
.
```

If you do not want to save your changes, type:

```
:q!
```

Press Return.

Printing a File

After you quit a `vi` file, you can print the file with the following command:

```
$ lp filename
```

In this example, *filename* is the name of the `vi` file to be printed. This command prints the file to your default printer. The file is printed without any formatting, line for line, just as it appears on the screen. See Chapter 8 for more information on printer commands.

Basic `vi` Commands

The following sections explain the following categories of `vi` commands.

- Moving around in a file
- Inserting text
- Changing and substituting text
- Undoing changes to text
- Deleting text
- Checking your spelling
- Formatting your file output
- Repeating commands

Moving Around in a File

In the previous sections you learned how to create, save, print, and exit a `vi` file. Now that you have created a file, you'll need to understand the concepts that are required

to navigate within it. Open your practice file now, and try each of the commands that are discussed in this section.

Moving the Cursor

When you start `vi`, the cursor is in the upper left corner of the `vi` screen. In command mode, you can move the cursor with a number of keyboard commands. Certain letter keys, the arrow keys, and the Return key, Back Space (or Delete) key, and the Space Bar can all be used to move the cursor when you're in command mode.

Note – Most `vi` commands are case sensitive. The same command typed in lowercase and uppercase characters might have different effects.

Moving With Arrow Keys

If your machine is equipped with arrow keys, try these now. You should be able to move the cursor freely about the screen by using combinations of the up, down, right, and left arrow keys. Notice that you can only move the cursor across already existing text or input spaces.

If you're using `vi` from a remote terminal, the arrow keys might not work correctly. The arrow key behavior depends on your terminal emulator. If the arrow keys don't work for you, you can use the following substitutes:

- To move left, press `h`.
- To move right, press `l`.
- To move down, press `j`.
- To move up, press `k`.

Moving One Word

Press `w` ("word") to move the cursor to the right one word at a time.

Press `b` ("back") to move the cursor to the left one word at a time.

Press `W` or `B` to move the cursor past the adjacent punctuation to the next or previous blank space.

Press `e` ("end") to move the cursor to the last character of the current word.

Moving to Start or End of Line

Press ^ to move the cursor to the start of the current line.

Press $ to move the cursor to the end of the current line.

Moving Down One Line

Press the Return key to move the cursor to the beginning of the next line down.

Moving Left

Press the Back Space key to move the cursor one character to the left.

Moving Right

Press the Space Bar to move the cursor one character to the right.

Moving to the Top

Press H (“high”) to move the cursor to the top of the screen.

Moving to the Middle

Press M (“middle”) to move the cursor to the middle of the screen.

Moving to the Bottom

Press L (“low”) to move the cursor to the bottom of the screen.

Paging and Scrolling

If you move down when the cursor is at the bottom of the screen, or move up when the cursor is at the top of the screen, you will see the text scroll up or down. This scrolling can be an effective way to display more text in a short file, but it can be tedious to move this way through a long file.

You can page or scroll backward or forward through a file, a screen or a half-screen at a time. To try out these commands on `paint`, you might want to add text so you have a longer file to work with.

Note that paging and scrolling are fundamentally different. Scrolling actually scrolls the cursor up or down through the text *a line at a time,* as though it were on a paper scroll. Paging moves the cursor up or down through the text *a screenful at a time*. On a fast system, you might not notice the difference. However, if you're working from a remote terminal or in any other situation where your system is running slower than usual, this difference can become painfully apparent.

Page Forward One Screen

To scroll forward (move down) one screenful, press Ctrl-F. (Hold down the Control key and press the F key.) The cursor moves to the upper left corner of the new screen.

Scroll Forward One-Half Screen

To scroll forward one half of a screen, press Ctrl-D.

Page Backward One Screen

To scroll backward (that is., move up) one screenful, press Ctrl-B.

Scroll Backward One-Half Screen

To scroll backward one half of a screen, press Ctrl-U.

Inserting Text

`vi` provides many commands for inserting text. This section introduces you to the most useful of these commands. Note that each of these commands places `vi` in entry mode. To use any of these commands, you must first be in command mode. Remember to press Esc to make sure you are in command mode.

Append

Type `a` (append) to insert text to the *right* of the cursor. Experiment by moving the cursor anywhere on a line and typing `a`, followed by the text you want to add. Press Esc when you're finished.

Type `A` to add text to the *end* of a line. To see how this command works, position the cursor anywhere on a text line and type `A`. The cursor moves to the end of the line, where you can type your additions. Press Esc when you are finished.

Insert

Insert text to the left of the cursor by typing `i` from command mode.

Type `I` to insert text at the beginning of a line. The command moves the cursor from any position on that line. Press Esc to return to command mode after you type the desired text.

Open Line

Use these commands to open new lines, either above or below the current cursor position.

Type `o` to open a line *below* the current cursor position. To experiment, type `o` followed by a bit of text. You can type several lines of text if you like. Press Esc when you are finished.

Type `O` to open a line *above* the current cursor position.

Changing Text

Changing text involves the substitution of one section of text for another. `vi` has several ways to do this, depending on circumstances.

Changing a Word

To replace a word, position the cursor at the beginning of the word to be replaced. Type `cw`, followed by the new word. To finish, press Esc.

To change *part* of a word, place the cursor on the word, to the *right* of the portion to be saved. Type `cw`, type the correction, and press Esc.

Changing a Line

To replace a line, position the cursor anywhere on the line and type `cc`. The line disappears, leaving a blank line for your new text (which can be of any length). Press Esc to finish.

Substituting Character(s)

To substitute one or more characters for the character under the cursor, type `s`, followed by the new text. Press Esc to return to command mode.

Replacing One Character

Use this command to replace the character highlighted by the cursor with another character. Position the cursor over the character and type `r`, followed by just one replacement character. After the substitution, `vi` automatically returns to command mode (you do not need to press Esc).

Undoing Changes

When you edit text and make changes to a `vi` file, you might occasionally wish that you had *not* changed something. `vi`'s undo commands enable you to back up one operation and continue on from there.

Undoing the Previous Command

If you make a mistake in `vi` or if you just change your mind after an operation is completed, you can undo your last command by pressing `u` immediately after the command. You do not need to press Esc after you type `u`. By pressing `u` a *second* time you undo the undo.

Undoing Changes to a Line

Type `U` to undo all changes you've made to a line. This command works only if you haven't moved the cursor off the line. You do not need to press Esc after you type `U`.

Deleting Text

These `vi` commands delete the character, word, or line you indicate. `vi` stays in command mode, so any subsequent text insertions must be preceded by additional commands to enter entry mode.

Deleting One Character

To delete one character, position the cursor over the character to be deleted and type `x`.

The `x` command also deletes the space the character occupied—when a letter is removed from the middle of a word, the remaining letters will close up, leaving no gap. You can also delete blank spaces in a line with the `x` command.

To delete one character before (to the left of) the cursor, type `X` (uppercase).

Deleting a Word or Part of a Word

To delete a word, position the cursor at the beginning of the word and type `dw`. The word and the space it occupied are removed.

To delete part of a word, position the cursor on the word to the *right* of the part to be saved. Type `dw` to delete the rest of the word.

Deleting a Line

To delete a line, position the cursor anywhere on the line and type `dd`. The line and the space it occupied are removed.

Copying and Moving Text — Yank, Delete, and Put

Many word processors allow you to "copy and paste" and "cut and paste" lines of text. The `vi` editor also includes these features. The `vi` command-mode equivalent of "copy and paste" is *yank and put*. The equivalent of "cut and paste" is *delete and put*.

The methods for copying or moving small blocks of text in `vi` involves the use of a combination of the `yank`, `delete`, and `put` commands.

Copying Lines

Copying a line requires two commands: `yy` or `Y` ("yank") and either `p` ("put below") or `P` ("put above"). Note that `Y` does the same thing as `yy`.

To yank one line, follow these steps.

1. Position the cursor anywhere on the line you want to yank.
2. Type `yy`.
3. Move the cursor to the line above where you want to put (copy) the yanked line.

4. Type `p`.

   A copy of the yanked line appears in a new line *below* the cursor.

   To place the yanked line in a new line *above* the cursor, type `P`.

The `yy` command works well with a count: to yank 11 lines, for example, type `11yy`. Eleven lines, counting down from the cursor, are yanked, and `vi` indicates this with a message at the bottom of the screen: `11 lines yanked`.

You can also use the `P` or `p` commands immediately after any of the deletion commands discussed earlier. This action puts the text you deleted above or below the cursor, respectively.

Caution – Use only cursor-moving commands between yanking or deleting and putting. If you delete or yank any other text before putting the new text in place, the lines you yanked or deleted are lost.

Moving Lines

Moving lines also requires two commands: `dd` (“delete”) and either `p` or `P`.

To move one line, position the cursor anywhere on the line and type `dd`. For example, to delete 5 lines, type `5dd`.

Next, move the cursor to the line above where you want the deleted line reinserted and type `p`. This inserts the text on a new line below the cursor.

Alternatively, you can put the deleted line above the cursor by typing `P`.

Using a Count to Repeat Commands

Many `vi` commands can be preceded by a repeat factor (called a *count*)—a number that precedes the command and tells it how many times to repeat the operation.

Most of the commands in the previous sections take counts. For instance, `3dd` repeats the command to delete a line three times, therefore deleting three lines. `2dw` deletes two words, and `4x` deletes four characters or spaces. You can also use counts with commands to move the cursor, such as `3w` and `2Ctrl-F`. In the section “Summary of Basic `vi` Commands” on page 79 each command that takes a count is indicated by “[count]” before the command name.

Typing a period (`.`) repeats the previous text-changing command. For example, if you have just deleted a line with `dd`, you can move the cursor to another line and delete it by simply typing a period.

Using ex Commands

ex commands are more accurate and convenient than yank, delete, and put when you're dealing with large blocks of text. Rather than counting lines on the screen and then searching for an insertion point, you give vi a range of lines to be moved or copied and then specify the line before the insertion point. Of course, with a delete command there is no insertion point.

Turning Line Numbers On and Off

To turn line numbers *on*, type :set nu and press Return.

Line numbers appear in the left margin. Note that these numbers do not appear when you print out the file. They are visible only on the screen.

```
1 Oh, when I die, take my saddle from the wall,
2 Put it on my pony, lead him out of the stall.
3 Tie my bones to his back, point our faces to the west,
4 And we'll ride the prairies that we love the best.
5
6 Ride around, little doggies,
7 Ride around real slow.
8 Firey and Snuffy are rarin' to go.
~
~
~
~
~
~
:set nu
```

To turn line numbers *off*, type :set nonu and press Return.

Copying Lines

The basic form of the ex copy command is:

:*line#*,*line#* **co** *line#*

The first two numbers (separated by a comma) specify the range of lines to be copied. The third number is the line *before* the insertion point.

For example, to copy lines 1 through 5 of paint and place the copy after line 12, you would type the following:

```
:1,5 co 12
```

Then press Return.

When you specify line ranges, use the abbreviations:

- Period (`.`) to denote "from the current line."
- Dollar sign (`$`) to denote "to end of file."

Thus, to copy the range "from the current line through line 5" and insert this block after line 12, you would type:

```
:.,5 co 12
```

To copy the range "from line 6 through the end of the file" and insert this block after line 2, you would type:

```
:6,$ co 2
```

Moving Lines

The basic form of the `ex` move command is similar to the copy command that is discussed in the previous section.

`:`*line#*`,`*line#* **`m`** *line#*

Line ranges and insertion points are specified in the same ways, including use of the abbreviations `.` and `$`. The difference in function is simply that "move" removes a block from one location and reinserts it elsewhere.

For example, to move lines 1 through 5 to the line following 12, you would type:

```
:1,5 m 12
```

Then press Return.

Deleting Lines

To delete a range of lines, use the command form:

`:`*line#*`,`*line#* **`d`**

For example, to delete lines 1 through 5, you would type:

```
:1,5 d
```

Searching and Replacing With vi

vi provides several ways to find your place in a file by locating a specified string of characters. vi also has a powerful global replacement function.

Finding a Character String

A *character string* is one or more characters in succession. A string might include letters, numbers, punctuation, special characters, blank spaces, tabs, or carriage returns. A string can be a grammatical word or it can be part of a word.

To find a character string, type / followed by the string you want to search for, and then press Return. vi positions the cursor at the next occurrence of the string. For example, to find the string "meta," type /meta followed by Return.

Type n to go to the *next* occurrence of the string. Type N to go to the *previous* occurrence.

To search backward in a file, you can use ? instead of /. In this situation, the directions of n and N are reversed.

Searches normally are case sensitive: a search for "china" will not find "China." If you want vi to ignore case during a search, type :set ic. To change it back to the default, case-sensitive mode, type :set noic.

If vi finds the requested string, the cursor stops at its first occurrence. If the string is not found, vi displays Pattern not found on the last line of the screen.

Certain special characters (/ & ! . ^ * $ \ ?) have special significance to the search process and must be "escaped" when they are used in a search. To escape a special character, precede it with a backslash (\). For example, to search for the string "anything?" type /anything\? and press Return.

You can use these special characters as commands to the search function. If you want to search for a string that includes one or more of these characters, you must precede the special character with a backslash. To escape a backslash itself, type \\.

Refining the Search

You can make searches more precise by tagging the string with indicators for the following characteristics:

- Beginning of line
- End of line
- Beginning of word
- End of word
- Wildcard characters

To match the beginning of a line, start the search string with a caret (^). For example, to find the next line beginning with “Search”, type:

```
/^Search
```

To match the end of a line, end the search string with a dollar sign ($). For example, to find the next line ending with “search.”, type:

```
/search\.$
```

Note that the period is escaped with a backslash.

To match the beginning of a word, type \< at the beginning of the string; to match the end of a word, type \> at the end of the string. Thus, to match a word, rather than a string, combine the end-of-word and beginning-of-word tags in the search pattern. For example, to find the next occurrence of the word—as opposed to the string—“search”, type:

```
/\<search\>
```

To match any character, type a period (.) in the string at the location to be matched. For example, to find the next occurrence of “disinformation” or “misinformation,” type:

```
/.isinformation
```

Because this is a search for a string and not a word, this search pattern might also find such constructions as “misinformationalist” and “disinformationism.”

To search for alternative characters in a string, enclose the alternatives in brackets. The search pattern /[md]*string*finds strings that begin with either “m” or “d.” Conversely, /[d-m]*string* finds strings that begin with any letter from “d” through “m.”

To match zero or more occurrences of the last character, type an asterisk (*) in the string. You can effectively combine brackets and the asterisk to look for well-defined alternatives. For example, to find all strings beginning with a through z and ending with “isinformation” *and* to find all occurrences of the string “isinformation”, type:

```
/[a-z]*isinformation
```

Replacing a Character String

The procedure for replacing a text string is based on the search procedures that are discussed previously. You can use all the special matching characters for searches in search-and-replace.

The basic command form is:

:g/*search-string***/s//***replace-string***/g**

Then press the Return key.

Therefore, to replace every occurrence of the string "disinformation" with "newspeak," type:

```
:g/disinformation/s//newspeak/g
```

Then and press Return.

You can modify this command to halt the search and make `vi` query whether you want to make the replacement in each instance. The following command uses `gc` (adding `c` for "consult") to make `vi` stop at every occurrence of "disinformation" and ask whether you want to make the substitution. Respond with `y` for yes or `n` for no.

```
:g/disinformation/s//newspeak/gc
```

Note – You can cancel a "consulted" search-and-replace function by pressing `Ctrl-C`.

Going to a Specific Line

To go to the last line of an open file, type `G`. To return to the first line of the file, type `1G`.

You can go to any other line by typing its number followed by `G`.

For example, suppose that you quit the file `paint` while editing line 51. You can access that line by opening the file and typing `51G`.

Inserting One File Into Another

`vi` makes it convenient to "read" (insert) a file into the file you are editing. The general form of the command is:

: *line#* **r** *filename*

If you do not specify a line number, `vi` inserts the file at the current cursor position.

For example, if you wanted to insert the file `orwell` at line 84 of the file `paint`, you would type:

```
:84 r orwell
```

Or you could position the cursor on line 84 and type:

```
:r orwell
```

Editing Multiple Files

`vi` enables you to edit multiple files. For example, to edit the file `orwell` while you are editing `paint`:

1. **First, save your current work in `paint`. Type `:w` and press Return.**
2. **To edit `orwell`, type `:n orwell` and press Return.**
3. **Make editing changes to `orwell` and save your work.**
4. **When you are finished working with `orwell` and have saved your work, you have three choices:**
   - Exit `vi`. Type `:q` and press Return.
   - Return to `paint`. Type `:n #` and press Return.
   - Swap back and forth between two files with the command `:n #`.

Summary of Basic `vi` Commands

The following table describes basic `vi` commands.

TABLE 6–1 Starting `vi`

Command	Description
`vi` *filename*	Open or create file
`vi`	Open new file to be named later

TABLE 6–1 Starting `vi` *(Continued)*

Command	Description
`vi -r` *filename*	Recover crashed file
`view` *filename*	Open file read-only

TABLE 6–2 Cursor Commands

Command	Description
`h`	Move left one character
`j`	Move down one line
`k`	Move up one line
`l`	Move right one character
`w`	Move right one word
`W`	Move right one word (past punctuation)
`b`	Move left one word
`B`	Move left one word (past punctuation)
`e`	Move to end of current word
`Return`	Move down one line
`Backspace`	Move left one character
`Spacebar`	Move right one character
`H`	Move to top of screen
`M`	Move to middle of screen
`L`	Move to bottom of screen
`Ctrl-F`	Page forward one screen
`Ctrl-D`	Scroll forward one-half screen
`Ctrl-B`	Page backward one screen
`Ctrl-U`	Scroll backward one-half screen

TABLE 6–3 Inserting Characters and Lines

Command	Description
`a`	Insert characters to right of cursor

TABLE 6–3 Inserting Characters and Lines *(Continued)*

Command	Description
`A`	Insert characters at end of line
`i`	Insert characters to left of cursor
`I`	Insert characters at beginning of line
`o`	Insert line below cursor
`O`	Insert line above cursor

TABLE 6–4 Changing Text

Command	Description
`cw`	Change word (or part of word) to right of cursor
`cc`	Change line
`C`	Change from cursor to end of line
`s`	Substitute string for character(s) from cursor forward
`r`	Replace character at cursor with one other character
`r Return`	Break line
`J`	Join current line and line below
`xp`	Transpose character at cursor and character to right
`~`	Change case of letter (uppercase or lowercase)
`u`	Undo previous command
`U`	Undo all changes to current line
`:u`	Undo previous last-line command

TABLE 6–5 Deleting Text

Command	Description
`x`	Delete character at the cursor
`X`	Delete character to the left of the cursor
`dw`	Delete word (or part of word to right of cursor)

TABLE 6–5 Deleting Text *(Continued)*

Command	Description
`dd`	Delete line containing the cursor
`D`	Delete part of line to right of cursor
`dG`	Delete to end of file
`d1G`	Delete from beginning of file to cursor
`:5,10 d`	Delete lines 5-10

TABLE 6–6 Copying and Moving Text

Command	Description
`yy`	Yank or copy line
`Y`	Yank or copy line
`p`	Put yanked or deleted line below current line
`P`	Put yanked or deleted line above current line
`:1,2 co 3`	Copy lines 1-2 and put after line 3
`:4,5 m 6`	Move lines 4-5 and put after line 6

TABLE 6–7 Setting Line Numbers

Command	Description
`:set nu`	Show line numbers
`:set nonu`	Hide line numbers

TABLE 6–8 Setting Case-sensitivity

Command	Description
`:set ic`	Searches should ignore case
`:set noic`	Searches should be case sensitive

TABLE 6–9 Finding a Line

Command	Description
`G`	Go to last line of file
`1G`	Go to first line of file

TABLE 6–9 Finding a Line *(Continued)*

Command	Description
`21G`	Go to line 21

TABLE 6–10 Searching and Replacing

Command	Description
/*string*	Search for *string*
?*string*	Search backward for *string*
`n`	Find next occurrence of *string* in search direction
`N`	Find previous occurrence of *string* in search direction
:g/*search*/s//*replace*/g	Search and replace

TABLE 6–11 Clearing the Screen

Command	Description
`Ctrl-L`	Clear (refresh) scrambled screen

TABLE 6–12 Inserting a File Into a File

Command	Description
`:r` *filename*	Insert (read) file after cursor
`:34 r` *filename*	Insert file after line 34

TABLE 6–13 Saving and Quitting

Command	Description
`:w`	Save changes (write buffer)
`:w` *filename*	Write buffer to named file
`:wq`	Save changes and quit vi
`ZZ`	Save changes and quit vi
`:q!`	Quit without saving changes

CHAPTER 7

Using Mail

SunOS provides a program called `mailx` for sending and receiving electronic mail (*email*). `mailx` provides features for reading, writing, sending, receiving, saving, and deleting messages. The `mailx` program is not window based, and can therefore be run on any terminal. Although you might prefer to use the window-based mail, you might find that the `mailx` program is handy when you are dashing off a quick note. Also, if you want to set up your own mail aliases, you can read about it here.

Note – If you are in a window environment and the Mail Tool icon appears on your screen, quit from the Mail Tool before trying the examples in this chapter. Otherwise, you will have two mail processes active, and these concurrent processes might generate error and warning messages. You can safely send mail messages in a Command Tool or Shell Tool window, but if you read your mail and save or delete messages, this affects your "in tray," thus confusing Mail Tool.

`mailx` Basics

This section describes how to perform basic tasks in `mailx`. Later sections describe features and functions that can enhance your ability to use this program.

An intended recipient's login name and machine name serve as a unique address for the `mailx` program. If the intended recipient is on the same machine as the sender, the login name is all that is required. Each user has a *mailbox* in which to receive mail. This mailbox is generally located in the `/var/mail/`*username* directory, where *username* is your login name.

The `mailx` program notifies you when you receive mail and places the mail in your mailbox. After you read your mail, `mailx` automatically places these letters in a storage file called `mbox`, which is also located in your home directory.

Starting `mailx`

Start `mailx` by typing the following command at a prompt and then pressing the Return key:

```
$ mailx
```

If you don't have any mail waiting for you, your terminal displays the message:

```
No mail for username $
```

In this message, *username* is your login name.

Sending Yourself a Sample Letter

To see at a glance how `mailx` works, you can begin by sending yourself a sample letter. At the prompt, give the `mailx` command again, but this time include your address (your login name plus your machine name). For example, if your login was `rose` and your machine name was `texas`, your address would be `rose@texas`. (The @ symbol is read as "at.") You might be able to use just your login on a local network—consult your system administrator when in doubt.

```
$ mailx rose@texas
```

The program responds with a `Subject:` line:

```
$ mailx rose@texas
Subject:
```

If you like, type a word or two here about the content of the letter you're sending yourself and press Return. Now type the body of the letter. Use short lines and press Return at the end of each line. Note that you can only make corrections as you go by moving back a space and retyping lines *before* you press Return.

Your sample letter might look something like this (the spaces between lines are made by pressing Return twice):

```
$ mailx rose@texas
Subject: to someone who really cares

Dear Rosey,

From the ends of your fingers
To the tip of your nose
```

```
You're a cool breeze in August
My sweet Texas Rose.


See you soon,

Rose
```

To send your sample letter, press Return to complete the last line of the letter and then press `Ctrl-D`. After your letter has been sent, the system returns a command prompt.

Reading Your Sample Letter

To read your sample letter, use the `mailx` command again. Your screen should look something like this:

```
$ mailx
Mail version 4.0 Thu Jan 16 12:59:09 PST 1992  Type ? for help.
"/var/mail/rose": 2 messages 1 new
 U  2 hal@uncertain   Fri Feb 14 12:01   14/318 financial status
>N  1 rose@texas      Mon Feb 17 08:12   21/453 to someone who
&
```

The first line identifies the version of `mail` that you are running. The second line indicates your mailbox, usually located in `/var/mail/`*username*, where your incoming mail is deposited. The third line in this example is the header of the letter you sent yourself. The "N" at the beginning of the line means that it's a "new" letter. A "U" (unread) means the letter was new, but was not read before quitting the `mailx` program previously. The information in this screen is discussed in greater detail in "Reading Letters" on page 88.

Every letter is assigned a number as it is received: Rose's letter to herself is shown as letter number `1`.

To read a letter, type an ampersand (`&`) and the message number at the `mailx` prompt, as follows:

```
& mailx
Mail version 4.0 Thu Jan 16 12:59:09 PST 1992  Type ? for help.
"/var/mail/rose": 1 message 1 new
>N  1 rose@texas   Fri Jul 14 12:01 21/453 to someone who
& 1

To: rose@texas
From: rose@texas
Subject: to someone who really cares

Dear Rose,

From the ends of your fingers
```

```
To the tip of your nose
You're a cool breeze in August
My sweet Texas Rose.


