Solaris® 2.2 Advanced User's Guide

2.2

Solaris®

SunSoft
A Sun Microsystems, Inc. Business

Solaris 2.2 Advanced User's Guide

2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 801-3660-10
Revision A, May 1993

SunSoft
A Sun Microsystems, Inc. Business

Contents

About This Book

Who Should Read This Book

This book is aimed at users of the Solaris™ 2.2 System Software. The Solaris 2.2 System Software consists of SunOS™ 5.2 and OpenWindows™ 3.2.

Before Reading This Book

Your system should be installed and ready for use. If it is not, see *Solaris 2.2 System Configuration and Installation Guide* or the installation manual specific to your system before continuing.

Additional Reading

The Solaris 2.2 System Software Answerbook provides access to a number of books about the Solaris 2.2 software, which are organized into the following related sets:

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

- Sun Developer's Set
  This set gives software developers the information they need to write, debug, and maintain programs on the system.

- Sun Reference Manual Set
  This set contains a description for every SunOS 5.2 command. Called `man` pages, they can optionally be installed as online documentation.
- Sun User's Set
  This set offers a detailed description of various aspects of the SunOS system, including using SunOS 5.2 commands, working with OpenWindows 3.2, customizing your work environment, handling problems, writing shell scripts, using electronic mail, and working on the network.

For more information on the Solaris 2.2 System Software Answerbook, see *Solaris 2.2 Roadmap to Documentation.*

Typographic Conventions

This book uses a number of typographic conventions:

- *Italic type* is used for emphasis, command arguments, variables, and book titles. For example:

  You must type the *filename* argument as described in *SunOS 5.2 Reference Manual.*

- `Courier bold type` indicates what you type during a sample session. For example:

```
$ date
```

- `Courier type` is used for program listings, command names, program names, or system names. For example:

  You can become superuser by typing `root` at the login prompt or by typing the `su` command at the command prompt.

It is also used for text that the system displays on the screen. For example:

```
$ who
bozo     console     Sep 11 15:36
```

A capitalized word within text indicates a key that you press. For example:

Press Return.

When you see two key names, press and hold the first key, and then type the second key. For example, when you see Control-D, press and hold the Control key, and then type D.

Note – Though the letter keys are labeled with capital letters (as in the D key in the sequence Control-D above), do not press the Shift key unless instructed to do so.

Logging In to SunOS and Starting OpenWindows 1

The OpenWindows Version 3.2 software should already be installed on your hard disk or on an accessible server in your file system. If you are unsure about this, see your system administrator, or refer to the *Solaris 2.2 System Configuration and Installation Guide.*

This chapter describes how to login to your system, how to use a shell command interpreter, how to mount and start up the OpenWindows user environment, how to exit the window system, and how to logout. It also describes some special cases, such as configuring the OpenWindows environment for dual monitors.

Logging In

A *standard work session* is defined as the interval between the time you log in to the system and the time you log out. The SunOS 5.2 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 those people who know the password. If you don't already have a login name and password, ask the person who is designated as the *system administrator* for your system to set up an account for you. Once 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:

```
login:
```

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

```
login: spanky
```

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

```
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. This is to help prevent others from discovering your password.

If you don't have a password on your account and would like one, or if you want to change your current password, see "Using a Password" in Chapter 5, "Passwords, Processes and Disk Storage" for instructions.

Your Login Shell

In the chapters that follow, you'll begin entering SunOS 5.2 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 take place within the system.

The default shell for SunOS 5.2 system software is the Bourne shell, but there are also two other shell programs available within the operating system: the C shell and the Korn shell. Each of these shells has its own unique differences. For complete information on each of these shells, see *SunOS 5.2 How-To Book: Basic System Administration Tasks*.

Note – You can get specific information about any SunOS 5.2 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" in Chapter 2, "Basic SunOS Commands."

When you initially log in to the system (or open a new Command Tool or Shell Tool 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 5.2 default (the Bourne shell), it is because a different shell (either the C or Korn shell) has been specified for you by your system administrator.

As mentioned, each shell has its differences. Some commands or procedures available when using one shell may not be available when using another. With this in mind, please note that whenever any commands or procedures are presented in this manual that are not available using the default shell for SunOS 5.2 (the Bourne shell), the sections are clearly marked as such.

Logging Out

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

```
$ exit
```

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

```
$ exit
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 – With the SunOS 5.2 operating system, turning off your workstation or terminal does *not* necessarily log you out. Unless you log out explicitly, you may remain logged in to the system.

OpenWindows Start-up Considerations

Before you start the OpenWindows software, take note of these considerations. If none of these considerations apply to you, skip ahead to "Displaying the OpenWindows Software," in this chapter.

- If it is your responsibility to set up an OpenWindows server on your network, refer to the *Solaris 2.2 System Configuration and Installation Guide.*
- If you are currently running the OpenWindows Version 2 software see Appendix A, "Migrating to OpenWindows Version 3.2."
- If you are currently running the SunView user environment, see Appendix A, "Migrating to OpenWindows Version 3.2."
- If you are currently running a version of OpenWindows that is earlier than Version 3.2, see "The OPENWINHOME Environment Variable" and "Using the Correct Start-Up File," in this chapter.

The `OPENWINHOME` *Environment Variable*

If you are currently running a version of the OpenWindows software earlier than Version 3.2, you may have set up your system to use the `OPENWINHOME` environment variable. It is no longer recommended that users set the `OPENWINHOME` environment variable, either by-hand or from a start-up file.

When you run the `openwin` command it automatically sets the `OPENWINHOME` environment variable to `/usr/openwin`; therefore, you do not need to do it.

If you have set the `OPENWINHOME` environment variable in either the `.profile` or `.cshrc` file in your home directory, comment out the line or delete it altogether *before* running OpenWindows Version 3.2.

To remove, or comment out, the `OPENWINHOME` environment variable in the `.profile` or `.cshrc` file:

1. **Open the** `.profile` **or** `.cshrc` **file using a text editor such as** `vi`.
2. **Type a pound sign (#) before the variable, as shown below, or delete the line entirely.**
   If you are working in the `.profile` file, do Step `a`; if you are working in the `.cshrc` file, do Step `b`.

a. **In the `.profile` file:**

```
#OPENWINHOME=/usr/openwin
```

a. **In the `.cshrc` file:**

```
#setenv OPENWINHOME /usr/openwin
```

3. **Save and quit the file.**
4. **Unset the `OPENWINHOME` environment variable (or logout and log back in).** If you are running the Bourne or Korn shell, do Step `a`. If you are running the C shell, do Step `b`.

   a. **In the Bourne or Korn shell, type:**

```
$ unset OPENWINHOME
```

   b. **In the C shell, type:**

```
example% unsetenv OPENWINHOME
```

Once you have unset the environment variable you are ready to run the OpenWindows software, as described in "Displaying the OpenWindows Software," in this chapter.

Using the Correct Start-Up File

If you are currently running a version of the OpenWindows software earlier than Version 3.2, it is important to determine the status of your `.xinitrc` file. The `.xinitrc` file is an OpenWindows start-up file your home directory that may contain user-defined options.

To determine the status of your `.xinitrc` file, type the following commands:

```
$ cd
$ ls -a .xinitrc
```

Depending on the output of this command, do one of the following things:

- If you do not have a `.xinitrc` file (that is, the results of the previous `ls -a` command does not return a listing for the file) do nothing. If there is no `.xinitrc` file in your home directory, OpenWindows uses the system default start-up file.
- If you have a `.xinitrc` file (that is, the result of the previous `ls -a` command returns a listing for the file), but you have never made any changes to the file or do not want to keep the changes you have made, do Step 1 in "Start-Up File Procedures."
- If you have a `.xinitrc` file (that is, the result of the previous `ls -a` command returns a listing for the file), and you have made changes to the file that you want to keep, do Step 2 in "Start-Up File Procedures."

Start-Up File Procedures

1. **To delete the `.xinitrc` file from your home directory, type the following command:**

```
$ rm .xinitrc
```

2. **To retain the changes to your `.xinitrc` file, do the following steps:**

   a. **Move `.xinitrc` to `.xinitrc.save`:**

```
$ mv .xinitrc .xinitrc.save
```

   b. **Copy `$OPENWINHOME/lib/Xinitrc` to `.xinitrc` in your home directory:**

```
$ cp $OPENWINHOME/lib/Xinitrc $HOME/.xinitrc
```

c. **Add the lines that you want to keep from the** `.xinitrc.save` **to** `.xinitrc`.

Caution – When editing the `.xinitrc` file, do not add a secondary version of `olwm`, do not add `svenv`, and do not remove the line containing `$OPENWINHOME/lib/openwin-sys`.

Starting the OpenWindows User Environment

To start the OpenWindows user environment you perform the following general steps:

1. Using NFS, mount the OpenWindows software from the server on which it is installed.

   For information on how to mount the OpenWindows software from a server, see *OpenWindows Version 3.2 Reference Manual*, or talk to your system administrator.

2. Start the OpenWindows software with the command `openwin`, adding any additional start-up options as needed.

Displaying the OpenWindows Software

Once you have mounted the OpenWindows software from a server and run the OpenWindows installation script you are ready to start the OpenWindows software.

To start the OpenWindows software, type `/usr/openwin/bin/openwin` at the shell prompt and press Return.

```
$ /usr/openwin/bin/openwin
```

This displays the OpenWindows Version 3.2 screen and sets up the OpenWindows working environment.

Displaying OpenWindows Quickly

Once you have successfully started OpenWindows, you can set up your system to use a shortcut so that you do not need to type the full OpenWindows path each time.

If you are using the Bourne or Korn shells you do this by placing a shell function in your `.profile` file. If you are using the C shell you put an *alias* in your `.cshrc` file. Both the `.profile` and `.cshrc` files are found in your home directory.

When you have placed the shortcut in the appropriate file for your shell, to start OpenWindows simply type:

```
$ openwin
```

How to enter the OpenWindows shortcut into your start-up files is described in the following sections, "In the .profile File," and "In the .cshrc File."

In the `.profile` *File*

To enter the OpenWindows shortcut into your `.profile` file:

1. **Open the `.profile` file using a text editor such as `vi`.**
2. **Enter the following shell function, exactly as shown, into the file:**

```
openwin () {
        /usr/openwin/bin/openwin
}
```

3. **Save and quit the file.**
4. **Logout and log back in to activate the shortcut, or type:**

```
$ . .profile
```

Now, whenever you want to start OpenWindows, you simply have to type `openwin`.

In the `.cshrc` *File*

To enter the OpenWindows shortcut into your `.cshrc` file:

1. **Open the `.cshrc` file using a text editor such as `vi`.**
2. **Enter the following alias command, exactly as shown, into the file:**

```
alias openwin /usr/openwin/bin/openwin
```

3. **Save and quit the file.**
4. **Logout and log back in to activate the shortcut, or type:**

```
example% source .cshrc
```

Now, whenever you want to start OpenWindows, you simply have to type `openwin`.

If OpenWindows Won't Display

When you start the OpenWindows software it is accessed through the directory `/usr/openwin`. The OpenWindows software is installed in this location by default. Many applications, for example Calendar Manager, cannot load unless the OpenWindows software is properly installed in `/usr/openwin`.

If your OpenWindows does not start when you type the command `/usr/openwin/bin/openwin`, either you do not have the OpenWindows software installed, or it is installed in a directory other than `/usr/openwin`. See the *Solaris 2.2 Quick Installation Guide*, the *OpenWindows Version 3.2 Reference Manual*, or see your system administrator.

Quitting the OpenWindows Environment

Once you have displayed the OpenWindows software and are working in the windows environment, you cannot logout as you would from a standard SunOS command-line session. You must first exit from the windows environment and then logout.

If you type `logout` at a shell prompt, you see the message:

```
Not login shell.
```

To exit from the OpenWindows environment, follow these steps:

1. **Position the mouse so that the arrow (*pointer*) is on the background of your screen (the *workspace*).**
2. **Press the MENU mouse button.**
   The Workspace menu appears, presenting several options.
3. **Drag the pointer down the menu until you highlight the last menu item,** `Exit`.
4. **Release the mouse button.**
   A pop-up window appears, asking you to confirm that you want to exit the window system.
5. **Position the pointer on** `Exit` **and click the SELECT mouse button.**
   After a few moments all the windows are dismissed and the system prompt appears in the lower left corner of your screen.

Special OpenWindows Start-up Options

Most users can start the OpenWindows software by following the steps described in "Displaying the OpenWindows Software." However, in some cases you may want to use additional options to modify the OpenWindows start-up.

This section describes the following special cases:

- Starting the OpenWindows software with reduced network security.
- Starting the OpenWindows software with various monitor and frame buffer types.
- Starting the OpenWindows software on multiple screens.

To start up the OpenWindows software with special options, you use the `openwin` command:

```
$ openwin [ options ]
```

In the preceding example, *options* are the command line options that enable you to tailor the default setup of the server. The following sections describe some of more commonly used options.

Starting with Reduced Network Security

If you are operating in an open networked environment *and are not concerned about network security,* you may want to use the `-noauth` option so that other users can run applications on your system.

The following command overrides the default security feature, which enables you to specify other users who can access your window server:

```
$ openwin -noauth
```

Starting with Various Monitor Types

If you have a gray-scale monitor (a non-color monitor with a frame buffer of 8 bits or more) you may want to use the `grayvis` option when you start up the OpenWindows software. This may improve certain aspects of your OpenWindows display, but it is not required.

To use this option, type the following at the system prompt:

```
$ cd
$ openwin -dev /dev/fb grayvis
```

Starting with Multiple Monitors

Note – The following procedures require some system administration experience. If you have never configured a system, ask your system administrator for assistance.

To run the OpenWindows environment on multiple screens, you must inform the system of the additional devices and display types you want to run. You can either specify the device options or use the default values available with the openwin script that starts up the OpenWindows software.

Two options are required with the openwin command when you start the software on dual monitors:

```
$ openwin [ [ -dev device ] [ deviceoptions ] ]
```

The double brackets indicate that the combination of [-dev *device]* *[deviceoptions*] can be entered more than once on the command line (that is, once per device).

[-dev *device*]

The *device* command line option specifies the frame buffer device which the server should use for the display, or screen.

If the command line does not show this option, the server uses the default, /dev/fb. Multiple (more than one) occurrences of the -dev option on the command line indicate multiple displays on the same server.

[*deviceoptions*]

The *deviceoptions* command line option is a list of device modifiers that change the behavior of the device specified in the -dev option.

Device Option Examples

This section provides examples of stacked and side-by-side dual-monitor arrangements.

Note – In all examples, the order of the devices is important. The first device specified must be the screen physically placed to the left or top of the second device. The second device specified must be the screen physically placed to the right or bottom of the first device.