See you soon,

Rose

&
```

Quitting `mailx`

When you have finished your work in `mailx`, you can quit the program by using one of two commands: `q` (quit) or `x` (exit).

If you type `q` at the `mailx` prompt and then press Return, you see a message similar to the following:

`Saved one message in` *home_directory*`/mbox.`

In this message, *home_directory* is the path name to your home directory.

When you use `q` to quit `mailx` after reading messages, `mailx` moves the letters from your mailbox and saves them in the `mbox` file in your home directory. `mailx` also saves any changes or deletions you've made.

If you type `x` at the `mailx` prompt and then press Return, the `mailx` program does *not* save any changes or deletions, nor does it move any letters you've already read into the `mbox` file.

Reading Letters

If you have mail, `mailx` notifies you each time you log in with the message

```
You have mail
```

or

```
You have new mail
```

To read your letters, type `mailx` at a command prompt and press Return. If no mail is waiting for you, you will see the message:

No mail for *username*

Otherwise, you see a list similar to the following:

```
$ mailx
Mail version 4.0 Thu Jan 16 12:59:09 PST 1992  Type ? for help.
"/var/mail/rose": 4 messages 1 new 2 unread
    1 rose@texas        Fri Feb 14 12:01 21/453 to someone who
 U  2 hank@fretful      Fri Feb 14 18:31 19/353 so lonely I
 U  3 farmer@freeway    Sat Feb 15 10:22 24/557 looks like my
>N  4 hoover@woofer     Sun Feb 16 23:59 14/280 big old furry

&
```

The `mailx` program displays information about itself (version number and date) and instructions for getting help (`Type ? for help`).

On the next line, `mailx` specifies the location of your mailbox, the number of letters received, and their status.

Next, `mailx` shows a numbered list of the letters in your mailbox. From left to right, the columns in each line specify:

- *Status*: Indicates whether a letter is new (N), unread (U), or read (no symbol). A ">" at the beginning of a line indicates the current letter. Deleted letters are marked with an asterisk (*).
- *Number*: Indicates the order in which letter was received.
- *Sender*: Indicates the name of the user (and usually machine) the letter came from.
- *Time*: Indicates the date and time the letter was sent.
- *Size*: Indicates the number of lines or the number of characters in letter.
- *Subject*: Indicates the sender-designated subject of the letter.

When you have several letters in your mailbox, the displayed list might not show all of your mail. If this situation applies, type:

- `z` – To display the next screenful of mail headers
- `h-` – To display the previous screenful of mail headers
- `h` – To redisplay the list of mail headers at any time

To view the current letter in the mailbox list (marked with >), press Return. Press Return again to display the next letter. To read any letter in the list, type its number and press Return.

Deleting (and Undeleting) Letters

After you have read a letter, you might decide to delete it rather than save it to your `mbox` file. A save to the `mbox` file is the default when you quit the `mailx` program.

To delete the last letter you read, just type `d` at the `mailx` prompt. To delete a specific letter from your mailbox, use the command form:

`d` *number*

For example, to delete the second letter, use this command from within `mailx`:

```
& d 2
```

You can also delete several letters at a time. To delete letters 1 *and* 3, use the command:

```
& d 1 3
```

To delete a range of letters, for example 1 *through* 3, use the command:

```
& d 1-3
```

Before you quit `mailx`, you can *undelete* letters you've removed from your mailbox. Use the command form:

`u` *number*

followed by Return. For example, to undelete the second letter, use this command:

```
& u 2
```

To undo your last deletion, just type `u` at the `mailx` prompt immediately after the deletion. For example, if your last deletion command was `d 2-5`, type `u` to undelete messages 2, 3, 4, and 5.

Note that all deletions are made permanent when you quit `mailx` with the `q` command. That is, you can no longer undelete letters that have been deleted. You can, however, quit `mailx` with the `x` command and leave your mailbox intact. As mentioned previously, if you quit with `x` you leave read letters marked with a U, deleted letters undeleted, and so forth.

Printing Letters

You can print a hard copy of a letter by piping a letter to a printer command. To do so, use the following command form at a `mailx` prompt:

|*number* `lp`

The | symbol is called a *pipe*. For example, to print a copy of letter 2, type:

```
& |2 lp
```

Then press Return. If a letter number is not specified, `mailx` pipes the current letter to the printer. For more information on piping, see "Redirecting and Piping Command Output" on page 23.

Sending Letters

To send mail with the `mailx` program, you need to know the login name(s) of the intended recipient(s) of your letter. If an intended recipient is on a different machine, you also need to know that user's machine name. To determine this information, you can use the `who`, `finger`, or `rusers` commands.

Type the `who` command to list all the users who are currently logged in to the file server you are on. The displayed list contains users' login names, their terminal types, and the date and time they logged in. For example:

```
$ who
    elmer      tty15        Feb 20 10:22
    susan      tty04        Feb 20 10:37
    stormy     tty07        Feb 20 11:49
    hankw      tty06        Feb 20 12:02
```

Type the `finger` command to display the same type of information as `who` with more detail. The information that appears depends on how your system administrator has set up this command. As an example, you might see something like the following:

```
$ finger
    Login     Name           TTY      Idle     When
    elmer     Elmer Brown    tty15     43      Thu 10:22
    susan     Susan Lake     tty04             Thu 10:37
    stormy    Stormy Ball    tty07     12      Thu 11:49
    hankw     Hank Wilson    tty06     22      Thu 12:02
```

The `rusers` command provides information on the users who are currently logged in to your local network. Refer to Chapter 9 for instructions on the use of the `rusers` command.

When you have determined the necessary user information, complete the following steps to send a letter.

1. **Type the `mailx` command, followed by a user's address:**

   `$` **`mailx`** *user***`@`***machine*

   In this command, *user* is the intended recipient's login name and *machine* is the name of the intended recipient's machine.

   - If you've already started `mailx`, you can just type `m` at the `mailx` prompt, followed by the intended recipient's login and machine name:

     `&` **`m`** *user***`@`***machine*

   - To send the same letter to multiple recipients, separate each address with a space or a comma, for example:

     ```
     $ mailx hank@fretful sally@dakota tex@twister
     ```

     or

     ```
     $ mailx hank@fretful,sally@dakota,tex@twister
     ```

2. **When you press Return, the `mailx` program prompts you for a subject. Type a subject for your letter and press Return again.**

3. **Type the body of your letter. To create a new line, press Return.**

   A sentence that wraps on your screen is not considered a new line until you press Return.

Note – Each line of text within your letter can be up to 256 characters long. When you exceed this limitation, your screen freezes. If this situation occurs, press Ctrl-C to abort your letter.

---

4. **When you have completed your letter, press Return to move the cursor to a new line. Then press Ctrl-D to send your letter.**

Undeliverable Letters

If you specify an incorrect user address when you send a letter, the system responds with the message

user@machine`...User unknown`

The letter is then returned to your mailbox. The next time you type the `mailx` command, the header states that you have returned mail, similar to the following example:

```
N 1 Mailer-Daemon Fri Jan 3 11:13 8/49 Returned mail: User unknown
```

When a letter cannot be delivered, the file is also copied to a file in your home directory named `dead.letter`.

Canceling an Unsent Letter

You can cancel a letter at any time *before* it is sent by pressing Ctrl-C twice.

Adding Carbon and Blind Carbon Copies

Before sending a letter, you can specify that "carbon copies" be sent to other than the main addressees. You can also send "blind carbons." This specification ensures that recipients of your letter can read the addresses for the carbon copies, but not the addresses for the blind carbons.

Many people send themselves carbons or blind carbons in order to retain a copy for their own record.

You can use three methods for sending carbon copies with a letter:

- Use a text editor to edit your `.mailrc` file (in your home directory) and insert the following line:

  `set askcc`

  The `mailx` program now displays the carbon copy prompt (`Cc:`) after the subject prompt. Type the addresses of the users you want to receive carbon copies. Separate multiple addresses with spaces.

- When you have typed the body of your letter, but before you press Ctrl-D, press Return to move to a new line and use the command form:

  `~c` *address(es)*

  To use this method to send carbon copies to multiple recipients, separate the addresses with spaces. For example:

```
~c hank@fretful george@lonesome stormy@snoozer
```

- You can also create a `Cc:` line with the `~h` command, which displays the entire header of the letter. `~h` prompts you with `To:`, `Subject:`, `Cc:`, and `Bcc:` (blind carbon copy) lines, one line at a time. You can fill in blank lines. You can also retype

filled lines. As with other tilde commands, always use the `~h` command on a new line.

Note – `~c`, `~h`, and other tilde commands are described in "Tilde Commands" on page 103.

Inserting a Copy of a Letter or File

You can insert a copy of any letter in your mailbox into the letter you're writing. Likewise, you can insert a copy of any text file.

Inserting a Letter

Use the following command form to insert a letter.

`~m` *number*

In this example, *number* is the number of the letter to be inserted. For example, to send to another user a letter that includes a copy of letter number 3 from your mailbox list, complete the following steps.

1. On a new line, type the command `~m 3`, and then press Return.

   `mailx` displays the following message.

   ```
   Interpolating: 3 (continue)
   ```

2. You do not see the text of message 3, but the recipient will. You can continue to compose your letter after `(continue)`, or you can send it as is.
3. To see the complete letter, interpolation included, type the command `~p`.

Inserting a File

You can also insert a copy of any text file into a letter. As you are writing a letter, use the following command form.

`~r` *filename*

For example, to insert the file `outline` in the current letter, type the following command.

```
& ~r outline
```

Replying to a Letter

Reply to mail at a `mailx` prompt by typing the following command.

`r` *number*

If you omit the letter number, `mailx` replies to the current letter.

For example, to reply to the sender of letter 2, type the following command.

```
& r 2
```

`mailx` automatically addresses your letter and supplies an `Re: Subject:` line that echoes the original Subject: line. Send your reply as you would with any other letter.

`R` is a variant of the reply command that sends your reply to all recipients of the original letter as well as to its sender. Use this command only when absolutely necessary, to avoid generating "junk mail."

Note – You can insert a letter into your reply as shown in the previous section. To insert a copy of the letter to which you are replying, type the command `~m` without a letter number.

Saving and Retrieving Letters

In addition to sending and receiving letters, you might also save and retrieve them for later use. In `mailx` you can save letters by appending them to regular text files. You can also append letters to special files called folders. Both methods are discussed subsequently.

`mailx` makes a distinction between *saving* letters and *copying* them. Saving removes a letter from the mailbox and appends it to a file or folder. Copying leaves a letter in the mailbox and appends a copy to a file or folder.

Saving and Copying Letters in Files

To save a letter into a file, use the following command form at the `mailx` prompt.

`s` *number filename*

where *number* is the number of the letter to be saved and *filename* is the file where you want to save the letter. For example, to save letter 3 into a file called `~/notes/finance`, you would type:

```
& s 3 ~/notes/finance
```

Remember that in a path name, the ~ represents your home directory.

You can also save several letters at once to the same file. For example, to save letters 3, 5, 6, 7, and 8 to `~/notes/finance`, type the following command.

```
& s 3 5-8 ~/notes/finance
```

If the file you specify does not exist, `mailx` creates it. If the file does exist, `mailx` appends the letter you are saving to the end of the file.

Saving a file removes it from your mailbox. `mailx` displays an asterisk (`*`) next to the header of any letter than has been saved.

To leave the letter in your mailbox while appending it to another file, use the `copy` command, as follows:

```
& c 3 ~/notes/finance
```

Saving and Copying Letters in Folders

You can avoid typing full path names to files if you save or copy letters to mail folders. Folders are special files that are stored in a folder directory.

The advantage of saving or copying letters to folders is that your letters are automatically stored in the same directory, where they are easily accessible without typing long path names.

Setting the Folder Directory

To use folders, you must first set up a folder directory. Follow these steps to set up a folder directory.

1. Create a directory by using the `mkdir` command.

   For example, if you wanted your folder directory to be called `Letters`, you would first create the directory:

   ```
   $ mkdir Letters
   ```

2. Edit the `.mailrc` file in your home directory (which contains `mailx` options) to set the folder directory path.

Edit the `folder` variable to include the full path name of your newly created folder directory. For example:

```
set folder=/home/austin/rose/Letters
```

Or use the C shell shortcut ~ to specify your home directory.