[left] [right]

The following command line instructs the system to start up two displays. The left display is the default frame buffer and the right display is a monochrome. This enables you to move the cursor left and right between the two displays.

```
$ openwin -dev /dev/fb left -dev /dev/fbs/bwtwo0 right
```

The following example is equivalent to the previous example. By default, the first device is to the left of the second device listed in the command line.

```
$ openwin -dev /dev/fb -dev /dev/fbs/bwtwo0
```

The following command line instructs the system to start up two displays. The right display is the default frame buffer and the left display is a monochrome. This setup enables you to move the cursor left and right between the two displays.

```
$ openwin -dev /dev/fb right -dev /dev/fbs/bwtwo0 left
```

[top] [bottom]

The following command line instructs the system to start up two displays. The top display is a CG6 and the bottom display is a monochrome. This setup enables you to move the cursor up and down between the two displays.

```
$ openwin -dev /dev/fbs/cgsix0 top -dev /dev/fbs/bwtwo0 bottom
```

The following example is *not* equivalent to the previous example. By default, the first device is to the *left* of the second device listed in the command line.

```
$ openwin -dev /dev/fbs/cgsix0 -dev /dev/fbs/bwtwo0
```

The following command line instructs the server to start up two displays. The bottom display is a CG6 and the top display is a monochrome. This setup enables you to move the cursor up and down between the two displays.

```
$ openwin -dev /dev/fbs/cgsix0 bottom -dev /dev/fbs/bwtwo0 top
```

Miscellaneous Notes

The following are important considerations when you are running multiple screens.

- By default, `olwm` manages all screens.
- You cannot move windows between screens.

Basic SunOS Commands 2

This chapter provides an introduction to user commands in the SunOS 5.2 operating system. It describes how to enter commands, how to correct typing mistakes, how to enter long or multiple commands, how to use command options, and other useful information about SunOS 5.2 commands.

To enter commands, use a Command Tool or Shell Tool window. To display these windows, select the Programs submenu on the Workspace menu.

The Command Prompt

Once you've logged in, the screen or window will be empty except for an initial prompt. The nature of this prompt will vary depending on the shell you are using and on how your system administrator originally set it up. Since the default command prompt for SunOS 5.2 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" in Chapter 10, "Customizing Your Working Environment" for instructions.

Entering Commands

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

```
$ date
Mon Feb 3 10:12:51 PST 1992
$
```

As you can see, this command displays the current date and time. Now try entering the same command, but capitalized:

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

As you can see, an uppercase `D` is not the same as a lowercase `d` when interpreted by the system. Nearly all commands in the SunOS 5.2 operating system are lowercase.

Correcting Typing Mistakes

Suppose you started to type `Date`, but then realized your mistake before pressing the Return key. The text you type is not sent to the system until you press Return. Therefore, it is still possible to correct your mistake. You have two choices:

- Press the Delete or Backspace key to backspace 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 and see how they work. (The Delete/Backspace key varies on some systems. Ctrl-U should work on most systems.)

Entering Multiple Commands and Long Commands

You can enter 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
Mon Feb 3 10:19:25 PST 1992
spanky
$
```

As you can see, this 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 very long command, you can use the backslash character (\) to continue typing on a second line. For example:

```
$ date; \
logname
Mon Feb 3 10:23:25 PST 1992
hankw
$
```

Even though the `date` and `logname` commands are by no means long commands, they are used in this example to demonstrate the concept of continuing a set of commands on the next line in as simple a manner as possible. Later, when the commands you want to use are longer than the width of your screen, you'll see how using the backslash character can be extremely useful.

Note – If you are using the Shell Tool or Command Tool windows in the OpenWindows environment, you won't 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're typing wrap to the next line automatically, and the system executes all commands when you press Return.

Repeating Previous Commands

Note – The command repeating features described in this section are available only if you are using the C shell.

A quick way to repeat the last command you typed is to type `!!` and press Return. The system keeps a *history* of commands you type and is able to repeat previous commands. For example, if the last command you entered was `date`:

```
example% !!
date
Mon Feb 3 10:26:20 PST 1992
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 /usr/home/worker
5  logname
6  date
7  history
```

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, you would type the following:

```
example% !-2
logname
hankw
example%
```

Using the example history list above, the `logname` command is repeated.

Still another method is to follow the `!` with the first few characters of a previous command. For example, if you had previously entered 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 will repeat the most recent command beginning 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
Mon Feb 3 11:06:51 GMT 1993
$
```

Most options are expressed as a single character preceded by a dash (`-`). Not all commands have options. Some commands have more than one. 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. There are special symbols that allow you to *redirect* the output of a command. For example, you may 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. Next, the contents of sample.file are displayed with the more command:

```
$ more sample.file
Mon Feb 3 12:56:26 PST 1993
$
```

As you can see, the contents of sample.file now contains the output from the date command. (See Chapter 3, "Working with Files and Directories," for information on the more command.)

Sometimes you may want to redirect the output of one command as input to another command. A set of commands strung together in this fashion 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 may 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, you would enter the following:

```
$ date | lp
$
```

This would print the results of the date command. (See "Submitting Print Requests to the Default Printer" in Chapter 8, "Using Printers," for information on using the lp command to print files.)

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

Running Commands in the Background

Often, it is convenient to initiate a command from the command prompt and then place that command in the *background*. When a command is not placed in the background, the next prompt does not appear until the command completes its task. However, some commands can take a long time to finish, and you might prefer to enter other commands in the meantime.

If you know you want to run a command in the background, type an ampersand (`&`) after the command as shown below. The number that follows is the process id:

```
$ bigjob &
[1] 21414
$
```

The command `bigjob` will now run in the background, and you can continue to enter other commands. After the job completes, you will see a message similar to the following the next time you enter another command, such as `date` in this example:

```
$ date
Mon Feb 3 10:23:25 PST 1992
[1]  + Done    bigjob
$
```

If you are likely to log off before a background job completes, use the `nohup` (no hangup) command to allow the job to complete, as shown in this example. Otherwise, the background job will be terminated when you log off:

```
$ nohup bigjob &
[1] 21414
$
```

Getting Help with OS Commands

This section describes various on-line help features. These features allow you to view reference information from your workstation or terminal.

Note – The features described here are *in addition to* the OpenWindows help facilities.

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 5.2 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 which 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
$
```

Notice the number in parentheses after the command name in the above example. This number indicates the 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. If you look for the printed manual reference page for a command, you will find it in alphabetical order within its group.

Keyword Lookup with `apropos`

Suppose you know what you want to do, but you're not sure of the command to use. Then the `apropos` command is helpful. This command locates commands by keyword lookup. The `apropos` command lists all commands whose one-line summaries contain any keywords you supply. This can lead to a very lengthy display, as some keywords may appear in many places.

For some examples of apropos output, try entering one or more of the following:

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

If you do enter a keyword that generates an unreasonably lengthy display, pressing Ctrl-C interrupts the display and returns you to the command prompt. (Hold down the Control key and press "c".)

Working with Files and Directories 3

The SunOS 5.2 command line is used to manipulate files and directories. You type in the file and directory names in conjunction with SunOS commands to carry out specific operations. This is different than using the OpenWindows File Manager, where files are displayed as icons that can be clicked up on and moved, and commands are selected from menus.

This chapter introduces you to the concepts and procedures used to work with files and directories from the SunOS command line. These operations apply to any SunOS 5.2 command line, whether you are using a Shell or Command Tool in the OpenWindows environment or are logged in from a remote terminal. To fully make use of the SunOS 5.2 operating system it is essential for you to understand the concepts presented in this chapter.

File Concepts

The *file* is the basic unit in the SunOS 5.2 operating system. Almost everything is treated as a file, including:

- **Documents**—These include text files, such as letters or reports, computer source code, or anything else that you write and want to save.
- **Commands**—Most commands are *executable* files; that is, they are files you can execute to run a particular program. For example, the `date` command that you saw 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 available for creating, listing, copying, moving, and deleting files. You'll also see how to list the contents of a file and how to determine the nature of a file.

Using File Commands

Each of the commands presented in this section includes an example of how the command is used. Try the examples as you read the text. This practice will make the commands and their respective concepts easier to understand and remember.

Before You Begin

Before you start experimenting with files, make sure that you are in your *home* directory. This is a directory established for you by your system administrator when your account was created. If you perform the tasks shown in the following examples from your home directory, you'll be less likely to create, copy, move, or (worst of all) delete files within portions of the system that other users expect to remain unchanged.

To make certain that you are indeed in your home directory, type the `cd` (change directory) command by itself. This moves you to your home (default) directory. Then type the `pwd` (print working directory) command to display your current location within the filesystem. 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 owning the home directory.

Creating a Test File

Use the `touch` command to create an empty file. If a file by the name you specify doesn't already exist, the `touch` command creates an empty file (if the file already exists, `touch` updates the last file access time).

```
$ touch tempfile
$
```

Listing Files (`ls`)

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

```
$ ls tempfile
tempfile
```

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

For more information on the `ls` command, refer to the *SunOS 5.2 Reference Manual*.

Copying Files (`cp`)

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

```
$ cp tempfile copyfile
$
```

Now try listing both files. Notice that both names end with the characters "file." You can use the *wildcard* character, asterisk (*), to stand for any character or sequence of characters. Therefore, the command `ls *file` should list 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 the cp command, refer to the *SunOS 5.2 Reference Manual*.

Moving and Renaming Files (mv)

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

```
$ mv tempfile emptyfile
$
```

Now list both files again to verify the change:

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

As you can see, tempfile is replaced by emptyfile.

For more information on the mv command, refer to the *SunOS 5.2 Reference Manual*.

Deleting Files (rm)

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

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

Caution – Once you delete a file, it is gone for good. Unless there is a backup copy, the file cannot be restored. Be careful when using the `rm` command, and be particularly careful when using `rm` with the wildcard character `(*)`. Files removed with `rm` cannot be recovered.

For more detailed information on the `rm` command, refer to the *SunOS 5.2 Reference Manual*.

Displaying File Contents (`more`, `cat`)

Use the `more` command to display the contents of a file. Type `more` followed by the name of the file to be displayed. The contents of the file scrolls down the screen. If the file is longer than one screen, this message appears:

```
--More--(nn%)  [Press space to continue, 'q' to quit.]
```

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 flashes through the entire file 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` or `cat` commands, refer to the *SunOS 5.2 Reference Manual*.

Displaying File Type (`file`)

Some files, such as binary files and executable files, are not printable and cannot be displayed on the screen. The `file` command can be handy if you're not sure of the file type.

Use the `file` command to show the file type:

```
$ file copyfile
copyfile:    ascii text
```

Directories and Hierarchy

By now you know how to list, copy, rename, and delete files. However, you may be wondering about larger issues. Where are these files located? This section discusses the directory hierarchy. Read the following narrative carefully, and then try the examples in the sections that follow.

Directory Hierarchy

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

As shown in the following example, Figure 3-1, each directory in the file system can have many directories within it. The convention is to distinguish directory levels with the `/` character. With this in mind, notice that the directory `/` (root) contains the subdirectories `/usr`, `/bin`, `/home` and `/lib`, among others. The subdirectory `/home` contains `user1`, `user2`, and `user3`.

You specify directories (and files within them) by including the names of the directories they're in. This is called a *path name*. For example, the path name for the `user3` directory in the illustration above is `/home/user3`.

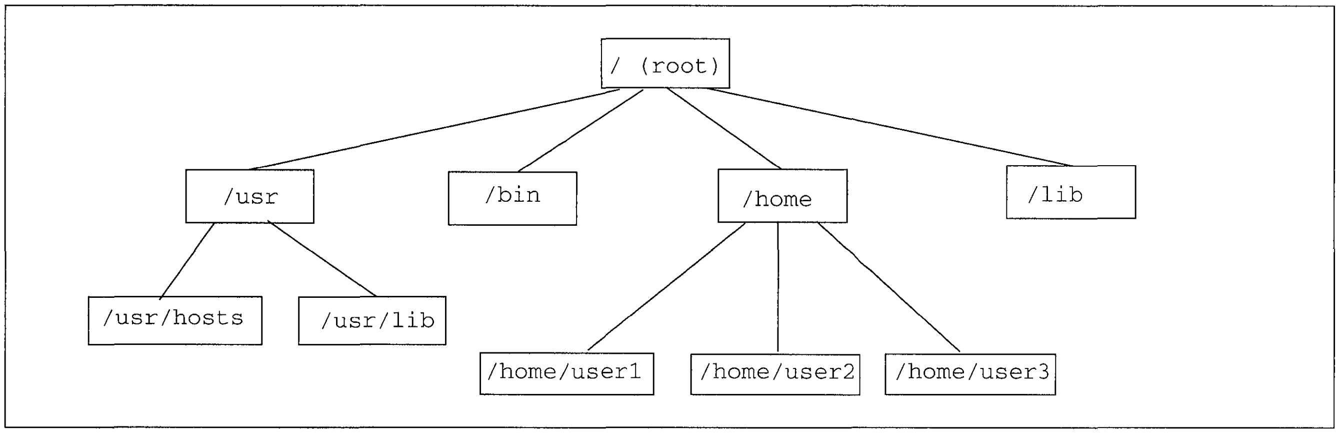


Figure 3-1 File System Hierarchy

All subdirectory 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`. There is no conflict 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 `/home/user2` directory contains a file called `report5`, the path name for this file is `/home/user2/report5`. This shows that the file `report5` is within the directory `user2`, which is within the directory `home`, which is within the root (`/`) directory.

Directories can contain only subdirectories, only files, or both.

Print Working Directory (pwd)

The command `pwd` (print working directory) tells you where you are in the file system hierarchy:

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

Your output will look somewhat different from that in the example, as your directory structure will be different. Remember that 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 the Command Tool or Shell Tool window in the OpenWindows environment, your initial location (working directory) is your home directory. This directory is established for you by the system administrator when your account is created.

Change Working Directory (`cd`)

The `cd` (change directory) command allows 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 `/home/user1`:

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

In the C shell, 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, would change to that user's home directory.

Note – If you are using the Bourne shell, the ~ shortcut will not work.

If you are using the Bourne shell, it may be possible that your system administrator has configured the system so that you can type `$home` to specify your home directory. If this is the case, then typing:

```
$ $home/music
```

changes you to the subdirectory `music` in your home directory. Likewise, typing:

```
$ $homeusername
```

changes you to the specified user's home directory, where *username* represents another user's login name.

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. Therefore, the command `cd ..` changes the working directory to the parent directory, as in this example:

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

Suppose your current working directory is /home/user1 and you want to work with some files in /home/user2. Here is a useful shortcut:

```
$ 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 is much easier than typing the entire path name /home/user2.