```
set folder=~/Letters
```

Now your folder directory is set to receive letters that are saved in folders. The change to the `.mailrc` file becomes effective the next time you start `mailx`.

Designating Folders

You use the same commands to save or copy letters into folders as into files, except that the folder name is preceded by a plus sign (+) instead of a path name. The + tells `mailx` that the folder is to be kept in the folder directory (`Letters`).

For example, to save letter 3 to a folder called `projects`, type:

```
& s 3 +projects
```

`mailx` interprets this command as "save letter 3 into `~/Letters/projects`." If the folder doesn't already exist, `mailx` creates it.

Copy the letter into a folder by typing:

```
& c 3 +projects
```

Sending a Letter Directly to a File or Folder

You can send copies of your letters directly to one of your files or folders. To send a copy to a folder, simply type the folder name in either the Cc: or the Bcc: field. Sending a copy to a file is similar, but you must include the full path name.

Reading Letters in Files and Folders

To read letters that are saved in a file, use the command form:

`mailx -f` *filename*

In the previous example, you would read the file `~/memos/finance` by typing:

```
$ mailx -f ~/memos/finance
```

You can read letters that are saved in a folder with a similar command—just use the + instead of a path name. For example, to read the letters in the folder `projects`, you would type:

```
$ mailx -f +projects
```

This command starts `mailx` in the file or folder that you have designated. Only headers for the letters in the file or folder are displayed. Select a letter to read by typing its number at the `mailx` prompt, and then press Return.

You can also work on mail folders within the `mailx` program. To see a list of your folders, type this command at a `mailx` prompt:

```
& folders
```

To switch from your mailbox to a folder, use the command form:

```
& folder +foldername
```

To return to your mailbox, type this command at a `mailx` prompt:

```
& %
```

To return to the previous folder, type:

```
& #
```

Using `vi` With `mailx`

You can use the `vi` text editor to compose letters while you run `mailx`. `vi` enables you to correct mistakes and add and delete text before you send your letters. If you are not already familiar with using `vi`, refer to Chapter 6 for instructions.

In the `mailx` program, you can use the standard `vi` commands for inserting, deleting, and changing text.

To write a letter with `vi`:

1. **Type the `mailx` command with an address at either the `mailx` prompt (`&`) or the command prompt (`$`).**
2. **Type the subject at the `Subject:` line. Press Return.**
3. **Start `vi` by typing the command `~v` on a new line.**
   The `vi` screen appears, representing an empty file in your `/tmp` directory.
4. **Use `vi` commands to type and edit the body of your letter.**

5. **When you are finished, quit `vi` with the command `:wq` or `ZZ`.**

   After you quit `vi`, `mailx` displays the `(continue):` message. You can either add to the letter (outside `vi`) or send the letter by pressing Ctrl-D.

Mail Aliases

A *mail alias* is a selection of user names that are grouped under a single name.

Use mail aliases when you want to send letters to the same group of people over and over. For example, if you occassionally send mail to `hank@fretful`, `george@lonesome`, and `sally@dakota`, you could create a mail alias called `amigos`. Then, each time you sent mail to `amigos`, all three people would receive it.

Two locations where you can set up mail aliases are:

- Your `.mailrc` file
- The `/etc/aliases` file

Mail aliases that are set up in `.mailrc` behave differently from mail aliases set up in `/etc/aliases`. These differences are summarized in Table 7-1 at the end of this section.

Setting Up Mail Aliases in `.mailrc`

Note the following about setting up mail aliases in `.mailrc`:

- Mail aliases in `.mailrc` are *private*. That is, only you can use them. For example, if you set up a mail alias called `amigos` in `.mailrc` and another user tries to send mail to `amigos`, he receives an `unknown user` error message.
- When the mail is sent, `.mailrc` aliases are automatically expanded to show everyone on the mail alias. For example, if you send mail to `amigos`, your mail arrives as though you had typed everyone's names as recipients. The recipients do not know that you used a mail alias to send the mail.

`.mailrc` is located in your home directory. This file contains several settings that control the behavior of `mailx` and Mail Tool.

To add a mail alias to `.mailrc`, type:

```
$ vi ~/.mailrc
```

Note – You can use any text editor to edit the `.mailrc` file. The previous example shows the method for using the `vi` editor to edit the file. If you are not already familiar with `vi`, refer to Chapter 6 for instructions.

Each mail alias is contained on one line of the file. That is, the line can visually "wrap around" to another line, but it cannot contain carriage returns. Each mail alias should contain the following, separated by spaces:

- The word "`alias`"
- The name of the mail alias (must be one word)
- The recipients (logins and machine names) in the mail alias, separated by spaces

The following example shows two mail aliases. The first alias (`amigos`) contains three people. The second alias (`softball`) contains eight. Notice in `softball` how the names are visually wrapped around on the screen. This is fine, as long as no carriage returns are used.

```
alias amigos hank@fretful george@lonesome sally@dakota
alias softball earl@woofer tex@twister elmer@farmhouse
jane@freeway hank@fretful jj@walker sally@dakota steve@hardway
```

To send mail to people on a `.mailrc` alias, address the mail to the mail alias name. Do *not* include your machine name. Suppose that you sent the following message:

```
$ mail amigos
Subject: Let's eat

Hey Compadres. How about
getting together for lunch on Friday?
Anyone interested?
```

The recipients would see the following (note the expanded `To:` line):

```
To: hank@fretful george@lonesome sally@dakota
Subject: Let's eat

Hey Compadres. How about getting together for lunch on Friday?
Anyone interested?
```

Setting Up Mail Aliases in `/etc/aliases`

Note the following about setting up mail aliases in `/etc/aliases`:

- Mail aliases in `/etc/aliases` are *public*. This means that if you set up a mail alias called `softball`, anyone can send to `softball@`*your-machinename* and make use of the mail alias.

- When the mail is sent, `/etc/aliases` mail aliases are *not* expanded. For example, if you send mail to `softball@`*machinename,* that's how the mail reads when it is received. The recipients know what the mail alias is, but not necessarily who else is on it.

The format of mail aliases that are created in `/etc/aliases` is somewhat different from those in `.mailrc`. Each `/etc/aliases` alias should use the following format:

- The name of the mail alias, followed by a colon (`:`)
- The recipients (logins and machine names), separated by commas. Note that the mail alias does *not* have to be on a single line.

To modify your `/etc/aliases` file, you must first become root. If root is password protected, you'll need to know the root password.

Type the following to become the root user on the system:

```
$ su
Password:
#
```

Notice that the command prompt changes when you become root.

The following example shows how the alias `softball@texas` is added to the default `/etc/aliases` file.

```
# vi /etc/aliases
##
#Aliases can have any mix of upper and lower case on the left-
#hand side,
#but the right-hand side should be proper case (usually lower)
#
#     >>>>>>>>>>The program "newaliases" will need to be run after
#     >> NOTE >>this file is updated for any changes to
#     >>>>>>>>>>show through to sendmail.
#
#@(#)aliases 1.10 89/01/20 SMI
##
# Following alias is required by the mail protocol, RFC 822
# Set it to the address of a HUMAN who deals with this system's
mail problems.
Postmaster: root

# Alias for mailer daemon; returned messages from our MAILER-
DAEMON
# should be routed to our local Postmaster.
MAILER-DAEMON: postmaster

# Aliases to handle mail to programs or files, eg news or vacation
# decode: "|/usr/bin/uudecode"
nobody: /dev/null

# Sample aliases:
```

```
# Alias for distribution list, members specified here:
#staff:wnj,mosher,sam,ecc,mckusick,sklower,olson,rwh@ernie

# Alias for distribution list, members specified elsewhere:
#keyboards: :include:/usr/jfarrell/keyboards.list

# Alias for a person, so they can receive mail by several names:
#epa:eric

########################
# Local aliases below #
########################
softball@texas: earl@woofer
tex@twister elmer@farmhouse
jane@freeway hank@fretful jj@walker sally@dakota steve@hardway
:wq          (to quit vi and save the /etc/aliases file)
# exit       (to exit root)
$
```

You can use any text editor to edit the `/etc/aliases` file. The previous example shows the method for using the `vi` editor to edit the file. If you are not already familiar with `vi`, refer to Chapter 6 for instructions.

Note that the pound signs (#) you see within the `/etc/aliases` file have been placed there to *comment out* the text and sample aliases. The pound signs prevent the system from processing this information as actual aliases.

Do not place pound signs in front of aliases you add to this file, unless you intentionally want to disable an alias.

To send mail to people on a `/etc/aliases` alias, address the mail by using the name of the alias and your machine name. Suppose that you sent the following:

```
$ mail softball@texas
Subject: Practice Today

Let's meet at the diamond
behind Building 4 after work tonight.
Goodness knows we can use the practice for Saturday's game! Be
there as early as you can.
```

The recipients would see the following:

```
To: softball@texas
Subject: Practice Today

Let's meet at the diamond behind Building 4 after work tonight.
Goodness knows we can use the practice for Saturday's game! Be
there as early as you can.
```

Notice that the `To:` line is *not* expanded.

Whenever you send mail by using a mail alias of this type, be sure to include the machine name of the machine on which it's located. If you set up a mail alias called `riders` on the machine `freeway`, then you should send your mail to `riders@freeway`.

Table 7-1 provides a summary comparison between mail aliases that are created in `.mailrc` and those that are created in `/etc/aliases`.

TABLE 7–1 Comparing Mail Aliases in `.mailrc` and `/etc/aliases`

	`.mailrc`	`/etc/aliases`
Must be `root` to modify?	no	yes
Send message to:	*alias*	*alias@machinename*
Recipients list seen by recipients?	yes	no
Names separated by commas?	no	yes
Names all on one line?	yes	no
Others can use the mail alias?	no	yes

For more detailed information on mail aliases, type `man aliases` or `man addresses` at the system prompt.

Tilde Commands

In the course of composing a letter, you can use tilde commands to perform a variety of functions. Tilde commands usually consist of the tilde character (~) followed by a single character. The following table describes some of the more useful tilde characters. Some of these characters have already been introduced in this chapter.

Note – If you want to include a literal tilde character in a letter, type two tildes in succession. Only one tilde is displayed.

TABLE 7–2 Tilde Commands (`mailx`)

Command	Description
~!*command*	Escapes to a shell command.
~.	Simulates pressing Ctrl-D to mark end of file.

TABLE 7–2 Tilde Commands (`mailx`) *(Continued)*

Command	Description
`~?`	Lists a summary of tilde commands.
`~b` *username*	Adds user name(s) to the blind carbon copies (Bcc:) list.
`~c` *username*	Adds user name(s) to the carbon copies (Cc:) list.
`~d`	Reads the contents of the `dead.letter` file into current letter.
`~f` *number*	Forwards the specified letter. Valid only when you send a message while you read mail.
`~h`	Prompts for header lines: Subject, To, Cc, and Bcc.
`~m` *number*	Inserts text from the specified letter into the current letter. Valid only when you send a message while you read mail.
`~p`	Prints to the screen the message that you are typing.
`~q`	Simulates pressing Ctrl-C twice. If the body of the current message is not empty, the contents are saved to `dead.letter`.
`~r` *filename*	Reads in the text from the specified file.
`~s` *string*	Changes the subject line to *string*.
`~t` *name*	Adds the specified name(s) to the To list.
`~w` *filename*	Writes the current letter without the header into the specified file.
`~x`	Exits `mailx`. Similar to ~q except message is not saved in the `dead.letter` file.

Getting Help: Other `mailx` Commands

`mailx` has two help commands that display lists of commands and functions. When in command mode, you can type `?` at the `mailx` prompt (`&`) to see a list of commands used in that mode. Similarly, when in input mode (for example, when writing a letter), you can use the equivalent command, `~?` to view a list of tilde commands (also called "tilde escapes").

The man pages contain extensive information on `mailx` in more technical form. To see this entry, type the command:

```
$ man mailx
```

For more information, refer to the *man Pages(1): User Commands*.

CHAPTER 8

Using Printers

The line printer subsystem (LP) print service provides the printing tools that are used in the Solaris operating environment. LP provides a wide variety of functions, many of which are not within the scope of this manual. This chapter provides the procedures necessary for you to perform the following basic printing tasks with the LP print service:

- Submitting a print request (sending a file to the printer)
- Determining a printer's status
- Cancelling a print request

See *System Administration Guide: Advanced Administration* for a complete description of the LP print service.

Submitting Print Requests

To print a file from the command prompt, use the `lp` command to send a request to the printer to print that file. When a request is made, the LP print service places it in the queue for the printer, displays the request ID number, and then redisplays the shell prompt.

Submitting Print Requests to the Default Printer

When your system administrator sets up the LP print service with a default printer, you can submit print requests without typing the name of the printer by typing the following command.

```
$ lp filename
```

In the previous example, *filename* is the name of the file you want to print.

The specified file is placed in the print queue of the default printer, and the *request id* is displayed.

For example, to print the `/etc/profile` file, type the following command.

```
$ lp /etc/profile
request id is jetprint-1 (1 file)
$
```

See *System Administration Guide: Advanced Administration* for information on how to specify a default printer.

Submitting Print Requests Using a Printer Name

Whether or not a default printer has been designated for your system, you can submit print requests to any printer that is configured for your system. To submit a print request to a specific printer, type the following command.

```
$ lp -d printername filename
```

In the previous example, *printername* is the name of the specific printer and *filename* is the name of the file you want to print.

The specified file is placed in the print queue of the destination printer, and the request id is displayed.

For example, to print the `/etc/profile` file on the printer `fastprint`, type the following command.

```
$ lp -d fastprint /etc/profile
request id is fastprint-1 (1 file(s))
$
```

If you submit a request to a printer that is not configured on your system, an information message is displayed, as shown in the following example.

```
$ lp -d newprint /etc/profile
newprint: unknown printer
$
```

See *System Administration Guide: Advanced Administration* for information on configuring printers. See "Determining Printer Status" on page 108 for information about how to find out which printers are available on your system.

Requesting Notification When Printing Is Complete

When you submit a large file for printing, you might want the LP print service to notify you when printing is complete. You can request that the LP print service notify you in two ways:

- Send an email message
- Write a message to your console window

To request email notification, use the `-m` option when you submit the print request.

```
$ lp -m filename
```

To request a message be written to your console window, use the `-w` option when you submit the print request.

```
$ lp -w filename
```

In the previous example, *filename* is the name of the file you want to print.

Printing Multiple Copies

You can use the `-n` option to the `lp` command to print more than one copy of a file.

Use the following command syntax to request multiple printed copies.

```
$ lp -nnumber filename
```

In the previous example, *number* is the desired number of copies and *filename* is the name of the file you are printing. The print request is considered as one print job, and only one header page is printed.

For example, to print four copies of the `/etc/profile` file, type the following command.