Creating a Directory (mkdir)

It is easy to create a new directory. Type the mkdir command followed by 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 much shorter name which defines the file or directory *relative* to the current working directory.

When you are in a parent directory, you can move to a subdirectory 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.

Try creating 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
$
```

This command 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, you will see 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 will not remove the directory.

Use `rm -r` (adding the *recursive* option `-r` to the `rm` command) to remove a directory and all its contents, including any subdirectories and their files, as follows:

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

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

Looking at Differences Between Files (`diff`)

It often happens that different people with access to a file make copies of the file and then edit their copies. `diff` will show you the specific differences between versions of an ASCII file. The command:

```
$ diff leftfile rightfile
```

scans each line in `leftfile` and `rightfile` looking for differences. When it finds a line (or lines) that differ, it determines whether the difference is the result of an addition, a deletion, or a change to the line, and how many lines are affected. It tells you the respective line number(s) in each file, followed by the relevant text from each.

If the difference is the result of an addition, `diff` displays a line of the form:

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

where *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 it is the result of a change on the line, `diff` uses a `c`.

The relevant lines 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, there is no output from `diff`.

The `diff` command has many more options than those discussed here. For more information, refer to the *SunOS 5.2 Reference Manual.*

Comparing Three Different Files (`diff3`)

If you have three versions of a file that you want to compare at once, use the `diff3` command as follows:

```
$ diff3 file1 file2 file3
```

`diff3` compares three versions of a file and publishes disagreeing ranges of text flagged with these codes:

==== all three files differ

====1 *file1* is different

====2 *file2* is different

====3 *file3* is different

Using `bdiff` *on Large Files*

If you are comparing very large files, use `bdiff` instead of `diff`. Both programs work in a similar manner:

```
$ bdiff leftfile rightfile
```

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

Looking Up 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 filenames that match a certain pattern or that have been modified within a specified time frame.

Unlike most commands, `find` options are several characters long, and the name of the starting directory must precede them on the command line as follows:

```
$ find directory options
```

where *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. Thus, the more options you apply, the narrower the field becomes. The `-print` option indicates that you want the results to be displayed. (As described later on, you can use `find` to run commands. You may want `find` to omit the display of selected files in that case.)

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/lib/calendar` is `calendar`. This portion of a file's name is often called its *base name*.

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

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

Other options include:

`-name` *filename*

Selects files whose rightmost component matches *filename*. Surround *filename* with single quotes if it includes filename substitution patterns.

`-user` *userid*

Selects files owned by *userid*. *userid* can be either a login name or user ID number.

`-group` *group*

Selects files belonging to *group*.

`-m time` *n*

Selects files that have been modified within *n* days.

`-newer` *checkfile*

Selects files modified more recently than *checkfile*.

You can specify an order of precedence 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
```

You can invert the sense of an option by prepending an escaped exclamation point. `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

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

option. This option is terminated with an escaped semicolon (`\;`). The quoted braces are replaced with the filenames that `find` selects.

As an example, you can use `find` to automatically remove temporary work files. If you name your temporary files consistently, you can use find to seek them out and destroy them wherever they lurk. For example, if you name your temporary files `junk` or `dummy`, this command will find them and remove them:

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

For more information, refer to the *SunOS 5.2 Reference Manual.*

File and Directory Security

Note – Read this section carefully. A clear understanding of file permissions is often important in day-to-day work.

File permissions help to protect files and directories from unauthorized reading and writing. Often you will have files you wish to allow others to read but not change. In other cases, you may have executable files (programs) to share. File permissions allow you to control access to your files.

These are the basic file and directory permission types:

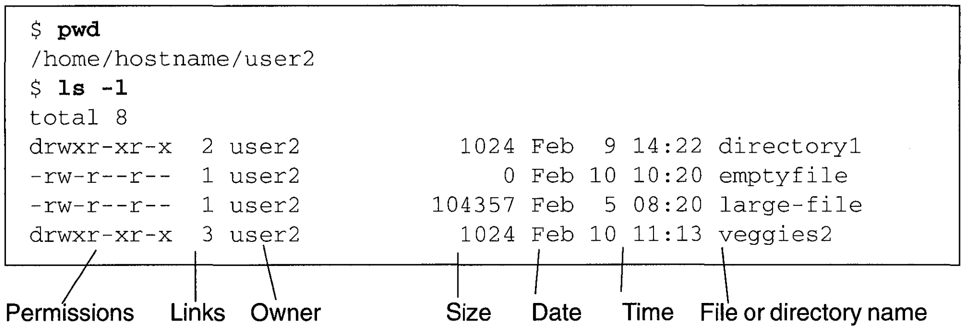
- `r` – *read* permission. A file must be readable to be examined or copied. A directory must be readable for you to list its contents.
- `w` – *write* permission. A file must be writable in order for you to modify it, remove it, or rename it. A directory must be writable in order for you to add or delete 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 for you to gain access to any of its subdirectories.

There are three categories of users for which you can set permissions:

- Self – The user
- Group – Other users within the same group as the user (for example, all accounting users). Groups are established and maintained by the system administrator.
- Others – Everyone else

Displaying Permissions and Status (`ls -l`)

You have already used the `ls` command to list files. The `ls` command has many options. Use the `-l` option to display a *long* format list. Files and directories are listed in alphabetical order. Figure 3-2 illustrates this method for displaying files:



```
$ pwd
/home/hostname/user2
$ ls -l
total 8
drwxr-xr-x  2 user2          1024 Feb  9 14:22 directory1
-rw-r--r--  1 user2             0 Feb 10 10:20 emptyfile
-rw-r--r--  1 user2        104357 Feb  5 08:20 large-file
drwxr-xr-x  3 user2          1024 Feb 10 11:13 veggies2
```

Figure 3-2 Displaying Permissions and Status

The very first character on the line indicates the file type. A dash (-) is 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)
- 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. Your results will differ from the example, of course.

Now try typing a command such as the following:

```
$ ls -l dirname
```

where *dirname* is the name of an actual directory within your file system. When you give the name of a directory, the `ls -l` command prints information on all the files and directories (if any) within that directory.

Listing "Hidden" Files (`ls -a`)

There are some files that are not listed by the ordinary `ls` command. These files have names beginning 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. There are two special files in this listing: the file `.` is the reference for the current directory, and the file `..` is the reference for the parent directory.

Generally speaking, files that begin with `.` are used by system utilities and are not usually modified by the user. There are a few exceptions to this.

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
```

where *permissions* indicates the permissions to be changed and name is the name of the affected file or directory.

The permissions can be specified in several ways. Here is one of the forms which is easiest to use:

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

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

```
$ ls -l carrots
drwxr-xr-x  3 user2          1024 Feb 10 11:15 carrots
$ chmod g+w carrots
$ ls -l carrots
drwxrwxr-x  3 user2          1024 Feb 10 11:15 carrots
$
```

As you can see, the hyphen (-) in the set of characters for group is changed to a w as a result of this command.

To make this same directory unreadable and unexecutable by other users outside your group (*permissions* is o-rx), you would enter the following:

```
$ ls -l carrots
drwxrwxr-x  3 user2           1024 Feb 10 11:15 carrots
$ chmod o-rx carrots
$ ls -l carrots
drwxrwx---  3 user2           1024 Feb 10 11:15 carrots
$
```

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

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

In general, the default settings for new files are:

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

and for new directories are:

```
drwxr-xr-x
```

So, to make a new file turnip executable by its owner (user2), you would enter the following:

```
$ ls -l turnip
-rw-r--r--  3 user2           1024 Feb 10 12:27 turnip
$ chmod u+x turnip
$ ls -l turnip
-rwxr--r--  3 user2           1024 Feb 10 12:27 turnip
$
```

If you want to affect all three categories of users at once, use the a option. To make a new file garlic executable by everyone, you would enter the following:

```
$ ls -l garlic
-rw-r--r--  3 user2          1024 Feb 10 11:31 garlic
$ chmod a+x garlic
$ ls -l garlic
-rwxr-xr-x  3 user2          1024 Feb 10 11:31 garlic
$
```

As a result, the x indicator appears in all three categories.

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

```
$ pwd
/home/user2/veggies
$ ls -l
-rwxrwxrwx  3 user2         21032 Feb 12 10:31 beats
-rwxrwxrwx  2 user2            68 Feb 10 11:09 corn
-rwxrwxrwx  3 user2         12675 Feb 08 09:31 garlic
-rwxrwxrwx  1 user2          1024 Feb 14 16:38 onions
$ chmod go-w *
$ ls -l
-rwxr-xr-x  3 user2         21032 Feb 12 10:31 beats
-rwxr-xr-x  2 user2            68 Feb 10 11:09 corn
-rwxr-xr-x  3 user2         12675 Feb 08 09:31 garlic
-rwxr-xr-x  1 user2          1024 Feb 14 16:38 onions
$
```

The pwd command is included in this example to illustrate that the directory on which you perform this chmod operation must be the current directory.

Setting Absolute Permissions

Up to this point, the discussion on permissions has only included using the `chmod` command to change permissions *relative* to their current settings. Using a different form of the `chmod` command, which applies numeric codes to specify permissions, you can set the permissions for a file or directory *absolutely.*

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

`chmod` *numcode name*

where *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 3-1 illustrates how the permissions described for `garlic` are represented by the code `771`.

Table 3-1 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 3-1 represents one of the categories: user, group, and others. To set read permissions, you add 4 to the appropriate column. To set write permissions, you add 2. To add execute permissions, you 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 this method for setting absolute permissions, with the `ls -l` command included to demonstrate the results:

```
$ ls -l onion
-rw-r--r--  3 user2          1024 Feb 10 11:46 onion
$ chmod 755 onion
$ ls -l onion
-rwxr-xr-x  3 user2          1024 Feb 10 11:48 onion
$
```

The permissions for the file `onion` are set so that the user can read, write, and execute; group members can read and execute; and others can also read and execute. Table 3-2 provides the breakdown of the numeric code used to set the permissions for `onion`.

Table 3-2 Permissions for `onion`

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

Of course, to provide read, write, and execute permissions for the file `cabbage` to yourself, your group, and all other users, you would enter the following:

```
$ ls -l cabbage
-rw-r--r--  3 user2          1024 Feb 10 11:51 cabbage
$ chmod 777 cabbage
$ ls -l cabbage
-rwxrwxrwx  3 user2          1024 Feb 10 11:53 cabbage
$
```

Table 3-3 provides the breakdown for this example.

Table 3-3 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, to set absolute permissions for all files in the current directory `veggies` so that you have read, write, and execute permissions; your group has read and execute permissions; and all other users have execute permissions only, you would enter the following:

```
$ pwd
/home/user2/veggies
$ ls -l
-rwxrwxrwx  3 user2          21032 Feb 12 10:31 beats
-rwxrwxrwx  2 user2             68 Feb 10 11:09 corn
-rwxrwxrwx  3 user2          12675 Feb 08 09:31 garlic
-rwxrwxrwx  1 user2           1024 Feb 14 16:38 onions
$ chmod 751 *
$ ls -l
-rwxr-x--x  3 user2          21032 Feb 12 10:31 beats
-rwxr-x--x  2 user2             68 Feb 10 11:09 corn
-rwxr-x--x  3 user2          12675 Feb 08 09:31 garlic
-rwxr-x--x  1 user2           1024 Feb 14 16:38 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, it's not necessary to know what the permissions are currently.

For more information on the `chmod` command, refer to the *SunOS 5.2 Reference Manual.*

Searching Files 4

This chapter describes how to search directories and files for keywords and strings using the SunOS 5.2 command `grep`.

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
```

where *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 may include "white space," 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 may 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*

grep is very often used as a "filter" with other commands. It allows you to filter out useless information from the output of commands. 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 ending in ".ps" that were created in the month of May:

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

The first part of this command line,

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

produces a list of files:

```
$ ls -l *.ps
-rw-r--r--  1 elvis       7228 Apr 22 15:07 change.ps
-rw-r--r--  1 elvis       2356 May 22 12:56 clock.ps
-rw-r--r--  1 elvis       1567 Jun 22 12:56 cmdtool.ps
-rw-r--r--  1 elvis      10198 Jun 22 15:07 command.ps
-rw-r--r--  1 elvis       5644 May 22 15:07 buttons.ps
$
```

The second part,

```
| grep May
```

pipes that list through `grep`, looking for the pattern `May`:

```
$ ls -l *.ps | grep May
-rw-r--r--  1 elvis       2356 May 22 12:56 clock.ps
-rw-r--r--  1 elvis       5644 May 22 15:07 buttons.ps
$
```

`grep` *with Multi-Word 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
$
```

grep 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 *don't* contain a certain string, use the -v option to grep. The following example shows how to find all of the lines in the user medici's home directory files that don't 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.
$
```

More on grep

You can also use the grep command to search for targets defined as patterns 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 and need to be quoted or escaped. Whenever you use a grep regular expression at the command prompt, surround it with quotes, or escape metacharacters (such as & ! . * $? and \) with a backslash (\).

- A caret (^) indicates the beginning of the line. So the command:

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

finds any line in the file list starting with "b."

- A dollar-sign ($) indicates the end of the line. The command:

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

  displays any line in which "b" is the last character on the line. And the command:

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

  displays any line in `list` where "b" is the *only* character on the line.

- Within a regular expression, dot (.) finds any single character. So the command:

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

  would match any three characters with "an" as the first two, including "any," "and," "management," and "plan" (because spaces count, too).

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

  Because it includes zero occurrences, usage of the asterisk is a little non-intuitive. Suppose you want to find all words with the letters "qu" in them. Typing:

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

  will work as expected. However, if you wanted to find all words containing the letter "n," you would have to type:

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

If you wanted to find all words containing the pattern "nn," you would have to type:

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

You may want to try this to see what happens otherwise.

- To match zero or more occurrences of *any* character in `list`, type:

```
$ grep .* list
```

Searching for Metacharacters

Suppose you want to find lines in the text that have a dollar sign (`$`) in them. Preceding the dollar sign in the regular expression with a backslash (`\`) tells `grep` to ignore (escape) its special meaning. This is true for the other metacharacters (`&  !  .  *  ?` and `\` itself) as well.

For example, the expression

```
$ grep ^\.
```

matches lines starting with a period, and is especially useful when searching for `nroff` or `troff` formatting requests (which begin with a period).

The following table, Table 4-1, provides a list of the more commonly used 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
`\`	Escapes special meaning of next character

Note that these search characters can also be used 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, you would type the following to use `grep` to search all files for the phrase "dang it, boys":

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

Single quotation marks (`'`) can also be used to group multiword phrases into single units. Single quotation marks also make sure that certain characters, such as `$`, are interpreted literally. (The `history` metacharacter `!` is always interpreted as such, even inside quotation marks, unless you escape it with a backslash.) In any case, it is a good idea to escape characters such as `&` `!` `$` `?` `.` `;` and `\` when you want them taken as ordinary typographical characters.

For example, if you type:

```
$ grep $ list
```

you will see *all* the lines in `list`. However, if you type:

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

you will see only those lines with the "`$`" character in them.

For more information on the `grep` command, refer to the *SunOS 5.2 Reference Manual*.

Passwords, Processes and Disk Storage 5

SunOS 5.2 provides a wealth of commands for doing a number of system tasks on the command line. This chapter describes how to set a password, how to list the processes running on your machine, how to kill unwanted processes, and how to display the amount of space being used on your disk.

For more information on additional commands you can use to work with processes, disk space, and passwords, see *SunOS 5.2 How-To Book: Basic System Administration Tasks.*

Using a Password

For the sake of your system's security, SunOS 5.2 requires the use of a password for your system. Changing your password several times a year helps to ensure that you are the only user with easy access to your account. If you believe someone has used your account without your permission, change your password immediately.

When choosing a password, keep the following in mind:

- Choose a password that you can remember without writing it down. A password that you can't 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.
- Although you can use any character in your password, some characters, such as Ctrl-C, Ctrl-Z, Ctrl-U, Ctrl-S, Esc, Tab, and in some cases # and @ can be interpreted by the terminal as signals. These characters should be avoided. The terminal may interpret these as signals rather than text characters, and this would preclude you from properly typing in your password.

Changing Your Password

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

```
$ passwd
Changing password for hankw on worker
Old password:
New password:
Retype new password:
$
```

1. **When the system prompts you for `Old Password:`, type your current password.**
   (If no password is currently assigned to your account, the system will skip the `Old Password:` prompt.) Note that the system does not echo (display) your password on the screen. This prevents other users from discovering your password.
2. **When the system prompts you for `New Password:`, type the password you've decided on.**
   Again, the password you type does not echo on the screen.
3. **At the final prompt, `Retype new password:`, type your new password a second time.**
   This is to verify that you typed exactly what you intended to type.

If you don't enter your password precisely the way you did at the previous prompt, the system refuses to change your password and responds with `Sorry`. If this happens 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 is using password aging (implemented with options to the `passwd` command), your password may have either a maximum, or a maximum *and* minimum lifespan. The lifespan of your password is set by your system administrator.

When the maturity date (or maximum age) of your password is reached, you are prompted to change your password. This occurs when you log in. The following is displayed:

```
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 that time has elapsed, the following is displayed:

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

To view aging information for your password, use the `-d` option to the `passwd` command:

```
$ passwd -d
username 2-14-92 14 60
```

The display shows, in order, the date the current password was created, the minimum age, and the maximum age. (This information appears only if password aging has been implemented.)

For more information on passwords and password aging, refer to the *SunOS 5.2 Reference Manual.*

Processes and PIDs

After each command is interpreted by the system, an independent *process*, with a unique process identification number (PID), is created 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. In addition to showing the *process identification number* (listed under `PID`) for each process you own (created as a result of a command you typed), `ps` 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`).

Adding the `-l` option to the `ps` command displays a variety of other information about the processes currently running, including the *state* of each process (listed under `S`). The codes used to show this are as follows:

- `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 created.
- `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, things can change. Since the `ps` command gives you only a snapshot of what's going on, it's only true for a split second after you type the command. The information may not be completely accurate by the time you see it.

The ps command has more options than those covered here. Refer to the *SunOS 5.2 Reference Manual.*

Terminating Processes (kill)

The kill command provides you with a direct way to stop command processes that you no longer want. This is particularly useful when you make a mistake typing a command that takes a long time to run.

To terminate a process:

1. **Type** ps **to find out the PID(s) for the process(es).**
2. **Type** kill **followed by the PID(s).**

The following example illustrates this procedure:

```
$ ps
PID     TTY    TIME    COMMAND
1291    co     0:12    -bin/csh (csh)
3250    p0     0:00    ps
1286    p1     0:05    -bin/csh (csh)
3248    p1     0:05    vi commands
$ kill 1291
[1}  Terminated        -bin/csh/ (csh)
$
```

Note that a faster way of determining the right PID is to pipe ps output through grep as follows:

```
$ ps | grep commandname
```

where *commandname* is the name of the command process you want to stop. (Use of the grep command is described in Chapter 4, "Searching Files.")

If you need to forcibly terminate a process, you can use the -9 option to the ps command as follows:

```
$ kill -9 PID#
```

where *PID#* is the process identification number of the process you want to stop.

Managing Disk Storage

Since space on the disk is a limited resource, it is a very good idea to keep track of the space currently in use.

Displaying Disk Usage (`df -k`)

`df -k` shows you the amount of space currently in use on each disk that is mounted (directly accessible) to your system. Just type:

```
$ df -k
```

to see the capacity of each disk mounted on your system, the amount available, and the percentage of space already in use.

File systems at or above 90 percent of capacity should be cleared of unnecessary files. You can do this either by moving them to a disk or tape that is less full, using `cp` to copy them and `rm` to remove them, or you can simply remove them outright. Of course, you should only perform these kinds of "housekeeping" chores 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 512-byte blocks; that is, units of 512 bytes or characters.

`du` shows you the disk usage in each subdirectory. To get a list of subdirectories in a filesystem, `cd` to the pathname associated with that filesystem, and run the following pipeline:

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

This pipeline, which uses the *reverse* and *numeric* options of the `sort` command, pinpoints large directories. Use `ls -l` to examine the size (in bytes) and modification times of files within each directory. Old files, or text files over 100 Kbytes, often warrant storage offline.

Using the `vi` *Editor* 6

`vi` (pronounced "vee-eye," short for visual display editor) is the standard SunOS 5.2 text editor. Since `vi` is not window-based, this multipurpose editor can be used on any kind of terminal to edit a wide range of file types.

You can enter and edit text with `vi`, but it is not a word-processor. It was not created to 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 allow you to format `vi` text by inserting codes that are then interpreted by the emulator.

`vi` contains a huge array of commands, many of which have overlapping functions. At first, it's quite normal for new users to feel overloaded by this. The purpose of this chapter, however, is to provide you with an overview of the most essential `vi` commands. As you begin to use `vi`, you'll find that it is an extremely powerful text editor, and that it may take you a while to become proficient.

Note that there is a read-only version of `vi` called `view`. When you open a file with `view`, you can use `vi` commands, but you cannot write (or save) your changes. This allows you or others to read a `vi` file without accidentally changing it.

Starting `vi`

In the sub-sections that follow, you'll learn how to start `vi`, enter text in a file, save (write) the file, and quit `vi`. You'll also create a practice file that you'll use for the rest of this chapter.

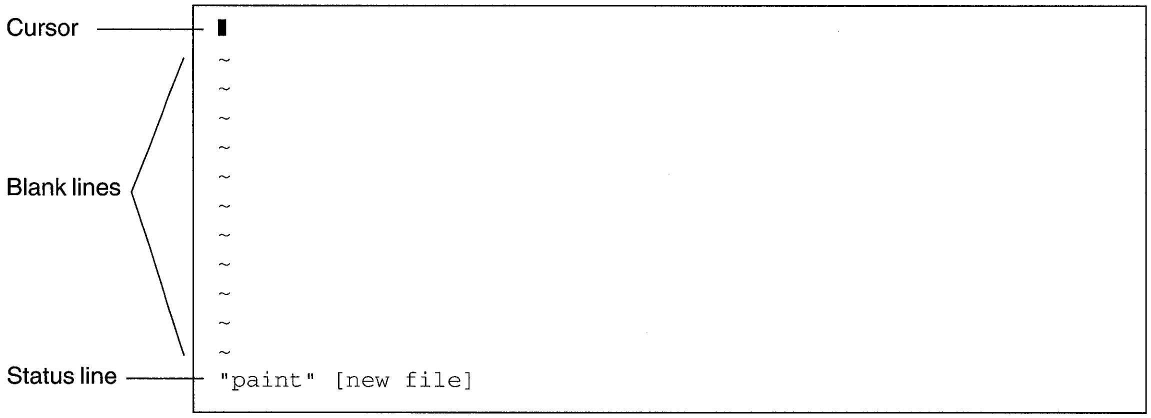
Creating a File

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

```
$ vi paint
```

If `paint` already exists, `vi` will open the existing file; if this is a new file, `vi` will create it. For the purposes of this example, `paint` should be a new file.

The `vi` editing screen appears in a moment:



```
▮
~
~
~
~
~
~
~
~
~
~
~
"paint" [new file]
```

Figure 6-1 The `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`.

The 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 our example, the status line indicates that it's a new file.

The Two Modes of `vi`

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

Since `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 enter 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 enter text in the sample file `paint`, type the `vi` "insert" command `i`. This takes `vi` out of 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. Later you will learn how to edit the text you entered.

When you finish entering text in `paint`, press Esc to return to command mode. The cursor moves back onto the last character entered. Now you can enter 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 (which is usually located next to the Esc key) acts as the Caps Lock. Pressing this key instead of Esc is a common error.

Note – Occasionally you may 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. This is similar to the OpenWindows Refresh command.

Command Mode

When you open a file with `vi`, you are in command mode. In this mode, you can enter commands to implement a wide range of functions. Most `vi` commands consist of one or two letters and an optional number. Usually, there are upper and lowercase versions of commands that 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 don't 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 preceded with a colon as a third, and uniquely separate mode of `vi`, *last-line mode*. This is 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 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 allows you to perform many file manipulation commands without ever leaving `vi`. See "Using ex Commands," 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 called the *buffer.* The permanent disk copy of the file is modified only when you *write* (save) the contents of the buffer.

This arrangement has its good and bad points. On the one hand, it means that you can quit a file and discard all the changes that you have made during an editing session, leaving the disk copy intact. On the other hand, you could lose the (unsaved) contents of the work buffer if the system crashes. (People on remote terminals connected by phone lines are especially vulnerable to unplanned interruptions.)

The best policy is to save your work frequently, especially when 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 more or less synonymous commands that control saving the buffer contents to a file and quitting `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
```

followed by Return.

Saving and Quitting

Save and quit by typing:

```
:wq
```

followed by 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 simply want to quit, type:

```
:q
```

followed by Return. If you have made changes, `vi` will not let you quit with `:q`. Instead, it will display the message, `No write since last change (:quit! overrides).`

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

```
:q!
```

followed by Return.

Printing a File

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

```
$ lp filename
```

where *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, "Using Printers," for more information on printer commands.

Basic `vi` Commands

The following sections explain several categories of `vi` commands. These include:

- Moving around in a file
- Inserting text
- Changing and substituting text
- Undoing changes to text
- Deleting text
- Copying and moving text
- 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 required to navigate within it. Open your practice file now, and try out each of the commands 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 could have radically 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 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 may not work correctly. This will depend on your terminal emulator. If the arrow keys don't work in your case, 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 can be an effective way to display more text in a very short file, but it can be tedious to move this way through a long file.

You may have noticed that moving the cursor either past the bottom or past the top of the screen has the effect of scrolling text up or down. This works for a very short file, but it is a tedious way to move 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 there is a fundamental difference between paging and scrolling. 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 some 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 (i.e., 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 works, position the cursor anywhere on a text line and type `A`. The cursor will move to the end of the line, where you can type your additions. Press Esc when you're done.

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 will move the cursor from any position on that line.) Again, as with all the commands in this section, press Esc to return to command mode after entering 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 enter 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 substituting 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`, enter 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.

Changing Part of a Line

To replace part of a line, place the cursor to the *right* of the portion to be saved. Type `C`, enter the correction, and press Esc. This changes the portion of the line from the current cursor position to the end of the line.

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 (there's no need to press Esc).

Transposing Characters

Correcting transposed characters takes just two keystrokes in `vi`. Suppose you find that you've typed "`teh`" when you meant to enter "`the`". Make the correction by putting the cursor over the first letter to be moved (in this case, `e`), and then type `xp`. The `e` and `h` will trade places – and `vi` will automatically return to command mode.

Breaking or Joining Lines

To break a line without affecting text, move the cursor to a space where you want the line to break and type `r` (for "replace") followed by Return. Note that if you type `r` with the cursor on a character and then press Return, that character will be replaced by the Return.

To join two lines, place the cursor on the upper line and type an uppercase `J`. (There's no need to press Esc after typing `J`.)

Undoing Changes

When editing text and making changes to a `vi` file, there will no doubt be times when you'll wish that you had not changed something. `vi`'s undo commands allow 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 once an operation is completed, you can undo your last command by pressing `u` immediately after the command. (There's no need to press Esc after typing `u`.) Pressing `u` a *second* time undoes 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. (There's no need to press Esc after typing `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.

Deleting Part of a Line

You can also delete part of a line.

To delete everything to the *right* of the cursor, position the cursor to the right of the part of the line you want to save, and type `D`.

To delete everything to the *left* of the cursor, position the cursor to the right of the part of the line you want to delete and type `d0` (`d`-zero).

Deleting to the End of the File

To delete everything from the current line to the end of the file, type `dG`. This also deletes the line on which the cursor is located.

Deleting from Beginning of File

To delete everything from the beginning of the file to the current line, type `d1G`. This also deletes the line on which the cursor is located.

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 using a combination of the `yank`, `delete`, and `put` commands.

Copying Lines

To copy 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, position the cursor anywhere on the line and type `yy`. Now move the cursor to the line above where you want the yanked line to be put (copied), and type `p`. A copy of the yanked line will appear 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, just type `11yy`. Eleven lines, counting down from the cursor, will be 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 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 will be 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 Named Buffers

To repeatedly insert a group of lines in various places within a document, you can yank (or delete) the lines into a named buffer. You specify named buffers by preceding a command with double quotes (`"`) and a name for the buffer. For example, to yank four lines into the named buffer *a*, type `"a4yy`. You can use several different buffers. For example, you might also delete text from one location and add it to several others. To delete 12 lines into the named buffer *b*, type `"b12dd`.

To insert the text, precede the `p` or `P` command with `"`*n*, where *n* is the named buffer. For example, to insert the lines saved in buffer *b*, type `"bP`.

You can overwrite named buffers with new lines. The buffers are saved until you exit `vi`.

When you use named buffers, you can safely delete and yank other text without affecting the lines you have already saved in the named buffers — unless, of course, you purposely overwrite the named buffer.

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`. This will all become evident as you learn the `vi` commands. In the section, "Summary of Basic vi Commands" at the end of this chapter, 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
```

and press Return.

When specifying 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 discussed above:

```
: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
```

and 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. It also has a powerful global replacement function.