```
$ lp -n4 /etc/profile
request id is jetprint-5 (1 file)
$
```

Summary Table of `lp` Options

You can use options to the `lp` command to customize your print request. Table 8–1 summarizes the frequently used options for the `lp` command. You can use these options individually or combine them in any order on the command line. When you combine options, use a space between each option and repeat the dash (`-`).

For example, to specify a destination printer, request email notification, and print six copies of a file, type the following command.

```
$ lp -d printername -m -n6 filename
```

In the previous example, *printername* is the name of the desired printer and *filename* is the name of the file you want to print.

TABLE 8–1 Summary of Frequently Used `lp` Options

Option	Description
`-d`	Destination. Specifies a destination printer by name.
`-m`	Mail. Sends email to the requestor when the file has printed successfully.
`-n`	Number. Specifies the number of copies to be printed.
`-t`	Title. Specifies a title (printed only on the banner page) for a print request.
`-o nobanner`	Option. Suppresses printing of the banner page for an individual request.
`-w`	Write. Writes a message to your terminal when the file has printed successfully.

See the `lp`(1) man page for a complete list of options.

Determining Printer Status

Use the `lpstat` command to determine the status of the LP print service. You can check the status of your own jobs in the print queue, determine which printers you can use, or determine request ids of your jobs if you want to cancel them.

Checking on the Status of Your Print Requests

Type the following command to find out the status of your print requests.

```
$ lpstat
```

A list of the files that you have submitted for printing is displayed.

In the following example, one file is queued for printing to the printer `jetprint`.

```
$ lpstat
jetprint-1              user2            11466   Nov 01 15:10 on jetprint
$
```

The `lpstat` command displays one line for each print job, showing the request id, followed by the user who spooled the request, the output size in bytes, and the date and time of the request.

Checking Available Printers

To find out which printers are configured on your system, type the following command.

```
$ lpstat -s
```

The status of the scheduler is displayed, followed by the default destination and a list of the systems and printers you can use.

In the following example, the scheduler is running, the default printer is `jetprint`, and the print server for the `jetprint` and `fastprint` printers is `prtsrv1`.

```
$ lpstat -s
scheduler is running
system default destination: jetprint
system for jetprint: prtsrv1
system for fastprint: prtsrv1
$
```

Displaying All Status Information

The `-t` option for `lpstat` gives you a complete listing of the status of the LP print service.

To display a complete listing of all status information, type the following command.

```
$ lpstat -t
```

The system displays all available status information.

In the following example, no jobs are in the print queue. When files are spooled for printing, the status of those print requests is also displayed.

```
$ lpstat -t
scheduler is running
system default destination: jetprint
system for jetprint: prtsrv1
```

```
system for fastprint: prtsrv1

jetprint accepting requests since Wed Nov  1 15:09:29 MST 2000
fastprint accepting requests since Wed Nov  1 15:09:47 MST 2000
printer fastprint is idle. enabled since Wed Nov  1 15:09:46 MST 2000.
jetprint-1              user2            11466   Nov 01 15:10 on jetprint
$
```

Displaying Status for Printers

You can use the `-p` to the `lpstat` command to request printer status information for individual printers. This option shows whether the printer is active or idle, when the printer was enabled or disabled, and whether the printer is available to accept print requests.

To request status for all printers on a system, type the following command.

```
$ lpstat -p
```

In the following example, two printers are idle, enabled, and available.

```
$ lpstat -p
printer jetprint is idle. enabled since Wed Nov  1 15:09:28 MST 2000.
   available.
printer fastprint is idle. enabled since Wed Nov  1 15:09:46 MST 2000.
   available.
$
```

If one of those printers had jobs in the print queue, those jobs would also be displayed.

To request status for an individual printer by name, type the following command.

```
$ lpstat -p printername
```

In this example, *printername* is the name of the specific printer.

Summary Table of `lpstat` Options

You can use the `lpstat` command to request different types of printing status information. Table 8–2 summarizes the frequently used options for the `lpstat` command. Use these options individually, or combine them in any order on the command line. When you combine options, use a space between each option and repeat the dash (`-`).

TABLE 8–2 Summary of Frequently Used `lpstat` Options

Option	Description
`-a`	Accept. Show whether print destinations are accepting requests.
`-c`	Class. Show classes and their members.
`-d`	Destination. Show default destination.
`-f`	Forms. Show forms.
`-o`	Output. Show status of print requests.
`-p [`*list*`] [-D] [-l}`	Printer/Description/Long list. Show status of printers.
`-r`	Request. Request scheduler status.
`-R`	Show position of job in the queue.
`-s`	Status. Show status summary.
`-S`	Sets. Show character sets.
`-u [`*username*`]`	User. Show requests by user.
`-v`	Show devices.

See the `lpstat`(1) man page for a complete list of options.

Canceling a Print Request

Use the `cancel` command to cancel a print request while it is in the queue or while it is printing. To cancel a request, you need to know the print request id. The request id always includes the name of the printer, a dash, and the number of the print request. When you submit the print request, the request id is displayed. If you do not remember your request id, type `lpstat -o` and press Return. Only the user who submitted the request, or someone logged in as `root` or `lp` can cancel a print request.

Canceling a Print Request by ID Number

To cancel a print request, type the following command.

`$` **`cancel`** *request_id*

In the previous example, *request_id* is the request id number of the desired print request.

A message informs you that the request is cancelled. The next job in the queue begins printing.

In the following example two print requests are canceled.

```
$ cancel jetprint-6 fastprint-5
jetprint-6: cancelled
fastprint-5: cancelled
$
```

Canceling a Print Request by Printer Name

You can also cancel just the job that is currently printing (if you submitted it) by typing the printer name in place of the request id.

```
$ cancel printername
```

In the previous example, *printername* is the name of the printer to which you sent the request.

A message informs you that the request is canceled. The next job in the queue begins printing

In the following example, the current print request has been canceled.

```
$ cancel jetprint
jetprint7: cancelled
$
```

Your system administrator can log in as `root` or `lp` and cancel the currently printing request by using the printer name as the argument for the `cancel` command.

CHAPTER 9

Using the Network

A *network* is a group of computers set up to communicate with one another. When your machine is part of a network, you have the opportunity to use the resources of other machines on the network while remaining logged in to your own machine. You can log in to other machines or you can execute remote commands that affect other machines from your own workstation.

This chapter describes the general concepts of networking, and provides information on the following tasks.

- How to log in to remote machines
- How to copy files from remote machines
- How to execute commands on remote machines
- How to request status information on remote machines
- How to test connections between networked machines
- How to run networked applications

If your machine is not currently attached to a network, the information presented here might not be relevant to your situation. However, it might be valuable for you to review this information to get an overall view of the benefits that networking can provide.

Networking Concepts

A network connection between machines enables these systems to transmit information to one another. The following list describes the more common network types.

- **Local area network** (LAN) — A network that ranges over a small area, generally less than a few thousand feet

- **Wide area network** (WAN) — A large network that can span thousands of miles
- **Campus area network** (CAN) — An intermediately sized network

A network that is composed of a linked group of networks is called an *internetwork*. For example, your machine might be part of a network within your building and part of an internetwork that connects your local network to similar networks across the country. Because the difference between a network and an internetwork is generally invisible to the user, the term *network* is used in this guide to refer to both networks and internetworks.

Networked machines use a *network protocol*, or common network language, to communicate. A network protocol ensures that information is transmitted to the appropriate location on the network. An *internetwork protocol*, sometimes referred to as a *relay*, links networks together.

Logging In Remotely (`rlogin`)

The `rlogin` command enables you to log in to other UNIX machines on your network.

To remotely log in to another machine, type the following command.

`$` **`rlogin`** *machinename*

In the previous example, *machinename* is the name of the remote machine.

If a password prompt appears, type the password for the remote machine and press `Return`. If your machine name is in the other machine's `/etc/hosts.equiv` file, the current machine does not prompt you to type the password.

```
venus$ rlogin starbug -l user2
Password:
Last login: Wed Nov  1 13:08:36 from venus
Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
venus$ pwd
/home/user2
venus$ logout
Connection closed.
venus$
```

`rlogin` Without a Home Directory

In the previous example, user `user2` logs in to the remote machine `starbug` at the directory `/home/user2`, as indicated by the `pwd` command. When you log in to a machine where you have no account, `rlogin` displays a message stating that you have no home directory on the remote machine. `rlogin` then logs you in to the root (`/`) directory of that machine:

```
venus$ earth -l user2
Password:
No directory! Logging in with home=/
Last login: Thu Nov  2 12:51:57 from venus
Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
earth$ pwd
/
earth$ logout
Connection closed.
earth$
```

`rlogin` as Someone Else

The `-l` option of the `rlogin` command enables to log in to a remote machine as someone else. This option can be useful if you are working on someone else's machine (using their username) and you want to log in to your own machine as yourself.

Use the following command syntax for the `-l` option of the `rlogin` command.

```
# rlogin machinename -l username
```

The following example shows how user `user2` on machine `venus` would log in to machine `starbug` as `user1`.

```
venus$ rlogin starbug -l user1
Password:
Last login: Thu Nov  2 12:51:57 from venus
Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
starbug$ pwd
/home/user1
starbug$ logout
Connection closed.
starbug$
```

Note that when you log in to a remote machine as someone else, you are placed in that user's home directory.

`rlogin` to an Unknown Machine

If you try to log in to a remote machine unknown to your machine, `rlogin` searches unsuccessfully through the hosts database and then displays the following message.

```
$ rlogin stranger
stranger: unknown host
$
```

Aborting an `rlogin` Connection

Normally you terminate an `rlogin` connection by typing `logout` at the end of a work session. If cannot terminate a session in this manner, you can abort the connection by typing a tilde character followed by a period (`~.`). The remote login connection to the remote machine is aborted and you are placed back at your original machine.

If you log in to a series of machines, gaining access to each machine through another machine, and you use `~.` to abort the connection to any of the machines, you are returned to your original machine.

```
venus$ rlogin starbug -l user2
Password:
Last login: Thu Nov  2 15:13:10 from venus
Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
starbug$ ~.  (You may not see the ~ on the screen.)
Closed connection.
venus$
```

If you want to back up to an intermediate rlogin connection, use two tildes followed by a period (`~~.`).

```
venus$ rlogin starbug -l user2
Password:
Last login: Thu Nov  2 15:14:58 from venus
Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
starbug$ rlogin rlogin earth -l user2
Password:
Last login: Thu Nov  2 15:24:23 from starbug
Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
earth$ ~~.  (You may not see the ~~ on the screen.)
Closed connection.
starbug$
```

Suspending an `rlogin` Connection

When you want to suspend an `rlogin` connection so you can return to it later, type the tilde character (`~`) followed by Ctrl-Z. The `rlogin` connection becomes a stopped process and you are placed back at the machine from which you logged in.

To reactivate the connection, type `fg`. You can also type the percentage sign (`%`) followed by the process number of the stopped process. If you do not specify a process number, `%` activates the process most recently suspended.

```
venus$ rlogin goddess -l user2
Password:
Last login: Thu Aug 31 14:31:42 from venus
Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
goddess$ pwd
/home/user2
goddess$~^Z

Stopped (user)
venus$ pwd
/home/user2/veggies
venus$fg
rlogin goddess
goddess$ logout
venus$
```

You can also type two tildes and press Ctrl-Z to suspend the current `rlogin` and place you at an intermediate `rlogin`.

For more information on the `rlogin`(1) command, refer to the *man Pages(1): User Commands*.

Verifying Your Location (`who am i`)

After logging in to a variety of remote machines, you might need to verify where you are. Type `who am i` to display the name of the machine you are currently logged in to as well as your current login name.

Copying Files Remotely (`rcp`)

The `rcp` command enables you to copy files from one machine to another. This command uses the remote machine's `/etc/hosts.equiv` and `/etc/passwd` files to

determine whether you have unchallenged access privileges. The syntax for `rcp` is similar to the command syntax for `cp`.

Note – To copy subdirectories and their contents from one machine to another, use `rcp -r`.

Copying Files From a Remote Machine

To copy from a remote machine to your machine, use the `rcp` command with the following syntax.

`$ rcp` *machinename*`:`*source destination*

In the previous example, *machinename* is the name of the remote machine, *source* is the name of the file(s) you want to copy, and *destination* is the path name on your machine where you want the copied file(s) to reside.

The following example illustrates how to copy the file `/etc/hosts` from the remote machine `starbug` to the `/tmp` directory on the local machine `venus`.

```
venus$ rcp starbug:/etc/hosts /tmp
```

You can also combine various abbreviations and syntaxes when using `rcp`. For example, to copy all of the files ending in `.doc` from user hank's home directory on remote machine `fretful` to the current directory on local machine `venus`, type the following command.

```
venus$ rcp fretful:~hank/*.doc .
venus$
```

Copying Files From Your Machine to Another

To copy files from your local machine to a remote machine, use the following command syntax.

`$ rcp` *source machinename*`:`*destination*

In the previous example, *source* is the file(s) you want to copy, *machinename* is the name of the remote machine, and *destination* is the path name on the remote machine where you want the copied file(s) to reside.

The following example illustrates how to copy the file `newhosts` from your `/tmp` directory to the `/tmp` directory on the remote machine `starbug`.