Finding a Character String

A *character string* is simply one or more characters in a row. It may include letters, numbers, punctuation, special characters, blank spaces, tabs, or carriage returns. A string may be a grammatical word or it may 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 case, 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 will stop at its first occurrence. If the string is not found, vi will display 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 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.

These special characters can be used as commands to the search function, so if you want to search for a string including one or more of these characters, you must indicate this by preceding the 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
- Wild card 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
```

Since this is a search for a string and not a word, this search pattern may 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* will find strings beginning with either m or d. On the other hand, `/[d-m]`*string* will find strings beginning 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 discussed above. All the special matching characters for searches can be used in search-and-replace.

The basic command form is:

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

followed by the Return key.

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

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

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 by pressing Ctrl-C.

Going to a Specific Line

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

Editing a Series of Files

To edit a series of files, list the file names after the `vi` command when you start `vi` from the command prompt:

```
$ vi paint orwell
```

The files appear in the order in which they are listed. First `paint` appears. When you finish editing `paint`, type `:n` to go on to the next file, `orwell`. To go on to the next file without saving changes in the current file, type `:n!` instead of `:n`.

If you have a series of files with related names (for example, `test1`, `test2`, `test3`), you can use wildcard characters to specify a group of files:

```
$ vi test*
```

The files will appear for editing in alphabetical order by name.

Copying Lines Between Files

To copy lines from one file to another, do the following:

1. **Edit the first file.**
2. **Save the desired lines in named buffers, using the** `yank` **command. For example, to save 10 lines in buffer *a*, type** `a10Y`**.**
3. **Without exiting** `vi`**, edit the next file (**`orwell` **in this example):**

```
:n orwell
```

4. **Add the lines from the first file with the** `put` **command. For example, to put the contents of buffer *a* below the current cursor position, type** `ap`**.**

Remember that the contents of all named buffers are lost whenever you exit `vi`. Don't use the quit (`:q`) command until you have finished any operations involving named buffers.

Setting `vi` *Parameters*

`vi` has many variables that affect its behavior and appearance. You can view a list of these variables (with their current settings) while running `vi` by typing:

```
:set all
```

followed by Return.

Recovering from a Crash

If the system crashes, the contents of your buffer are at risk. Often, though, you can recover most of your work by restarting `vi` with the command form:

```
vi -r filename
```

where *filename* is the file you were editing at the time of the crash. The system will usually send you mail after the system is restarted, telling you there is a recover file.

Summary of Basic `vi` Commands

The following table provides a convenient reference for basic `vi` commands.

Table 6-1 Basic `vi` Commands

Command	**Meaning**
Starting `vi`	
`vi` *filename*	Open or create file
vi	Open new file to be named later
`vi -r` *filename*	Recover crashed file
`view` *filename*	Open file read-only
Cursor Commands	
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

Table 6-1 Basic `vi` Commands *(Continued)*

Command	Meaning
Return	Move down one line
Back Space	Move left one character
Space Bar	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
Inserting Characters and Lines	
a	Insert characters to right of cursor
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
Changing Text	
cw	Change word (or part of word) to right of cursor
c	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 (upper or lower)

Table 6-1 Basic `vi` Commands *(Continued)*

Command	Meaning
u	Undo previous command
U	Undo all changes to current line
:u	Undo previous last-line command
Deleting Text	
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)
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
Copying and Moving Text	
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
Setting Line Numbers	
:set nu	Show line numbers
:set nonu	Hide line numbers
	Setting Case-sensitivit
:set ic	Searches should ignore case
:set noic	Searches should be case-sensitive
Finding a Line	
G	Go to last line of file

Table 6-1 Basic `vi` Commands *(Continued)*

Command	**Meaning**
1G	Go to first line of file
21G	Go to line 21
Searching and Replacing	
/*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
Clearing the Screen	
Ctrl-L	Clear (refresh) scrambled screen
	Inserting a File into a File
`:r` *filename*	Insert (read) file after cursor
`:34 r` *filename*	Insert file after line 34
Saving and Quitting	
: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

Using Mail 7

SunOS 5.2 provides a program called `mailx` for sending and receiving electronic mail (*email*). `mailx` provides facilities 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 may prefer to use the window-based mail, you may 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're in the OpenWindows 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 this may 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 will affect your "in tray," thus confusing Mail Tool.

`mailx` *Basics*

In this section you will learn just enough about `mailx` to get by. In later sections you will learn about features and functions that will greatly 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've 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 will display the message:

```
No mail for username
$
```

where *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 may be able to use just your login on a local network—consult your system administrator when in doubt.

```
$ mailx rose@texas
```

The program will respond with a `Subject:` line:

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

If you like, type in 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 backspacing 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, give the `mailx` command again. Your screen will probably 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," in this chapter.)

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 its number at the `mailx` prompt, the ampersand (`&`), 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're finished using `mailx`, you can quit the program using one of two commands: `q` (quit) or `x` (exit).

If you type `q` at the `mailx` prompt and then press Return,

```
& q
```

you will then see a message similar to the following:

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

where *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,

```
& x
```

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 there's no mail waiting for you, you will see the message:

No mail for *username*

Otherwise, you will 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 order in which letter was received.
- *Sender*: indicates name of user (and usually machine) letter came from.
- *Time*: indicates date and time letter was sent.
- *Size*: indicates number of lines/number of characters in letter.
- *Subject*: indicates sender-designated subject of letter.

When you have a large number of letters in your mailbox, the displayed list may not show all of your mail. If this is the case, 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. Pressing Return again will display the next letter. To read any letter in the list, type its number and press Return.

Deleting (and Undeleting) Letters

When you are finished reading a letter, you may decide you want to delete it rather than have it saved to your `mbox` file (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, give this command from within `mailx`:

```
& d 2
```

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

```
& d 1 3
```

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

```
& d 1-3
```

Before quitting `mail`, 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, give this command:

```
& u 2
```

If you want 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`, typing `u` will undelete messages `2`, `3`, `4`, and `5`.

Note that all deletions are made permanent when you quit `mailx` with the `q` command; that is, deleted letters can no longer be undeleted. You can, however, quit `mailx` with the `x` command, leaving your mailbox intact — as mentioned previously, quitting with `x` will 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 this, use the command form:

|*number* `lp`

at a `mailx` prompt. (The | symbol is called a *pipe*.) For example, to print a copy of letter 2, type:

```
& |2 lp
```

and 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" in Chapter 2, "Basic SunOS Commands."

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.

Typing the `who` command lists all the users who are currently logged in to the file server you are on. The displayed list contains user's 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
```

Typing the `finger` command displays the same type of information as `who` with a little more detail. The information that appears depends on how your system administrator has set up this command. As an example, you may 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 currently logged in to your local network. Refer to Chapter 9, "Using the Network," for instructions regarding 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
```

where *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. When you want 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 will freeze. If this 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`

and the letter is 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." (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.

There are 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 will now display the carbon copy prompt (`Cc:`) after the subject prompt. Enter the addresses of the users you want to receive carbon copies. Separate multiple addresses with spaces.

- When you've finished typing 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
```

- A `Cc:` line is also created by 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. Blank lines can be filled in; filled lines can be retyped. 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," in this chapter.

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

The command form to insert a letter is

`~m` *number*

where *number* is the number of the letter to be inserted. For example, to send a letter to another user that includes a copy of letter number 3 from your mailbox list, you would do the following:

1. **On a new line give the command `~m 3` and then press Return.**
2. `mailx` **displays the message,** `Interpolating: 3 (continue)`
3. **You won't see the text of message `3`, but the recipient will. You can go on with your letter after `(continue)`, or you can send it as is.**
4. **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. Use the command form:

`~r` *filename*

as you're writing a letter. For example, to insert the file `outline` in the current letter, type:

```
~r outline
```

Replying to a Letter

Reply to mail by giving the command

`r` *number*

at a `mailx` prompt. (If you omit the letter number, `mailx` replies to the current letter.) For example, to reply to the sender of letter 2, give the command:

```
& r 2
```

`mailx` automatically addresses your letter and supplies an `Re: Subject:` line that echoes the original `Subject:` line. Send your reply like 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, just give the command `~m` without a letter number.

Saving and Retrieving Letters

In addition to sending and receiving letters, you may also want to 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 below.

`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, the command form at the `mailx` prompt is:

`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`, you would type:

```
& 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 dispense with typing full pathnames to files if you save or copy letters to mail folders. Folders are special files that are stored in a folder directory.

The advantages to saving or copying letters to folders is that your letters are automatically kept together in the same directory, where they are easily accessible without typing long pathnames.

Setting the Folder Directory

To use folders, you must first set up a folder directory. This is a two-step process:

1. **First, make the directory using the `mkdir` command.**
   For example, if you wanted your folder directory to be called `Letters`, you would first make the directory:

```
$ mkdir Letters
```

2. **Second, use a text editor to edit the `.mailrc` file in your home directory (which contains `mailx` options) to set the folder directory path.**
   Here you need to edit the `set folder` variable to include the full path name of your newly created folder directory. For example:

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

   or, using the C shell shortcut ~ to specify your home directory.

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

Now your folder directory is set to receive letters saved in folders. (The change to the `.mailrc` file will take effect 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 meaning "save letter 3 into `~/Letters/projects`." (If the folder doesn't already exist, `mailx` will create 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 saved in a file, use the command form:

`mailx -f` *filename*

Using the example above, you would read the file `~/memos/finance` by typing:

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

You can read letters saved in a folder with a similar command—just use the + instead of a pathname. 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 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 pressing Return.

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

```
& %
```

To return to the previous folder, type:

```
& #
```

Using `vi` *with* `mailx`

You can use the `vi` text editor to compose letters while running `mailx`. This gives you the capability of correcting mistakes and adding and deleting text before you send your letters. If you are not already familiar with using `vi`, refer to Chapter 6, "Using the vi Editor," 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. **Give the `mailx` command with an address at either the `mailx` prompt (`&`) or the command prompt (`$`).**
2. **Type in the subject at the `Subject:` line. Press Return.**
3. **Start `vi` by giving the command `~v` on a new line. The `vi` screen will appear, representing an empty file in your `/tmp` directory.**
4. **Use `vi` commands to input and edit the body of your letter.**
5. **When done, quit `vi` with the command `:wq` or `ZZ`.**

After you quit `vi`, `mailx` displays the message `(continue):` here 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 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 wanted to send mail from time to time 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.

There are two locations where you can set up mail aliases:

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

Mail aliases 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 tried to send mail to `amigos`, he would receive 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 sent mail to `amigos`, your mail arrives as though you had typed everyone's names as recipients. It is not apparent to the recipients that you used a mail alias to send the mail.

`.mailrc` is located in your home directory. This file contains a number of 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 example above shows the method for using the `vi` editor to edit the file. If you are not already familiar with `vi`, refer to Chapter 6, "Using the vi Editor," for instructions.

Each mail alias is contained on one line of the file; that is, it 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 example below shows two mail aliases. The first (`amigos`) contains three people. The second (`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, just address it to the mail alias name. Do *not* include your machine name. For example, if you sent the following:

```
$ 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 sent mail to `softball@`*machinename*, that's how the mail will read when it is received. The recipients will know what the mail alias is, but not necessarily who else is on it.

The format of mail aliases 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 example above shows the method for using the `vi` editor to edit the file. If you are not already familiar with `vi`, refer to Chapter 6, "Using the vi Editor," 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 using the name of the alias and your machine name. For example, if 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 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 created in `.mailrc` and those 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 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 will be displayed.

Table 7-2 Tilde Commands (`mailx`)

Command	Function
~!*command*	Escapes to a shell command
~.	Simulates pressing Ctrl-D to mark end of file
~?	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 sending a message while reading 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 sending a message while reading mail.
~p	Prints the message being entered to the screen.
~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. Likewise, when in input mode (for example, when writing a letter), you can give 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, give the command:

```
$ man mailx
```

or refer to the *SunOS 5.2 Reference Manual.*

Using Printers 8

The LP (for *line printer* subsystem) print service is the portion of SunOS 5.2 which provides the tools used for printing. It provides a wide variety of functions, many of which are not within the scope of this manual. This chapter provides only the procedures necessary for you to perform the following basic printing tasks using the LP print service:

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

See *SunOS 5.2 Setting Up User Accounts, Printers, and Mail* for a complete description of the LP print service.

Submitting Print Requests

To print a file from the command prompt, you 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 the LP print service is set up with a default printer, you can submit print requests as follows without typing the name of the printer:

```
$ lp filename
```

where *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/passwd` file, type:

```
$ lp /etc/passwd
request id is pinecone-8 (1 file)
$
```

See *SunOS 5.2 Setting Up User Accounts, Printers, and Mail* 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:

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

where *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/passwd file on the printer acorn, type:

```
$ lp -d acorn /etc/passwd
request id is acorn-9 (1 file)
$
```

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 thorn /etc/passwd
UX:lp: ERROR: Destination "thorn" is unknown to the
              LP print service.
$
```

See *SunOS 5.2 Setting Up User Accounts, Printers, and Mail* for information on configuring printers. See "Determining Printer Status," in this chapter, 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 may 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
```

where *filename* is the name of the file you're printing.

Printing Multiple Copies

You can print more than one copy of a file. When you request more than one copy, the file is printed the number of times you specify using the `-n` option to the `lp` command. The print request is considered as one print job, and only one header page is printed.

Enter the following to request multiple copies:

```
$ lp -nnumber filename
```

where *number* is the desired number of copies and *filename* is the name of the file you are printing.

For example to print four copies of the `/etc/passwd` file:

```
$ lp -n4 /etc/passwd
request id is pinecone-9 (1 file)
$
```

Summary Table of `lp` Options

You can customize your print request using options to the `lp` command: specifying forms, character sets, filters, titles, banners, and so forth. 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, you would enter the following:

```
$ lp -d printername -m -n6 filename
```

where *printername* is the name of the desired printer and *filename* is the name of the file you are printing.

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.
-h	Header. Puts a header on each page of the print request.
-c	Copy. Copies the file before printing.
-w	Write. Writes a message to your terminal when the file has printed successfully.

See the *SunOS 5.2 Reference Manual* for a complete list of options.

Determining Printer Status

Use the `lpstat` command to find out about the status of the LP print service. You can check on the status of your own jobs in the print queue, determine which printers are available for you to use, or determine request ids of your jobs if you want to cancel them.

Checking on the Status of Your Print Requests

Enter the following to find out the status of your own spooled print requests:

```
$ lpstat
```

A list of the files that you have submitted for printing is displayed.

In the following example, on the system pine, one file is queued for printing to the printer pinecone:

```
$ lpstat
pinecone-10            fred           1261   Mar 12 17:34 on pine
$
```

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 on Available Printers

To find out which printers are configured on your system, type the following:

```
$ lpstat -s
```

The status of the scheduler is displayed followed by the default destination and a list of the systems and printers that are available to you.