```
venus$ cd /tmp
venus$ rcp newhosts starbug:/tmp
```

```
venus$
```

For more information on the `rcp`(1) command and its options, refer to the *man Pages(1): User Commands*.

Executing Commands Remotely (`rsh`)

The `rsh` command (for *remote shell*) enables you to execute a single command on a remote machine without having to log in to the remote machine. If you know you only want to do one thing on a remote machine, `rsh` enables you to quickly execute one command on a remote machine.

To execute a command on a remote machine, use the following command syntax.

`rsh` *machinename command*

The following example shows how you would view the contents of the directory `/etc/skel` on the machine `starbug`.

```
venus$ rsh starbug ls /etc/skel*
local.cshrc
local.login
local.profile
venus$
```

Similar to the `rlogin` and `rcp` commands, `rsh` uses the remote machine's `/etc/hosts.equiv` and `/etc/passwd` files to determine whether you have unchallenged access privileges to the remote machine.

For more information on the `rsh`(1) command and its options, refer to the *man Pages(1): User Commands*.

Viewing User Information (`rusers`)

The `rusers` command (for *remote users*) shows you which users are logged in to other machines on your network. Type the `rusers` command by itself to display each machine on the network and the user(s) logged in to these machines.

```
$ rusers
starbug          user2 user1 root
earth            user1
```

```
venus                   user3
```

Notice that three users are logged in to the machine `starbug`.

To display information on a specific remote machine, type the `rusers` command followed by the name of the machine.

```
$ rusers starbug
starbug          user2 user1 root
$
```

The `-l` option to the `rusers` command displays the following information.

- User names
- Machine and terminal names
- Time each user logged in
- How long each user has been idle (if more than one minute)
- Name of the machine that each user logged in from

```
$ rusers -l starbug
root       starbug:console       Oct 31 11:19         (:0)
user2      starbug:pts/7         Oct 31 11:20   40:05 (starbug)
user1      starbug:pts/13        Nov  1 14:42   17:18 (starbug)
$
```

You can also use the `-l` option without providing a machine name.

For more information on the `rusers`(1) command and its options, refer to the *man Pages(1): User Commands*.

Running Networked Applications

Normally, all the applications on your system are programs that are running on your local machine. However, if your workstation is part of a network, you can run applications on another machine and display them on your local screen. Running an application on another machine saves computing cycles on your local machine, and gives you access to an entire network of applications.

This section describes how to run an application on a remote machine and display it on your local screen. See “More About Security” on page 122 for additional information on the complexities of running networked applications.

Note – The process for running a networked application in your computing environment might vary.

Using `rlogin` to Run a Networked Application

To use the following procedure to run a remote application, you must meet these requirements.

- You must have access rights to the remote machine.
- Your home directory must be NFS-mountable on the remote machine.
- The application and appropriate libraries must be installed on the remote machine.

Contact your system administrator if you do not understand these requirements.

To run a networked application on a remote machine, set your environment variables with the following values.

- Set the `DISPLAY` environment variable in your shell on the remote machine to your local screen.
- Enable the remote machine to display the application program on your local machine. To enable this display access on the local machine, use the `xhost` command:

  `$` **`xhost +`***`remote_machine_name`*

The following procedure describes how to run an application on a remote machine with the `rlogin` command.

▼ How to Run an Application on a Remote Machine

1. **Use the `xhost` command on the local machine to give the remote machine display access.**

   ```
   starbug$ xhost + venus
   ```

2. **Log in to the remote machine.**

   ```
   starbug$ rlogin venus -l user2
   Password:
   Last login: Wed Nov  1 16:06:21 from starbug
   Sun Microsystems Inc.   SunOS 5.9       Generic February 2002
   ```

3. **Set the `DISPLAY` variable to the local machine.**

   ```
   venus$ DISPLAY=starbug:0.0
   ```

4. **Export the `DISPLAY` variable to the local machine.**

```
venus$ export DISPLAY
```

5. **Run the application.**

```
venus$ bigprogram &
```

Even though you interact with this application just as you would with any other application on your screen, the application runs on the remote machine.

The benefit from running an application in this way is that it uses fewer computing resources than an application that is installed on your machine. This example shows how to run any remote application to which you have access.

More About Security

This section describes some fundamentals of network security that you might find useful as you run applications across the network, including:

- User-based and host-based access control mechanisms
- The `MIT-MAGIC-COOKIE-1` and `SUN-DES-1` authorization protocols
- When and how to change a server's access control
- How to run applications remotely, or locally as a different user

Who Should Read this Section

You do not need to change the default security configuration in the Solaris operating environment, at minimum, unless you run applications in any of the following configurations:

- You run an application linked with versions of `Xlib` or `libcps` *older* than the OpenWindows Version 2 software or `X11R4`.
- You run an application that is statically linked to OpenWindows Version 2 libraries and you want to use the `SUN-DES-1` authorization protocol.
- You run an application on a remote server.

Access Control Mechanisms

An access control mechanism controls which clients or applications have access to the X11 server. Only properly authorized clients can connect to the server. All other clients are denied access, and are terminated with the following error message.

```
Xlib: connection to hostname refused by server
Xlib: Client is not authorized to connect to server
```

The connection attempt logs to the server console as:

```
AUDIT: <Date Time Year>: X: client 6 rejected from IP 129.144.152.193
   port 3485    Auth name: MIT-MAGIC-COOKIE-1
```

Two different types of access control mechanisms are used: *user based* and *host based*. That is, one mechanism grants access to a particular user's account, while the other mechanism grants access to a particular host, or machine. Unless the `-noauth` option is used with `Xsun`, both the user-based access control mechanism and the host-based access control mechanism are active. For more information see "Manipulating Access to the Server" on page 125.

User-Based Access

A user-based, or authorization-based, mechanism allows you to give access explicitly to a particular user on any host machine. The user's client passes authorization data to the server. If the data match the server's authorization data, the user is allowed access.

Host-Based Access

A host-based mechanism is a general-purpose mechanism. This type of mechanism enables you to give access to a particular host, in which all users on that host can connect to the server. A host-based mechanism is a weaker form of access control. If the host has access to the server, all users on that host are allowed to connect to the server.

The Solaris environment provides the host-based mechanism for backward compatibility. Applications that are linked with versions of `Xlib` or `libcps` older than OpenWindows Version 2 software or X11R4 do not recognize the new user-based access control mechanism.

Note – Relink clients that are linked with older versions of `Xlib` or `libcps` to enable them to connect to the server with the new user-based access control mechanism.

Authorization Protocols

Two authorization protocols are supported in this version of the Solaris operating environment: `MIT-MAGIC-COOKIE-1` and `SUN-DES-1`. They differ in the authorization data used. The protocols are similar in the access control mechanism used. The `MIT-MAGIC-COOKIE-1` protocol that uses the user-based mechanism is the default in the Solaris operating environment.

MIT-MAGIC-COOKIE-1

The MIT-MAGIC-COOKIE-1 authorization protocol was developed by the Massachusetts Institute of Technology. At server startup, a *magic cookie* is created for the server and the user who started the system. On every connection attempt, the user's client sends the magic cookie to the server as part of the connection packet. This magic cookie is compared with the servers' magic cookie. The connection is allowed if the magic cookies match, or denied if they do not match.

SUN-DES-1

The SUN-DES-1 authorization protocol, developed by Sun Microsystems, is based on Secure Remote Procedure Call (RPC) and requires Data Encryption Standard (DES) support. The authorization information is the machine-independent netname, or network name, of a user. This information is encrypted and sent to the server as part of the connection packet. The server decrypts the information and, if the netname is known, allows the connection.

This protocol provides a higher level of security than the MIT-MAGIC-COOKIE-1 protocol. No other user can use your machine-independent netname to access a server, but another user can use the magic cookie to access a server.

This protocol is available only in libraries in the Solaris 1.1 and compatible environments. Any applications built with static libraries, in particular Xlib, in environments prior to OpenWindows Version 3 cannot use this authorization protocol.

"Allowing Access When Using SUN-DES-1" on page 127 describes how to allow another user access to your server by adding that person's netname to your server's access list.

Changing the Default Authorization Protocol

You can change the default authorization protocol, MIT-MAGIC-COOKIE-1, to SUN_DES-1, the other supported authorization protocol, or to no user-based access mechanism at all. You can change the default authorization protocol by editing the Xsun line in the /usr/dt/config/Xservers file. For example, to change the default from MIT-MAGIC-COOKIE-1 to SUN-DES-1, add the -auth sun-des option to the Xsun command by editing the following line in the /usr/dt/config/Xservers file.

```
:0  Local local_uid@console root /usr/openwin/bin/Xsun :0 -nobanner -auth sun-des
```

If you must run the Solaris operating environment without the user-based access mechanism, add the -noauth option to the Xsun command by editing the following line in the /usr/dt/config/Xservers file.

```
:0  Local local_uid@console root /usr/openwin/bin/Xsun :0 -nobanner -noauth
```

Caution – By using the `-noauth` option, you weaken security. It is equivalent to running Solaris software with the host-based access control mechanism only. The server inactivates the user-based access control mechanism. Anyone who can run applications on your local machine is allowed access to your server.

Manipulating Access to the Server

Unless the `-noauth` option is used with `Xsun` (see "Changing the Default Authorization Protocol" on page 124), both the user-based access control mechanism and the host-based access control mechanism are active. The server first checks the user-based mechanism, then the host-based mechanism. The default security configuration uses `MIT-MAGIC-COOKIE-1` as the user-based mechanism, and an empty list for the host-based mechanism. Because the host-based list is empty, only the user-based mechanism is effectively active. Using the `-noauth` option instructs the server to inactivate the user-based access control mechanism, and initializes the host-based list by adding the local host.

You can use either of two programs to change a server's access control mechanism: `xhost` and `xauth`. For more information, see these man pages. These programs access two binary files that are created by the authorization protocol. These files contain session-specific authorization data. One file is for server internal use only. The other file is located in the user's `$HOME` directory:

- `.Xauthority`
- `Client Authority file`

Use the `xhost` program to change the host-based access list in the server. You can add hosts to, or delete hosts from the access list. If you are starting with the default configuration—an empty host-based access list—and use `xhost` to add a machine name, you lower the level of security. The server allows access to the host you added, as well as to any user who specifies the default authorization protocol. See "Host-Based Access" on page 123 or an explanation of why the host-based access control mechanism is considered a lower level of security.

The `xauth` program accesses the authorization protocol data in the `.Xauthority` client file. You can extract this data from your `.Xauthority` file so that other users can merge the data into their `.Xauthority` files, thus allowing them access to your server, or to the server to which you connect.

See "Allowing Access When Using `MIT-MAGIC-COOKIE-1`" on page 126 for examples of how to use `xhost` and `xauth`.

Client Authority File

The client authority file is `.Xauthority`. It contains entries of the form:

```
connection-protocol          auth-protocol          auth-data
```

By default, `.Xauthority` contains `MIT-MAGIC-COOKIE-1` as the *auth-protocol*, and entries for the local display only as the *connection-protocol* and *auth-data*. For example, on host *anyhost*, the `.Xauthority` file might contain the following entries:

```
anyhost:0        MIT-MAGIC-COOKIE-1   82744f2c4850b03fce7ae47176e75
localhost:0      MIT-MAGIC-COOKIE-1   82744f2c4850b03fce7ae47176e75
anyhost/unix:0   MIT-MAGIC-COOKIE-1   82744f2c4850b03fce7ae47176e75
```

When the client starts, an entry corresponding to the *connection-protocol* is read from `.Xauthority`, and the *auth-protocol* and *auth-data* are sent to the server as part of the connection packet. In the default configuration, `xhost` returns empty host-based access lists and states that authorization is enabled.

If you have changed the authorization protocol from the default to `SUN-DES-1`, the entries in `.Xauthority` contain `SUN-DES-1` as the *auth-protocol* and the netname of the user as *auth-data*. The netname is in the following form:

`unix.`*userid@NISdomainname*

For example, on host *anyhost* the `.Xauthority` file might contain the following entries, where unix.`15339@EBB.Sun.COM` is the machine-independent netname of the user:

```
anyhost:0          SUN-DES-1          "unix.15339@EBB.Sun.COM"
localhost:0        SUN-DES-1          "unix.15339@EBB.Sun.COM"
anyhost/unix:0     SUN-DES-1          "unix.15339@EBB.Sun.COM"
```

Note – If you do not know your network name, or machine-independent netname, ask your system administrator.

Allowing Access When Using `MIT-MAGIC-COOKIE-1`

If you are using the `MIT-MAGIC-COOKIE-1` authorization protocol, follow these steps to allow another user access to your server:

1. On the machine that is running the server, use `xauth` to extract an entry that corresponds to *hostname:0* into a file.

   For this example, *hostname* is *anyhost* and the file is *xauth.info*:

   ```
   myhost% /usr/openwin/bin/xauth nextract - anyhost:0 > $HOME/xauth.info
   ```

2. Send the file that contains the entry to the user who requests access,

   Use an email tool, the `rcp` command, or some other file transfer method to send the file.

Note – Mailing the file that contains your authorization information is a safer method than using `rcp`. If you do use `rcp`, do *not* place the file in a directory that is easily accessible by another user.

3. The other user must merge the entry into his or her `.Xauthority` file.

   For this example, *userhost* merges `xauth.info` into the other user's `.Xauthority` file:

   ```
   userhost% /usr/openwin/bin/xauth nmerge - < xauth.info
   ```

Note – The *auth-data* is session-specific. Therefore, it is valid only as long as the server is not restarted.

Allowing Access When Using `SUN-DES-1`

If you are using the `SUN-DES-1` authorization protocol, follow these steps to allow another user access to your server.

1. On the machine that runs the server, use `xhost` to make the new user known to the server.

   For this example, to allow new user *somebody* to run on *myhost*:

   ```
   myhost% xhost + somebody@
   ```

2. The new user must use the `xauth` command to add the entry into his or her `.Xauthority` file.

   For this example, the new user's machine independent netname is `unix.15339@EBB.Sun.COM`. Note that this command should be typed on one line with no carriage return. After the pipe symbol, type a space followed by the remainder of the command.

   ```
   userhost% echo 'add myhost:0 SUN-DES-1 "unix.15339@EBB.Sun.COM"' |
   $OPENWINHOME/bin/xauth
   ```

Running Clients Remotely, or Locally as Another User

X clients use the value of the `DISPLAY` environment variable to get the name of the server to which they should connect.

To run clients remotely, or locally as another user, follow these steps:

1. On the machine that runs the server, allow another user access.

   Depending on which authorization protocol you use, follow the steps outlined in either "Allowing Access When Using `MIT-MAGIC-COOKIE-1`" on page 126 or

"Allowing Access When Using SUN-DES-1" on page 127.

2. Set DISPLAY to the name of the host that runs the server.

   For this example, the host is *remotehost*:

   ```
   myhost% setenv DISPLAY remotehost:0
   ```

3. Run the client program as shown.

   ```
   myhost% client_program&
   ```

   The client is displayed on the remote machine, *remotehost*.

CHAPTER 10

Customizing Your Working Environment

You can modify the *environment variables* in your system *initialization files* to control and adjust many aspects of your working environment. When you log in, your system reads the initialization files and uses the environment variables to configure your system. By setting the environment variables, you can customize your system to make it easier and more efficient to do your work.

This chapter describes the following tasks.

Modifying Initialization Files

The particular initialization files responsible for your system's configuration depend on which shell the system administrator has specified as your default shell when your system was first installed. The Bourne shell is the default shell for the Solaris operating environment, but you can also use the C shell or Korn shell. Each of these shells has its own initialization file (or files).

To determine your default shell (your *login shell*), follow these steps.

1. **Type** `echo` **`$SHELL`.**

```
$ echo $SHELL
/bin/sh
```

2. **Review the output of the command to determine your default shell.**

   Refer to the following list to identify your default shell.

   - `/bin/sh` – Bourne shell
   - `/bin/bash` – Bourne Again shell
   - `/bin/csh` – C shell
   - `/bin/ksh` – Korn shell
   - `/bin/tcsh` – TC shell
   - `/bin/zsh` – Z shell

   Regardless of the shell you are using, when you first log in your system generally runs the system profile file, `/etc/profile`. This file is generally owned by the system administrator and is readable (but not writable) by all users.

   After your system executes the system profile, it runs the *user profile*. The user profile is one (or more) initialization files that define your working environment. For example, if you're in the CDE environment your system checks this file (or set of files) each time you start a new terminal or window.

   Depending on which shell is set up as your default, your user profile can be one of the following:

   - `.profile` (for the Bourne and Korn shells)
   - `.bash-profile` (for the Bourne Again shell)
   - `.login` and `.cshrc` (for the C shell)
   - `.tcshrc` and `.cshrc` (for the TC shell)
   - `.zlogin` and `.zshrc` (for the Z shell)

   Your user profile file(s) is located in your home directory and enables you to configure your working environment to your preference.

Setting Environment Variables

Your system sets up your system environment by using a set of specifications that are defined in the initialization files. If you want to temporarily modify your environment for the current work session, you can issue commands directly at the command prompt. However, if you want to modify your working environment on a more permanent basis, you can store "permanent" environment variables in the appropriate user profile files.

To display the environment variables that are currently set on your system, use the `env` command.

- **Type the `env` command and press Return:**

```
$ env
HOME=/home/user2
PATH=/usr/bin:
LOGNAME=user2
HZ=100
TERM=dtterm
TZ=US/Mountain
SHELL=/bin/csh
MAIL=/var/mail/user2
PWD=/home/user2
USER=user2
$
```

Note – You can also use the `env` command to identify your login shell. It is specified in the `SHELL` environment variable. In the previous example, the shell is set to `/bin/csh` (the C shell).

User Profile

This section describes some of the more commonly used environment variables. Many of these variables might already be in your user profile. As previously mentioned, your user profile file is located in your home directory.

Note – To view hidden ("dot") files, use the `-la` options of the `ls` command.

The following is a partial list of environment variables that you can include in your user profile. Your current shell determines the syntax for defining environment variables.

- `CDPATH` – Specifies the directories to be searched when a unique directory name is typed without a full path name.
- `HISTORY` – Sets the number of commands available to the `history` command.
- `HOME` – Defines the absolute path to your home directory. The system uses this information to determine the directory to change to when you type the `cd` command with no arguments.
- `LANG` – Specifies the local language. Appropriate values are Japanese, German, French, Swedish, and Italian.
- `LOGNAME` – Defines your login name. The default for this variable is automatically set to the login name specified in the `passwd` database as part of the login process. See *System Administration Guide: Basic Administration* for information on the `passwd` database.

- LPDEST – Defines your default printer.
- MAIL – Specifies the path to your mailbox, which is usually located in the /var/mail/*username* directory, where *username* is your login name. See Chapter 7 for more information on this file.
- MANSECTS – Sets the available sections of online man pages.
- PATH – Lists, in order, the directories that the system searches to find a program to run when you type a command. If the appropriate directory is not in the search path, you have to type it or else type the complete path name when you enter a command.

  The default for this variable is automatically defined and set as specified in your user profile file as part of the login process.
- PS1 – Defines your command prompt. The default prompt for the Bourne, Bourne Again, and Korn shells is the dollar sign ($). The default prompt for the C, TC, and Z shells is the percent sign (%). The default prompt for root in either shell is the pound sign (#).
- SHELL – Defines the shell that is used by vi and other tools.
- TERMINFO – Specifies the path name for an unsupported terminal that has been added to the terminfo database. You do not need to set this variable for default terminals in this database. See *System Administration Guide: Advanced Administration* for information on the terminfo database.
- TERM – Defines the terminal you're currently using. When you run an editor, the system searches for a file with the same name as the definition of this variable. The system first searches the path (if any) referenced by the TERMINFO variable, and then the default directory, /usr/share/lib/terminfo, to determine the characteristics of the terminal. If a definition is not found in either location, the terminal is identified as "dumb."
- TZ – Defines the time zone for your system clock.

Setting the PATH Variable

The PATH environment variable is used to locate commands within the SunOS directory hierarchy. By setting the PATH, you create a fixed set of directories that the system always searches whenever you type the name of a command.

For example, if you have no PATH variable set and you want to copy a file, you need to type the full path name for the command, /usr/bin/cp. However, if you have set the PATH variable to include the directory /usr/bin, then you can simply type cp and your system will always execute the command. This is because your system searches for the cp command in every directory that is named in the PATH variable, and executes it when it is found. You can significantly streamline your work by using the PATH variable to list the commonly used SunOS command directories.

For the Bourne, Bourne Again, and Korn shells, you can specify the PATH variable in your user profile file (in your home directory) by using the following syntax.

```
PATH=.:/usr/bin:/home/bin
```

In the previous example, *home* represents the path name of your home directory.

For the C, TC, and Z shells, you can specify the PATH variable in your user profile file (in your home directory) by using the following syntax:

```
set path=(/usr/bin home/bin .)
```

In this command, *home* is the path name of your home directory.

Note – In the C, Korn, TC, and Z shells you can use the shortcut ~ to represent the path name of your home directory.

If you modify the PATH variable, and you are running the C, TC, or Z shell, use the source command to make the changes effective in your current window without having to log out:

```
example% source user-profile-file
```

If you are running the Bourne, Bourne Again, or Korn shell, type the following to make the changes effective in your current window without having to log out:

```
$ . user-profile-file
```

Command Aliases

Command aliases are helpful shortcuts for commands you often type. For example, the default setting for the remove command (rm) does not ask for confirmation before it removes files. Sometimes this default is inconvenient, as a typing error can remove the wrong file. However, you can use the alias variable to change this setting by editing your user profile file.

In the C and TC shells, add the following line to your user profile file.

```
alias rm  'rm -i'
```

In the Bourne Again, Korn, and Z shells, add the following line to your user profile file.

```
alias rm='rm -i'
```

With this line in your user profile file, typing rm is now the same as typing rm -i, which is the interactive form of the rm command. You will then always be asked to confirm the command before any files are deleted. The quote marks around rm -i in

the previous example are necessary to include the blank space between `rm` and `-i`. Without the quotation marks, the shell cannot correctly interpret the text after the space.

To make your changes to your user profile file effective immediately in your current window, you need to type an additional command. In the C and TC shells, type the following command to make your alias effective immediately.

`example%` **`source`** *user-profile-file*

The `source` command causes the system to read the current user profile file and execute the commands in this file.

In the Bourne Again, Korn, and Z shells, type the following command to make your alias effective immediately.

`$ .` *user-profile-file*

In the Bourne Again, Korn, and Z shells, the `.` command performs the same actions as the `source` command in the C and TC shells.

Note – Command aliases that are created by using the `alias` command apply only to the current session.

Changing Your Command Prompt

The syntax you use to change your command prompt depends on what shell you are using.

Bourne, Bourne Again, Korn, and Z Shells

In the Bourne, Bourne Again, Korn, and Z shells, use the `PS1` command to redefine your command prompt. The following are three examples:

- To set the prompt to a colon (:), followed by a space, type the following command

  ```
  PS1=": "
  ```

- To create a prompt that consists of your machine name, followed by a colon and a space, type the following command.

  ```
  PS1="`hostname`: "
  ```

- To set the prompt to your machine name, followed by your login name in braces { }, a colon, and a space, type the following command

  ```
  PS1="`hostname`{`logname`}: "
  ```

Type any of the previous examples to change your current command prompt. This change applies until you change your command prompt again or log out.

If you want to make your changes more permanent, add one of the previous examples (or a prompt of your own creation) to your user profile file. If you follow this guideline, the prompt you specify appears each time you log in in or start a new shell.

C and TC Shells

For the C and TC shells, you personalize your command prompt with the `set prompt` command. The following are three examples:

- To set the prompt to a percent sign followed by a space, type the following command

  ```
  example% set prompt="% "
  ```

- To create a prompt that consists of your machine name, followed by the history number of the command (*hostname*`1`, *hostname*`2`, *hostname*`3`, and so on), followed by a colon, type the following command.

  ```
  example% set prompt="`hostname`\!: "
  ```

- To set the prompt to your machine name, followed by your login name in braces, a colon, and a space, type the following command.

  ```
  example% set prompt="`hostname`{`logname`}: "
  ```

Type any of the previous examples to change your current command prompt. This change applies until you change your command prompt again or log out.

If you want to make your changes more permanent, add one of the previous examples (or a prompt of your own creation) to your user profile file. If you follow this procedure, the prompt you specify appears each time you log in or start a new shell.

Other Useful Variables

You can set many other variables in your user profile file. For a complete list, refer to the *man Pages(1): User Commands*. The following sections describe some of the more commonly used options.

`noclobber` Variable

Use `set noclobber` to prevent unintentional overwriting of files when you use the `cp` command to copy a file. This variable affects the Bourne Again, C, Korn, and TC shells. Type the following in your user profile file:

```
set noclobber
```

`history` Variable

The `history` variable enables you to set the number of commands that you saved in your history list. The `history` command is useful to view commands you have previously typed. You can also use the history file to repeat earlier commands. Type the following in your `.cshrc` or `.tcshrc` file:

```
set history=100
```

You can also affect the Bourne, Bourne Again, Korn, and Z shells in the same manner by typing the following line in your user profile file.