In the following example, on the system elm, the scheduler is running, the default printer is pinecone, and two network printers, pinecone and acorn, are available:

```
$ lpstat -s
scheduler is running
system default destination: pinecone
system for pinecone: pine
system for acorn: oak
$
```

Displaying All Status Information

The -t option for lpstat gives you a short listing of the status of the LP print service.

To display a short listing of all status information, type the following:

```
$ lpstat -t
```

All available status information is displayed.

In the following example, there are no jobs 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: pinecone
system for acorn: oak
pinecone accepting requests since Wed Jan  2 18:20:10 PST 1991
acorn accepting requests since Mon Mar  4 15:53:47 PST 1991
printer pinecone is idle. enabled since Wed Jan  2 18:20:22 PST
1991. available.

printer acorn is idle. enabled since Mon Mar  4 15:53:44 PST 1991.
available.
$
```

Displaying Status for Printers

You can request printer status information for individual printers using the `-p` option to `lpstat`. This option shows whether the printer is active or idle, when it was enabled or disabled, and whether it is available to accept print requests.

To request status for all printers on a system, type the following:

```
$ lpstat -p
```

In the following example, two printers are idle, enabled, and available. If one of those printers had jobs in the print queue, those jobs would also be displayed.

```
$ lpstat -p
printer pinecone is idle. enabled since Wed Jan  2 18:20:22 PST
1991. available.
printer acorn is idle. enabled since Mon Mar  4 15:53:44 PST 1991.
available.
$
```

To request status for an individual printer by name, type the following:

```
$ lpstat -p printername
```

where *printername* is the name of the specific printer.

Displaying Printer Characteristics

If you want to see all of the characteristics for a printer, use the `-p` option together with the `-l` (long) option to `lpstat`. This command can be especially useful for finding the printer type and content type.

To show characteristics for all printers on a system, enter the following:

```
$ lpstat -p -l
```

A table shows all the configuration information that is used by the LP print service for each printer.

In the following example, all of the fields are blank except for the content type and the printer type of the printer pinecone.

```
$ lpstat -p pinecone -l
printer pinecone is idle. enabled since Wed Jan  2 18:20:22 PST
1991. available.
        Content types: PS
        Printer types: PS
        Description:
        Users allowed:
                (all)
        Forms allowed:
                (none)
        Banner not required
        Character sets:
                (none)
        Default pitch:
        Default page size:
$
```

Summary Table of `lpstat` *Options*

You can request different types of printing status information using the `lpstat` command. 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 (-).

For example, to show a long list of status for an individual printer, you would enter the following:

```
$ lpstat -p printername -l
```

where *printername* is the name of the printer for which you want to view status.

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 output.
`-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 *SunOS 5.2 Reference Manual* 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 its 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` 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:

```
$ cancel requestid
```

where *requestid* is the request id number of the desired print request.

A message is displayed telling you that the request is canceled. The next job in the queue begins printing.

In the following example two print requests are canceled:

```
$ cancel pinecone-3 pinecone4
request "pinecone-3" cancelled
request "pinecone-4" 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 as follows:

```
$ cancel printername
```

where *printername* is the name of the printer to which you sent the request.

A message is displayed telling you that the request is canceled. The next job in the queue begins printing

In the following example the currently printing request has been canceled:

```
$ cancel pinecone
request "pinecone-3" cancelled
$
```

Your system administrator can log in as `root` or `lp` and cancel the currently printing request using the printer name as the argument for the `cancel` command.

Using the Network 9

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 affecting other machines from your own workstation.

In this chapter, the following information is provided:

- General concepts of networking
- 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

If the machine you're using is not currently attached to a network, the information presented here may not be relevant to your situation. However, it may be valuable for you to at least skim this information to get an overall view of the benefits that networking can provide.

Networking Concepts

A network connection between machines allows them to transmit information to one another. Networks are often referred to as being *local area networks* (LANs), which range over small areas, generally less than a few thousand feet; *wide area networks* (WANs), which can span thousands of miles; or *campus area networks* (CANs), which are intermediate in size.

A network comprised of a linked group of networks is called an *internetwork*. For example, your machine may be part of a network within your building and part of an internetwork that connects your local network to similar networks across the country. Since the difference between a network and an internetwork is generally invisible to the user, the term "network" is used in this manual to refer to both networks and internetworks.

Machines attached to a network communicate using a *network protocol*, or common network language, to ensure that information is transmitted to the appropriate locations. An *internetwork protocol*, sometimes referred to as a *relay*, links networks together.

Logging In Remotely (`rlogin`)

The `rlogin` command allows you to log in to other UNIX machines on your network.

To remotely log in to another machine, type:

```
$ rlogin machinename
```

where *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 other machine "trusts" your machine name and won't require you to type the password.

```
$ rlogin lonesome
Password: (type password)
Last login: Mon Jan 6 09:37:55 from blue
Sun Microsystems, Inc.   SunOS 5.1      October 1992
(The following commands done on lonesome.)
$ pwd
/home/keithp
$ logout
Connection closed.
$
```

rlogin *without a Home Directory*

In the example above, user keithp logged in to lonesome at the directory /home/keithp, as indicated by the pwd command. When you log in to a machine where you don't have a home directory, rlogin displays a message stating that you have no home directory on the remote machine and logs you in to the root (/) directory of that machine:

```
$ rlogin fretful
Password:
No directory! Logging in with home=/
Last login: Fri Jan 3 10:21:59 from blue
Sun Microsystems, Inc.    SunOS 5.1      October 1992
(The following commands done on fretful.)
$ pwd
/
$ logout
Connection closed.
$
```

`rlogin` *as Someone Else*

There may be times when you want to log in to a remote machine as someone else. For example, if you're working on someone else's machine (using their username) and you want to log in to your own machine as yourself. The `-l` option to `rlogin` allows you to do this. The command syntax is:

`rlogin` *machinename* `-l` *username*

For example, the following shows how user `keithp` on machine `blue` would log in to machine `lonesome` as `earl`:

```
$ rlogin lonesome -l earl
Password:
Last login: Wed Jan 8 07:12:25 from blue
Sun Microsystems, Inc.    SunOS 5.1      October 1992
(The following commands done on lonesome.)
$ pwd
/home/earl
$ logout
Connection closed.
$
```

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 whose name isn't known to your machine, `rlogin` searches unsuccessfully through the hosts database and then displays a notification as follows:

```
$ 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 for some reason you can't terminate a session in this manner, you can abort the connection by typing a tilde character followed by a period (`~.`) at the beginning of a line. The 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 in the series, you are returned to your original machine:

```
$ rlogin dakota
Password:
Last login: Fri Jan 10 09:14:43 from blue
Sun Microsystems, Inc.    SunOS 5.1      October 1992
(The following command done on dakota.)
$ ~.   (You may not see the ~ on the screen.)
Connection closed.
$
```

If you want to back up to an intermediate rlogin connection, use two tildes followed by a period (`~~.`) as follows:

```
$ rlogin lonesome
Password:
Last login: Tue Jan 7 08:12:49 from blue
Sun Microsystems, Inc.    SunOS 5.1      October 1992
(The following command done on lonesome.)
$ rlogin dakota
Password:
Last login: Tue Jan 7 10:17:40 from lonesome
Sun Microsystems, Inc.    SunOS 5.1      October 1992
(The following command done on dakota.)
$ ~~.   (You may not see the ~~ on the screen.)
Connection closed.
$
```

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`. Alternatively, you can type the percentage sign (`%`) followed by the process number of the stopped process (the default for `%`, if no process number is included, is the process most recently suspended).

```
$ rlogin lonesome
Password:
Last login: Tue Jan 7 08:12:49 from blue
Sun Microsystems, Inc.    SunOS 5.1      October 1992
(The following command done on lonesome.)
 ~^Z    (You may not see the ^Z on the screen.)
Stopped
(The following command done on blue.)
$ pwd
/home/keithp
$ %
rlogin lonesome

(The following command done on lonesome.)
$ logout
Connection closed.
$
```

Similar to aborting `rlogin` with `~~.`, typing two tildes and Ctrl-Z suspends the current `rlogin` and places you at an intermediate `rlogin`.

Verifying Your Location (`who am i`)

After logging in to a variety of remote machines, perhaps under different login names, you might need to verify exactly where you are. Typing `who am i` displays the name of the machine you're currently logged into as well as your current identity.

Type `man rlogin` at the command prompt or refer to the *SunOS 5.2 Reference Manual.*

Copying Files Remotely (rcp)

The rcp command allows you to copy files from one machine to another. It 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 that used for cp.

Note – To copy subdirectories and their contents from one machine to another, use rcp -r.

Copying from Another Machine to Yours

To copy from a remote machine to your machine, the syntax is:

rcp *machinename:source destination*

where *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 /home/dakota/doc/letter from the remote machine dakota to the /tmp directory on local machine blue:

```
$ rcp dakota:/home/dakota/doc/letter /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 blue, you would type the following:

```
$ rcp fretful:~hank/*.doc .
$
```

Copying from Your Machine to Another

To copy from your local machine to a remote machine, the syntax is reversed as follows:

`rcp` *source machinename:destination*

where *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 you would copy the file `austin` from your directory `~/usa/texas` to the directory `~hank/cities` on the remote machine `fretful` (remember that ~ is your home directory and `~hank` is user `hank`'s home directory):

```
$ rcp ~/usa/texas/austin fretful:~hank/cities
$
```

For more information on the `rcp` command and its options, refer to the *SunOS 5.2 Reference Manual.*

Executing Commands Remotely (`rsh`)

The `rsh` command (for *remote shell*) lets you execute a single command on a remote machine without having to log in formally. It can be a real time-saver when you know you only want to do one thing on the remote machine.

To execute a command on a remote machine, type:

`rsh` *machinename command*

The following example shows how you would view the contents of the directory `/home/lonesome/guitar` on the machine `lonesome`:

```
$ rsh lonesome ls /home/lonesome/guitar
collings        gibson        santacruz
fender          martin        taylor
$
```

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.

For more information on the `rsh` command and its options, refer to the *SunOS 5.2 Reference Manual.*

Viewing User Information (`rusers`)

The `rusers` command (for *remote users*) shows you who's logged on to other machines on your network. Typing the `rusers` command by itself displays each machine on the network and the user(s) logged in to them, as follows:

```
$ rusers
aspen           susan
blue            keithp
dakota          sally
farmhouse       elmer
freeway         lindab     johnj     karenm
fretful         hank
lonesome        george
twister         tex
$
```

Notice that machine `freeway` has three different users currently logged in.

To display information on a specific remote machine, type the `rusers` command followed by the name of the machine, as follows:

```
$ rusers freeway
freeway         lindab     johnj     karenm
$
```

The `-l` option to the `rusers` command provides more detailed information, including user names, machine and terminal names, the time each user logged in, how long each user's been idle (if more than one minute), and the name of the machine that each user logged in from (if any):

```
$ rusers -l freeway
lindab       freeway:ttyd8     Feb 10 08:12     5:29
johnj        freeway:console   Feb 10 09:16
karenm       freeway:ttyp0     Feb 10 11:56       36
$
```

You can also use the `-l` option without providing a machine name.

For more information on the `rusers` command and its options, refer to the *SunOS 5.2 Reference Manual.*

Customizing Your Working Environment 10

The SunOS 5.2 operating system makes it possible for you to control and adjust many aspects of your working environment. You do this by modifying the *environment variables* contained in your system's *initialization files*. When you login 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 how to customize your system by modifying your initialization files and setting the most common environment variables. It also describes how to alias SunOS commands, how to change your system prompt, how to set default file permissions, and how to customize OpenWindows fonts.

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 SunOS 5.2, but you can also use the C shell or Korn shell. Each of these shells has its own initialization file (or files).

If you're not sure which shell is your default shell (referred to as your *login shell*):

1. **Type** `echo $SHELL`**:**

```
$ echo $SHELL
/bin/sh
```

2. **Look at the output of the command. If it is:**

- `/bin/sh` – your login shell is the Bourne shell
- `/bin/csh` – your login shell is the C shell
- `/bin/ksh` – your login shell is the Korn shell

Regardless of the shell you are using, when you first login 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 OpenWindows environment your system checks this file (or set of files) each time you start a new Shell Tool or Command Tool 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)
- `.login` and `.cshrc` (for the C shell)

Your user profile file(s) is located in your home directory and allows you to configure your working environment to your preference.

Environment Variables

Your system sets up your system environment using a set of specifications 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 `.profile`, `.login`, or `.cshrc` files.

To display the environment variables currently set on your system:

1. Type the `env` **command and press Return:**

```
$ env
HISTORY=100
HOME=/home/texas/keith
HZ=100
LOGNAME=keith
MAIL=/var/mail/keith
MANSECTS=\1:1m:1c:1f:1s:1b:2:\3:3c:3i:3n:3m:3k:3g:3e:3x11:3xt:3
w:3b:9:4:5:7:8
PATH=/usr/bin
SHELL=/bin/sh
TERM=sun
TZ=EST5EDT
```

Note – You can also use the `env` command to identify your login shell. It is specified in the `SHELL` environment variable. In the example above, the shell is set to `/bin/sh` (the Bourne shell).

The User Profile

This section describes some of the more commonly used environment variables. Many of these variables may already be in your user profile. As previously mentioned, your user profile file (`.profile` for the Bourne and Korn shells and `.cshrc` for the C shell) is located in your home directory.

Note – Hidden ("dot") files can be listed by typing `ls -la`.

The following is a partial list of environment variables that can be included in your user profile. The syntax for defining environment variables depends on the shell you're using:

- `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 (for the C shell only).
- 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 *SunOS 5.2 Setting Up User Accounts, Printers, and Mail* for information on the passwd database.
- LPDEST – Defines your default printer.
- MAIL – Specifies the path to your mailbox. usually located in the /var/mail/*username* directory, where *username* is your login name. See Chapter 7, "Using Mail," for more information on this file.
- MANSECTS – Sets the available sections of on-line 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 enter 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 .profile file (Bourne or Korn shell), or .cshrc file (C shell) as part of the login process.
- PS1 – Defines your command prompt. The default prompt for the Bourne and Korn shells is the dollar sign ($). The default prompt for the C shell is the percent sign (%). The default prompt for root in either shell is the pound sign (#).
- SHELL – Defines the shell 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 *SunOS 5.2 Setting Up User Accounts, Printers, and Mail* 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. It 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 timezone 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 enter the name of a command.

For example, if you have no PATH variable set and you want to copy a file, you need to enter the full pathname 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 named in the PATH variable, and executes it when it is found. Using the PATH variable to list the commonly used SunOS command directories can thus significantly streamline your work.

For the Bourne and Korn shells, the PATH variable is specified in your .profile file (in your home directory) using the following syntax:

```
PATH=.:/usr/bin:/home/bin
```

where *home* represents the path name of your home directory.

For the C shell, the PATH variable is specified in your .cshrc file (in your home directory) using the following syntax:

```
set path=(. /usr/bin home/bin)
```

where *home* is the path name of your home directory.

Note – In the C shell 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 shell, use the source command to make the changes effective in your current window without having to logout:

```
example% source .cshrc
```

If you are running the Bourne or Korn shell, type the following to make the changes effective in your current window without having to logout:

```
$ . .profile
```

Aliases (C Shell Only)

Aliases are useful shortcuts for commands you often type. For example, the default setting for the remove command (rm) does not ask for confirmation before removing files. Sometimes this is inconvenient, as a typing error can remove the wrong file. However, the C shell lets you use the alias variable to change this by adding the following line to your .cshrc file:

```
alias rm  'rm -i'
```

With this line in the .cshrc, typing rm will now be 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 example above are necessary to include the blank space between rm and -i. Without them the C shell cannot correctly interpret the text after the space.)

To make your changes to the .cshrc file effective immediately in your current window, use the source command. The source command causes the system to read the current .cshrc file and execute the commands in it:

```
example% source .cshrc
```

Changing Your Command Prompt

The syntax you use to change your command prompt depends on whether you are using the Bourne, Korn or C shell.

Bourne and Korn Shells

For the Bourne or Korn shells, you redefine your command prompt with the PS1 command. The following are three examples:

```
PS1=": "
PS1="'hostname': "
PS1="'hostname'{'id'}: "
```

- The first example sets the prompt to a colon (:), followed by a space.
- The second example creates a prompt consisting of your machine name followed by a colon and a space.
- The third example sets the prompt to your machine name, followed by your login name in braces { }, a colon, and a space.

Type any of the examples above to change your current command prompt. The prompt will remain until you change it again, or logout.

If you want to make your changes more permanent, add one of the above examples (or a prompt of your own creation) to your `.profile` file. If you do this, the prompt you specify will appear each time you login in or start a new shell.

C Shell

For the C shell, you personalize your command prompt with the `set prompt` command. The following are three examples:

```
set prompt="% "
set prompt="'hostname'\!: "
set prompt="'hostname'{'id'}: "
```

- The first example sets the prompt to the percent sign, followed by a space.

- The second example creates a prompt consisting of your machine name followed by the history number of the command (hostname1, hostname2, hostname3, and so on).
- The third example sets the prompt to your machine name, followed by your login name in braces, a colon, and a space.

Type any of the examples above to change your current command prompt. The prompt will remain until you change it again, or logout.

If you want to make your changes more permanent, add one of the above examples (or a prompt of your own creation) to your .cshrc file. If you do this, the prompt you specify will appear each time you login in or start a new shell.

Other Useful Variables

There are many other variables which you can set in your .profile or .cshrc files. For a complete list, refer to the *SunOS 5.2 Reference Manual*. The following are a few brief descriptions of some of the more commonly used options.

Use set noclobber to prevent unintentional overwriting of files when you use the cp command to copy a file. This variable affects the C shell only. Enter the following in your .cshrc file:

```
set noclobber
```

Use set history to set the number of commands saved in your history list. The history command is useful to view commands you have previously entered. The history file can also be used to repeat earlier commands. This variable affects the C shell only. Enter the following in your .cshrc file:

```
set history=100
```

You can also affect the Bourne and Korn shells in the same manner by placing the following in your .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`.cshrc` and `.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 used to calculate the code for `umask` is distinctly different from the method for `chmod`.

To begin with, 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)

and all directories 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 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 created for each digit of the umask command's numeric code.

Table 10-1 Permissins 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 *SunOS 5.2 Reference Manual.*

Customizing OpenWindows Fonts

If you choose, you can customize the size and style of the fonts displayed in your OpenWindows applications. The following sections describe how to customize these fonts.

Specifying the Font Style and Point Size

The default font for windows is Lucida Sans in 12 point (`medium`); the default font for window headers is Lucida Sans Bold. If you prefer, you can specify another font style and size for windows and window headers. You can make the change for a single window or you can make a permanent change for all your applications with Workspace Properties. The following subsections describe each of these options.

Fixed-Width and Proportionally-Spaced Fonts

Note that there are two general categories of fonts—*fixed-width* and *proportionally-spaced*. Each character in a fixed-width font takes up the same amount of space as every other character. By contrast, the characters in a proportionally-spaced font require varying amounts of space, depending upon their individual width. Proportionally-spaced fonts are more pleasing to the eye. However, some applications (such as Command Tool, Shell Tool, and `xterm`, a popular terminal emulator application) work best with fixed-width fonts.

Choosing Between Fixed and Proportional Fonts

Note that the default font displayed in Command Tool and Shell Tool is a proportionally-spaced font. Although this font is pleasing to the eye, problems occur in character alignment (when spacing and tabbing) with any proportionally-spaced font in terminal windows. If the spacing and tabbing character alignment are a problem for you, it is best to choose a fixed-width font for these windows. In the examples that follow, only fixed-width fonts are used for terminal windows; the examples for other windows and headers use proportionally-spaced fonts.

Specifying the Font for a Single Window

This section describes how to open a single application with a modified font style and point size. Note that changes cannot be made to existing windows; you must start a new application to display the new font. To start a new application, you type its application name on a command line.

The basic command, shown below, specifies the application name, the `-fn` (font name) option, and the font style and size. The ampersand (`&`) returns your system prompt to the window after you type the command; this enables you to continue using that window.

```
$ application -fn fontstyle-pointsize &
```

The following are examples of how to use the command to open an application with a specified font style and size.

- The example below starts a new Command Tool with the proportionally-spaced font, `Lucida Sans Typewriter Bold`.

  The point size is not specified; therefore the default (`12`-point) is used.

```
$ cmdtool -fn lucidasans-typewriter-bold &
```

- The example below starts a new Shell Tool with `Lucida Sans Typewriter Bold` and increases the size of the font from `12` point to `14` point.

  Note that when you change the size of the font, the size of the window changes as well.

```
$ shelltool -fn lucidasans-typewriter-bold-14 &
```

- The example below starts a new `xterm` terminal window with the font `terminal-bold` in `16` point:

```
$ xterm -fn terminal-bold-16 &
```

- The example below starts a new Text Editor with the font `Helvetica Bold` in `14` point:

```
$ textedit -fn helvetica-bold-14 &
```

Use the `-fn` option with any application and any font style and size you choose. The section "The Available Font List," in this chapter, describes how to list all the fonts available for OpenWindows applications. The section "Choosing a Font from the List," also in this chapter, introduces several demo programs that help you view fonts before using them.

Making Font Assignments Permanent

If you find that you are repeatedly running applications with customized fonts, you might like to add the customization to your workspace menu. You can do this using the Programs Menu category of Workspace Properties. This will save you the effort of typing the command-line options every time. For example, if you often want to run the text editor with a larger point size, you could add the following command line to the programs menu:

```
textedit -fn lucidasans-typewriter-14
```

You can have more than one instance of the same application in your programs menu if you want them to have different font sizes. This is useful if you run an application at a variety of different point sizes. For instance, you may want to have the option of running a text editor using `12`, `14`, or `18` point fonts. You would add the following commands to your programs menu:

```
textedit -fn lucidasans-typewriter-12
textedit -fn lucidasans-typewriter-14
textedit -fn lucidasans-typewriter-18
```

Once you have customized your programs menu from Workspace Properties in this way, you can invoke the text editor at any of these point sizes simply by selecting the appropriate item from your programs menu.

Note – Command lines added to the programs menu should not be followed by an ampersand (&).

Listing the Available Fonts

You may want to experiment with more fonts than have been shown in the previous examples, and you may want to apply them to other OpenWindows applications. To do this you first list the available fonts and then select them.

The Available Font List

You can see the entire list of available fonts by entering `xlsfonts` at the prompt in a terminal emulator window. It is best to use Command Tool to display the list because it is likely that the list will scroll off the top of the screen, and Command Tool has a scrollbar that will let you view the entire list.

Note – The list generated from `xlsfonts` is very long; there are over 400 fonts available. If the listing on your screen does not contain the expected number of fonts, check with your system administrator. It is possible that a subset of the available fonts was installed.

Each font has a long name in addition to a shortened version. The full name for `lucidasans-typewriter`, for instance, is:

```
-b&h-lucida sans typewriter-medium-r-normal-sans-12-120-72-72-m-
0-iso8859-1
```

The fonts you see in the `xlsfonts` listing are the long names followed by their short names. For the purposes described in this chapter, just use the short names.

Once you have chosen a font, follow the instructions in "Specifying the Font Style and Point Size," in this chapter, to customize the fonts in your application windows.

Choosing a Font from the List

Note – The following notation describes the order in which you select the submenus on the Workspace Programs menu to carry out the operation you want.

There are three demonstration programs that enable you to see the available font styles and sizes on your screen:

- `Fontview`

  `Fontview` is a demonstration program available in the menu under Workspace ➤ Programs ➤ Demos. When the demo appears, position the pointer in the demo window and press `MENU` to make selections.

- `Text`

  `Text` is a demonstration program available in the menu under Workspace ➤ Programs ➤ Demos. When the demo appears, position the pointer in the demo window and press `MENU` to make selections.

- `xfontsel`

  To run `xfontsel`, type `xfontsel &` at the command line.

```
$ xfontsel &
```

Migrating to OpenWindows Version 3.2 A

In some cases you may be running a version of the user environment software that is no longer current with Solaris 2.2, which runs OpenWindows Version 3.2 as its default user environment. For example, you may be running the SunView user environment software, or OpenWindows Version 2. If so, you may want to upgrade to OpenWindows Version 3.2. This appendix describes how to do this.

Note – The SunView software is no longer supported in OpenWindows Version 3.2. Unlike previous versions of OpenWindows, once you migrate to Version 3.2 you no longer have the option of also running SunView.

Migrating from the SunView Environment

If you are migrating to the OpenWindows environment from the SunView environment, the following information may make your transition easier.

The `.defaults` *and* `.Xdefaults` *Files*

To customize your OpenWindows environment in the same way as your SunView environment, you can convert your `.defaults` file (used by the SunView software) into an `.Xdefaults` file (used by the OpenWindows software). If you have a `.defaults` file in your home directory, you should run the `convert_to_Xdefaults(1)` program in your home directory, as follows:

```
$ cd
$ $OPENWINHOME/bin/convert_to_Xdefaults .defaults
```

This creates an `.Xdefaults` file in your home directory that is used to customize your OpenWindows environment when you start the software.

Migrating from the OpenWindows Version 2 Environment

Read this section carefully if you formerly ran the OpenWindows Version 2 environment and now want to migrate to Version 3.2.

The `OPENWINHOME` *Environment Variable*

If you are currently running a version of the OpenWindows software earlier than Version 3.2, you may have set up your system to use the `OPENWINHOME` environment variable. It is no longer recommended that users set the OPENWINHOME environment variable, either by-hand or from a start-up file.

When you run the openwin command it automatically sets the `OPENWINHOME` environment variable to `/usr/openwin`; therefore, you do not need to do it.

If you have set the `OPENWINHOME` environment variable in either the `.profile` or `.cshrc` file in your home directory, comment out the line or delete it altogether before running OpenWindows Version 3.2.

To remove, or comment out, the `OPENWINHOME` environment variable in the `.profile` or `.cshrc` file:

1. **Open the** `.profile` **or** `.cshrc` **file using a text editor such as** `vi`**.**
2. **Type a pound sign (#) before the variable, as shown below, or delete the line entirely.**
   If you are working in the `.profile` file, use example `a`; if you are working in the `.cshrc` file, use example `b`

a. **In the `.profile` file:**

```
#OPENWINHOME=/usr/openwin
```

a. **In the `.cshrc` file:**

```
#setenv OPENWINHOME /usr/openwin
```

3. **Save and quit the file.**

The `.xinitrc` File

The following are important notes about the `.xinitrc` and `$OPENWINHOME/lib/Xinitrc` files:

1. In the OpenWindows Version 2 environment, the `openwin` script automatically created a copy of `$OPENWINHOME/lib/Xinitrc` to a file called `.xinitrc` in your home directory. This is no longer the case in the OpenWindows Version 3.2 environment. This is significant because:
   a. The `openwin` start-up script uses the default start-up file, `$OPENWINHOME/lib/Xinitrc`, unless there is an `.xinitrc` file in your home directory, which overrides the default.
   b. It is important that you use the default `$OPENWINHOME/lib/Xinitrc` file that came with the OpenWindows Version 3.2 software. (However, if you want to retain any special changes you made to the `.xinitrc` file in the Version 2 software, you can do so by following the instructions in this section.)
2. If you run your system with multiple screens, you no longer need multiple instances of `olwm`.

Using the Correct Start-up File

If you are currently running a version of the OpenWindows software earlier than Version 3.2, it is important to determine the status of your `.xinitrc` file. The `.xinitrc` file is an OpenWindows start-up file your home directory that may contain user-defined options.

To determine the status of your `.xinitrc` file, type the following commands:

```
$ cd
$ ls -a .xinitrc
```

Depending on the output of this command, do one of the following things:

- If you do not have a `.xinitrc` file (that is, the results of the previous `ls -a` command does not return a listing for the file) do nothing. If there is no `.xinitrc` file in your home directory, OpenWindows uses the system default start-up file.
- If you have a `.xinitrc` file (that is, the result of the previous `ls -a` command returns a listing for the file), but you have never made any changes to the file or do not want to keep the changes you have made, do Step 1 "Start-Up File Procedures."
- If you have a `.xinitrc` file (that is, the result of the previous `ls -a` command returns a listing for the file), and you have made changes to the file that you want to keep, do Step 2 in "Start-Up File Procedures."

Start-Up File Procedures

1. **To delete the** `.xinitrc` **file from your home directory, type the following command:**

```
$ rm .xinitrc
```

2. **To retain the changes to your** `.xinitrc` **file, do the following steps:**

   a. **Move** `.xinitrc` **to** `.xinitrc.save`:

```
$ mv .xinitrc .xinitrc.save
```

   b. **Copy** `$OPENWINHOME/lib/Xinitrc` **to** `.xinitrc` **in your home directory:**

```
$ cp $OPENWINHOME/lib/Xinitrc $HOME/.xinitrc
```

c. **Add the lines that you want to keep from the** `.xinitrc.save` **to** `.xinitrc`.

Caution – When editing the `.xinitrc` file, do not add a secondary version of `olwm`, do not add `svenv`, and do not remove the line containing `$OPENWINHOME/lib/openwin-sys`.