```
HISTORY=100
```

Setting Default File Permissions

The `umask` command sets a default file permission for all the files and directories you create. For example, if you are security conscious and you want to grant members of your group, and all users, only read and execute permissions (`-rwxr-xr-x`) on your directories and files, you can set the `umask` in your user profile file so that every new file or directory you create is protected with these permissions.

Like the `chmod` command, `umask` uses a numeric code to represent absolute file permissions. However, the method that is used to calculate the code for `umask` is different from the method for `chmod`.

For example, if `umask` is set to 000, all files you create have the following (read and write, but not execute) permissions:

`rw-rw-rw-` (mode 666)

All directories that are created have the following (read, write, and execute) permissions:

`rwxrwxrwx` (mode 777)

To determine the value to use for `umask`, you subtract the value of the permissions you want (using the value you would specify for the `chmod` command) from the current default permissions assigned to files. The remainder is the value to use for the `umask` command.

For example, suppose you want to change the default mode for files from 666 (`rw-rw-rw-`) to 644 (`rw-r--r--`). Subtract 644 from 666. The remainder, 022, is the numeric value you would use with umask as follows:

```
$ umask 022
```

Similar to the numeric code for the `chmod` command, the three numbers that are used with `umask` are as follows.

- The first digit controls user permissions
- The second controls group permissions
- The third digit controls permissions for all others

Table 10-1 shows the file permissions that are created for each digit of the `umask` command's numeric code.

TABLE 10–1 Permissions for `umask`

umask code	Permissions
0	rwx
1	rw-
2	r-x
3	r--
4	-wx
5	-w-
6	--x
7	---(none)

For more information on the `umask` command, refer to the *man Pages(1): User Commands*.

APPENDIX **A**

Modifying the Keyboard

This appendix describes how to remap options for special keyboard keys and how to disable and enable the Compose key on your keyboard.

For information on remapping your mouse buttons (for example, more convenient left-handed use of the mouse), see the user documentation for your desktop environment.

Disabling and Enabling the Compose Key

IA only – The Compose key is defined to be Ctrl-Shift-F1 on IA-based systems.

If you do not use the Compose key, you can disable it so that you do not press it inadvertently. First, find out the keycode for Multi_key:

```
$ xmodmap -pk | grep Multi_key
```

Your system displays a line similar to:

nn `0xff20 (Multi_key)`

Use the two-digit keycode number at the beginning of the line, represented by *nn*, to construct the following line in your `.xinitrc` file.

```
xmodmap -e 'keycode nn = NoSymbol'
```

To re-enable the Compose key, comment out the previous line in your `.xinitrc` file and restart the OpenWindows software.

SPARC: Left-Handed Key Remapping

The key remapping script in this section (provided for the Type-4 and Type-5 keyboards) remaps most of the special keys on the left panel and right panel of the keyboard (that is, the keypads to the left and right of the main keyboard area).

SPARC only – Note the following sections that describe "Left-Handed Key Remapping" apply only to SPARC-based machines.

SPARC: Using the Remapping Script

To remap the special keys on the left panel and right panel of your keyboard, follow these steps.

1. **Create a file called `lefty.data` by using any text editor.**

   This file can be in any directory. step 4 must occur in the same directory in which you create this file.

2. **Type in the script as shown in "`lefty.data` Script" on page 141.**

   Any line with an exclamation point in front of it is a comment line, and does not execute any operation.

3. **Save the changes and quit the editor.**

4. **At the prompt, type:**

   ```
   $ xmodmap lefty.data
   ```

   Note – You must be in the same directory as the script file.

   ---

5. **Click a mouse button in the Workspace to make the script effective.**

   The keyboard is now remapped for left-handed use.

`lefty.data` Script

Type the following script into the file `lefty.data`, as described in step 2.

```
!
! lefty.data
!
! Data for xmodmap to set up the left and right function keys
! for left-handed use on Sun Type-4 keyboard. To use this data,
! type the following where <filename> is the name of the file
! (i.e., lefty.data).
!
! xmodmap <filename>
!
! The comments below correspond to the keycode assignments
! following immediately thereafter.
!
! swap L2 (Again) with R1 (Pause)
! swap L3 (Props) with R6 (KP_Multiply)
! swap L4 (Undo) with R4 (KP_Equal)
! swap L5 (Front) with R9 (KP_9)
! swap L6 (Copy) with R7 (KP_7)
! swap L7 (Open) with R12 (KP_6)
! swap L8 (Paste) with R10 (KP_Left)
! swap L9 (Find) with R15 (KP_3)
! swap L10 (Cut) with R13 (KP_1)
!
! chng R3 (Break) to L1 (Stop)
! chng R2 (Print) to R10 (Left)
! chng R5 (KP_Divide) to R12 (Right)
!
! chng Linefeed to Control-R
!
keycode 10 = R1 R1 Pause
keycode 28 = L2 L2 SunAgain
keycode 32 = R6 R6 KP_Multiply
keycode 54 = L3 L3 SunProps
keycode 33 = R4 R4 KP_Equal
keycode 52 = L4 L4 SunUndo
keycode 56 = R9 R9 KP_9 Prior
keycode 77 = L5 L5 SunFront
keycode 58 = R7 R7 KP_7 Home


keycode 75 = L6 L6 SunCopy
keycode 79 = Right R12 KP_6
keycode 100 = L7 L7 SunOpen
keycode 80 = Left R10 KP_4
keycode 98 = L8 L8 SunPaste
keycode 102 = R15 R15 KP_3 Next
keycode 121 = L9 L9 SunFind
keycode 104 = R13 R13 KP_1 End
keycode 119 = L10 L10 SunCut
keycode 30 = L1 L1 SunStop
```

```
keycode 29 = Left R10 KP_4
keycode 53 = Right R12 KP_6
keycode 118 = Control_R
add control = Control_R
```

SPARC: Undoing the Keyboard Remapping

You can switch the keys back to their original settings in the following ways.

- Exit the Solaris operating environment and start it up again
- Create a second script and initiate it any time you want to switch back

Follow these instructions to create the second script.

1. **Use a text editor to create a file called** `nolefty.data`**.**
2. **Type in the script as shown in "**`nolefty.data` **Script" on page 142**

   Any line with an exclamation point in front of it is a comment line, and does not execute any operation.
3. **Save the** `nolefty.data` **script in the same directory as the** `lefty.data` **script and quit the editor.**
4. **At the prompt, type the following command.**

   ```
   $ xmodmap nolefty.data
   ```

Note – For the `nolefty.data` file to execute, you must type the previous command in the same directory as the script file.

`nolefty.data` Script

```
!
! nolefty.data
!
! Data for xmodmap to reset the left and right function keys
! after being set for left-handed use on the Sun type-4 keyboard.
! To use this data, type the following where <filename> is the name
! of the file.
!
! xmodmap <filename>
!
! Reassign standard values to left function keys.
!
keycode 10 = L2 L2 SunAgain
keycode 32 = L3 L3 SunProps
keycode 33 = L4 L4 SunUndo
```

```
keycode 56 = L5 L5 SunFront
keycode 58 = L6 L6 SunCopy
keycode 79 = L7 L7 SunOpen
keycode 80 = L8 L8 SunPaste
keycode 102 = L9 L9 SunFind
keycode 104 = L10 L10 SunCut
!
! Reassign standard values to right function keys.
!
keycode 28 = R1 R1 Pause
keycode 29 = R2 R2 Print
keycode 30 = R3 R3 Scroll_Lock Break
keycode 52 = R4 R4 KP_Equal
keycode 53 = R5 R5 KP_Divide
keycode 54 = R6 R6 KP_Multiply
keycode 75 = R7 R7 KP_7 Home
keycode 77 = R9 R9 KP_9 Prior
keycode 98 = Left R10 KP_4
keycode 100 = Right R12 KP_6
keycode 119 = R13 R13 KP_1 End
keycode 121 = R15 R15 KP_3 Next
!
! Reassign the Linefeed key as such and remove from control map.
!
remove control = Control_R
keycode 118 = Linefeed
```

IA: Function Key and Control Key Remapping

You can remap the function keys of an IA machine so that they function like the Help, Cut, Copy, Paste, Undo, and Front keys on a SPARC keyboard. You can also remap the right Control key to be a Meta key.

IA only – Note that the following sections describing "Function Key Remapping" apply only to IA machines. After you remap the keys, you cannot use `kdmconfig` to change setup or video information without first undoing the keyboard remapping.

IA: Using the Remapping Script

Follow these steps to create and use your remapping script:

1. **Create a file in your home directory that is called `fkeys` by using any text editor.**

2. **Type in the script as shown in "`fkeys` Script" on page 144.**
3. **Save the changes and quit the editor.**
4. **At the prompt, type the following command.**

```
$ xmodmap fkeys
```

Note – You must be in the same directory as the script file.

---

5. **Click a mouse button in the Workspace to make the script effective.**

   After you complete these steps, you can use the function keys as Help, Cut, Copy, Paste, Undo, and Front keys.

`fkeys` Script

```
!
keysym F2 = L10
keysym F3 = L6
keysym F4 = L8
keysym F5 = L9

keysym F8 = L4
keysym F9 = L5

remove control = Control_R
keycode 0x47 = Meta_R
add mod1 = Meta_R
```

IA: Undoing the Keyboard Remapping

You can switch the keys back to their original settings in two ways.

- Exit the Solaris operating environment and start it up again
- Create a second script and initiate it any time you want to switch back

Follow these instructions to create the second script:

1. **Use a text editor to create a file called `normal`.**
2. **Type the script as shown in "`normal` Script" on page 145.**
3. **Save the `normal` file in the same directory as the `fkeys` script and quit the editor.**

4. **At the prompt, type:**

```
$ xmodmap normal
```

Note – You must type the previous command in the same directory as the script file.

`normal` Script

Type the following script into the file `normal`, as described in step 1.

```
keycode 8 = grave asciitilde
keycode 9 = 1 exclam
keycode 10 = 2 at
keycode 11 = 3 numbersign
keycode 12 = 4 dollar
keycode 13 = 5 percent
keycode 14 = 6 asciicircum
keycode 15 = 7 ampersand
keycode 16 = 8 asterisk
keycode 17 = 9 parenleft
keycode 18 = 0 parenright

keycode 19 = minus underscore
keycode 20 = equal plus
keycode 21 =
keycode 22 = BackSpace
keycode 23 = Tab
keycode 24 = Q
keycode 25 = W
keycode 26 = E
keycode 27 = R
keycode 28 = T
keycode 29 = Y
keycode 30 = U
keycode 31 = I
keycode 32 = O
keycode 33 = P
keycode 34 = bracketleft braceleft
keycode 35 = bracketright braceright
keycode 36 = backslash bar brokenbar
keycode 37 = Caps_Lock

keycode 38 = A
keycode 39 = S
keycode 40 = D
keycode 41 = F
keycode 42 = G
keycode 43 = H
keycode 44 = J
keycode 45 = K
keycode 46 = L
```

```
keycode 47 = semicolon colon
keycode 48 = apostrophe quotedbl
keycode 49 =
keycode 50 = Return
keycode 51 = Shift_L
keycode 52 =
keycode 53 = Z
keycode 54 = X
keycode 55 = C
keycode 56 = V
keycode 57 = B
keycode 58 = N
keycode 59 = M
keycode 60 = comma less
keycode 61 = period greater
keycode 62 = slash question
keycode 63 =
keycode 64 = Shift_R
keycode 65 = Control_L
keycode 66 =
keycode 67 = Alt_L
keycode 68 = space
keycode 69 = Alt_R
keycode 70 =
keycode 71 = Control_R
keycode 72 =
keycode 73 =
keycode 74 =
keycode 75 =
keycode 76 =
keycode 77 =
keycode 78 =
keycode 79 =
keycode 80 =
keycode 81 =
keycode 82 = Insert
keycode 83 = Delete
keycode 84 =
keycode 85 =
keycode 86 = Left
keycode 87 = Home
keycode 88 = End
keycode 89 =
keycode 90 = Up
keycode 91 = Down
keycode 92 = Prior
keycode 93 = Next
keycode 94 =
keycode 95 =
keycode 96 = Right
keycode 97 = Num_Lock
keycode 98 = Home KP_7 KP_7
keycode 99 = Left KP_4 KP_4
keycode 100 = End KP_1 KP_1
```

```
keycode 101 =
keycode 102 = KP_Divide
keycode 103 = Up KP_8 KP_8
keycode 104 = KP_5 KP_5 KP_5
keycode 105 = Down KP_2 KP_2
keycode 106 = KP_Insert KP_0 KP_0
keycode 107 = KP_Multiply
keycode 108 = Prior KP_9 KP_9
keycode 109 = Right KP_6 KP_6
keycode 110 = Next KP_3 KP_3
keycode 111 = Delete KP_Decimal KP_Decimal
keycode 112 = KP_Subtract
keycode 113 = KP_Add
keycode 114 =
keycode 115 = KP_Enter
keycode 116 =
keycode 117 = Escape
keycode 118 =
keycode 119 = F1
keycode 120 = F2
keycode 121 = F3
keycode 122 = F4
keycode 123 = F5
keycode 124 = F6
keycode 125 = F7
keycode 126 = F8
keycode 127 = F9
keycode 128 = F10
keycode 129 = SunF36
keycode 130 = SunF37
keycode 131 = Print SunSys_Req
keycode 132 = Scroll_Lock
keycode 133 = Pause Break
keycode 134 =
keycode 135 = Multi_key
keycode 136 = Mode_switch
```

Index

B

C

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z