Workspace Properties

In prior versions of OpenWindows (before version 3.2), a change made in the Workspace Properties menu would be stored in the file `.Xdefaults` in your home directory. In OpenWindows version 3.2, changes made in the Workspace Properties menu are now stored in the file `.OWdefaults` , also in your home directory. The `.Xdefaults` file can still exist, but precedence is given to the customizations made in `.OWdefaults`.

The `.Xdefaults` file should be used *only* to make additional customization changes that cannot be made through Workspace Properties. For example, you can edit the `.Xdefaults` file using a text editor such as `vi` to make customizations to non-OpenWindows applications or to add C pre-processor macros. Using Workspace Properties does not disturb these customizations.

If you already have a `.Xdefaults` file in your home directory and you do not want to make any customization changes to it, you do not need to remove it. Since the `.OWdefaults` file takes precedence over it, it does not interfere.

Customizing the Workspace Menu

In OpenWindows 3.2, you customize the Programs submenu on the workspace menu using Workspace Properties. Prior to OpenWindows 3.2, you did this customization by editing the `.openwin-menu` file in your home directory.

Note – If you do not have a `.openwin-menu` file in your home directory, it is not necessary to do the following procedure. You can customize the workspace menu by using Workspace Properties.

If you do have a `.openwin-menu` file, you must perform the following steps in order to use Workspace Properties to customize your workspace menu.

If you see the following line in your .openwin-menu file:

```
"Programs"MENU $OPENWINHOME/lib/openwin-menu-programs
```

delete it and replace it with this line:

```
"Programs"INCLUDE openwin-menu-programs
```

If your .openwin-menu file does not contain the line that needs to be removed and replaced, simply add the substitute line to the .openwin-menu file, as shown above.

Adding or substituting this line adds the default Programs menu to your workspace menu. This enables you to customize it using Workspace Properties.

If, by chance, you end up with redundant items in your workspace menu, simply edit them out by removing the redundant lines from .openwin-menu.

Modifying the Keyboard and Mouse B

This appendix provides instructions for remapping your keyboard and mouse. It includes remapping options for special keyboard keys, and for more convenient left-handed use of the mouse. It also provides information on how to disable and enable the Compose key on your keyboard.

Disabling/Enabling the Compose Key

If you do not use the Compose key, you can disable it so that you do not have to worry about pressing 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)

The important piece of information is the two-digit keycode number at the beginning of the line, represented by *nn*. Use this keycode number 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.

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 and right panels of the keyboard (that is, the keypads to the left and right of the main keyboard area).

Using the Remapping Script

Follow these steps to create and use your remapping script:

1. **Create a file called `lefty.data` using any text editor.**
   This 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 Table B-1 "The `lefty.data` Script.".**
   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`
   You must be in the same directory as the script file.
5. **Click a mouse button in the Workspace to make the script take effect.**
   Once you have completed these steps you can use the keyboard so the keys are mapped for a left-handed person.

Type the following script into the file `lefty.data`, as described in step 1.

Table B-1 The `lefty.data` Script

```
!
! 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).
!
```

Table B-1 The `lefty.data` Script

```
! 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 (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      SunXK_Again
keycode 32 = R6          R6      KP_Multiply
keycode 54 = L3          L3      SunXK_Props
keycode 33 = R4          R4      KP_Equal
keycode 52 = L4          L4      SunXK_Undo
keycode 56 = R9          R9      KP_9      Prior
keycode 77 = L5          L5      SunXK_Front
```

Table B-1 The `lefty.data` Script

```
keycode 58 = R7         R7     KP_7        Home
keycode 75 = L6         L6     SunXK_Copy
keycode 79 = Right      R12    KP_6
keycode 100 = L7        L7     SunXK_Open
keycode 80 = Left       R10    KP_4
keycode 98 = L8         L8     SunXK_Paste
keycode 102 = R15       R15    KP_3        Next
keycode 121 = L9        L9     SunXK_Find
keycode 104 = R13       R13    KP_1        End
keycode 119 = L10       L10    SunXK_Cut
keycode 30 = L1         L1     SunXK_Stop
keycode 29 = Left       R10    KP_4
keycode 53 = Right      R12    KP_6

keycode 118 = Control_R
add control = Control_R
```

Undoing the Keyboard Remapping

There are two ways to switch the keys back to their original settings. The first is to exit the OpenWindows software and start it up again. The second method, which is much preferable if you may want to switch the keys back periodically, is to create a second script and initiate it any time you want to switch back.

Follow these instructions to create the second script:

1. **Use any editor to create a file called `nolefty.data`.**
   This must be in the same directory that contains the `lefty.data` script.

2. **Type in the script as shown in Table B-2 "The `nolefty.data` Script.".**
   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 nolefty.data
```

For the `nolefty.data` file to take affect, you must enter the previous command in the same directory as the script file.

Table B-2 The `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 t ype-4 keyboard.  To use this data type the following
! where <filename> is the name of this file.
!
! xmodmap <filename>
!
!Reassign standard values to left function keys
!
keycode 10 = L2     L2    SunXK_Again
keycode 32 = L3     L3    SunXK_Props
keycode 33 = L4     L4    SunXK_Undo
keycode 56 = L5     L5    SunXK_Front
keycode 58 = L6     L6    SunXK_Copy
keycode 79 = L7     L7    SunXK_Open
keycode 80 = L8     L8    SunXK_Paste
keycode 102 = L9    L9    SunXK_Find
keycode 104 = L10   L10   Sun XK_Cut
!
! Reassign standard values to right function keys.
!
```

Table B-2 The `nolefty.data` Script

```
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
5keycode 118 = Linefeed
```

Running Networked Applications C

This appendix describes an advanced feature of the OpenWindows environment that enables you to run applications that reside on another machine on your network.

Note – Most users do not need to read this appendix. If you want to explore the possibility of running networked applications, you can talk to your system administrator about the special applications that may be available on your network.

Normally in the OpenWindows environment all the applications on your screen (such as Mail Tool and Calendar Manager) 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 in this manner saves computing cycles on your local machine, and gives you access to an entire network of applications.

This appendix describes the simplest scenario for running an application on a remote machine and displaying it on your local screen. Because your computing environment may vary, you will need to be flexible in following these instructions. The section "More About Security," provides additional information on the complexities of running networked applications.

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, or *host*.

Contact your system administrator if you do not understand these requirements.

Using `rlogin` *to Run a Networked Application*

The key to running a networked application on a remote machine is to make sure your environment variables are set correctly:

- The `HOME` environment variable in your shell on the remote machine must be set to your home directory.
- The `DISPLAY` environment variable in your shell on the remote machine must be set to your local screen.
- If the OpenWindows libraries have not been installed in the standard `/usr/lib` or `/usr/local` shared library directories, you must set the `LD_LIBRARY_PATH` environment variable to the appropriate directory (`$OPENWINHOME/lib`).

Below is an example of running a Command Tool on a remote machine using `rlogin`. In this example, the home directory is mounted on the remote machine on `/home/mydirectory`, and the OpenWindows software is located in /usr/openwin on the remote machine. Change the variables, *mydirectory* and *mymachine* as appropriate for your arrangement. Also, replace cmdtool, with the name of the application you want to run.

```
$ rlogin remotemachine
    .
    .
(The following commands are executed on the remote machine.) ]
    .
    .
$ HOME=/home/mydirectory
$ DISPLAY=mymachine:0
$ LD_LIBRARY_PATH=/usr/openwin/lib
$ /usr/openwin/bin/cmdtool &
```

After you enter the last line, a Command Tool window appears on your screen. Even though you interact with this application just as you would with any other application on your screen, the Command Tool application itself is actually running on the remote machine.

Although there is no particular advantage to running a Command Tool in this way (it is already locally available on your machine and does not use a significant amount of your computing resources), this example shows how to run any remote application that may be available to you.

More About Security

This section describes some fundamentals of network security that you may 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

The default security configuration in the OpenWindows Version 3.2 software does not need to be changed 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 is a means of deciding which *clients*, or applications, have access to the X11/NeWS server. Only properly authorized clients are allowed to connect to the server; all others are denied access, and are terminated with an error message.

There are two different types of access control mechanisms: *user-based* and *host-based*. (That is, one mechanism grants access to a particular user's account, while the other grants access to a particular *host*, or machine.) Unless the `-noauth` option is used with `openwin`, 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" in this chapter.

User-Based Access

A user-based, or authorization-based mechanism allows you to explicitly give access 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. It allows you to give access to a particular host, in which all users on that host can connect to the server. This is a weaker form of access control: if that host has access to the server, all users on that host are allowed to connect to the server.

The host-based mechanism is primarily used for backward compatibility. Applications linked with versions of Xlib or libcps older than the OpenWindows Version 2 software or X11R4 do not recognize the new user-based access control mechanism. To give these applications the ability to connect to the server, a user must either switch to the host-based mechanism, or relink with the newer versions of Xlib and libcps.

Note – If possible, clients linked with older versions of Xlib or libcps should be relinked with newer versions of these libraries 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 OpenWindows software: MIT-MAGIC-COOKIE-1 and SUN-DES-1. They differ in the authorization data used; they are similar in the access control mechanism used. At any time, the server implements only one protocol. The MIT-MAGIC-COOKIE-1 protocol using the user-based mechanism is the default in the OpenWindows software.

MIT-MAGIC-COOKIE-1

The MIT-MAGIC-COOKIE-1 authorization protocol was developed by the Massachusetts Institute of Technology. At server start-up, 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 RPC (Remote Procedure Call) and requires DES (Data Encryption Software) 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. There is no way for another user to use your machine independent netname to access a server, but it is possible for another user to use the magic cookie to access a server.

"Allowing Access When Using SUN-DES-1," in this chapter, describes how to allow another user access to your server by adding their netname to your server's access list.

Changing the Default Authorization Protocol

The default authorization protocol, MIT-MAGIC-COOKIE-1, can be changed to SUN_DES-1, the other supported authorization protocol, or to no user-based access mechanism at all. You change the default by supplying options with the openwin command. For example, to change the default from MIT-MAGIC-COOKIE-1 to SUN-DES-1, start the OpenWindows software as follows:

```
$ openwin -auth sun-des
```

If you must run the OpenWindows software without the user-based access mechanism, use the -noauth command line option:

```
$ openwin -noauth
```

Caution – Using -noauth weakens security. It is equivalent to running the OpenWindows software with the host-based access control mechanism only; the server inactivates the user-based access control mechanism. Anyone that can run applications on your local machine will be allowed access to your server.

Manipulating Access to the Server

Unless the -noauth option is used with openwin (see "Changing the Default Authorization Protocol"), 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. Since 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.

There are three programs you can use to change a server's access control mechanism: `xhost`, `xauth`, and `newshost`. These programs access two binary files 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` and `newshost` programs 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` or `newshost` to add a machine name, you will lower the level of security. The server will allow access to the host you added, as well as to any user specifying the default authorization protocol. See "Host-Based Access" for 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 information from your `.Xauthority` file so that another user can merge the data into their `.Xauthority` file, thus allowing them access to your server, or to the server to which you connect.

See "Allowing Access When Using MIT-MAGIC-COOKIE-1" 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 may 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 up, 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` and `newshost` return empty host-based access lists and state 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 may contain the following entries, where unix.*15339@EBB.Eng.Sun.COM* is the machine-independent netname of the user:

```
anyhost:0        SUN-DES-1           "unix.15339@EBB.Eng.Sun.COM"
localhost:0      SUN-DES-1           "unix.15339@EBB.Eng.Sun.COM"
anyhost/unix:0   SUN-DES-1           "unix.15339@EBB.Eng.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 running the server, use** `xauth` **to extract an entry corresponding to** *hostname:0* **into a file.**
   For this example, *hostname* is *anyhost* and the file is *xauth.info*:

```
$ $OPENWINHOME/bin/xauth nextract - anyhost:0 > $HOME/xauth.info
```

Note that because of the length of the line above, it wraps to a second line; however, it should appear on a single line when you type it on your screen.

2. **Send the file containing the entry to the user requesting access (using Mail Tool,** `rcp` **or some other file transfer method).**

Note – Mailing the file containing 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 their** `.Xauthority` **file.**
   For this example, *userhost* merges `xauth.info` their `.Xauthority` file:

```
$ $OPENWINHOME/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 running 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*:

```
$ xhost + somebody@
```

2. **The new user must use** xauth **to add the entry into their** .Xauthority **file.**
   For this example, the new user's machine independent netname is unix.*15339@EBB.Eng.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.

```
$ echo 'add myhost:0 SUN-DES-1 "unix.15339@EBB.Eng.Sun.COM"' |
$OPENWINHOME/bin/xauth
```

Note that because of the length of the line above, it wraps to a second line; however, it should appear on a single line when you type it on your screen

Running Clients Remotely, or Locally as Another User

X and NeWS clients use the value of the DISPLAY environment variable to get the name of the server to which they should connect. For backward compatibility, NeWS clients can also use the NEWSSERVER environment variable.

Note – For NeWS clients, care should be taken when connecting to remote servers: NEWSSERVER is checked before DISPLAY, so if they point to different machines or remote servers, the NeWS client will connect to the machine referred to by NEWSSERVER. Either ensure that both environment variables are set to the same remote server, or unset NEWSSERVER before setting DISPLAY.

To run clients remotely, or locally as another user, follow these steps:

1. **On the machine running 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" or "Allowing Access When Using SUN-DES-1."

2. **Set** `DISPLAY` **to the name of the host running the server.**
   For this example, the host is *remotehost*:

```
$ DISPLAY=remotehost:0
```

3. **Run the client program as shown.**

```
$ client_program&
```

The client is displayed on the remote machine, *remotehost*.

DECnet Internetworking(DNI) D

This appendix describes how to internetwork the OpenWindows environment and the DECwindows™ environment via the NSP DECnet transport protocol. There are two DNI scenarios:

- Running an X11 client on a VAX system (under the VMS® operating system) and displaying the client's window on an OpenWindows machine
- Running an X11 client on an OpenWindows machine and displaying the client's window on a VAX system

These two scenarios are described in the following sections after an initial section on how to set up the DNI software for either scenario.

Note – DECnet internetworking is available only with 7.x DNI.

Setting Up DECnet Internetworking

To set up DECnet internetworking, follow these steps:

1. **Enable a connection via DNI.**
   The OpenWindows server and client libraries use a dynamically loadable version of the DNI transport library `libdni`. In order for the server and client libraries to load `libdni` you must set the environment variable `DNI_X_ENABLE` to the directory where `libdni.so` is installed.

The example below assumes you loaded DNI via pkgadd in the default location:

```
$ DNI_X_ENABLE=/opt/SUNWconn/dni/lib
```

2. **Start the OpenWindows server.**
   By default, the OpenWindows server supports "MIT-MAGIC-COOKIE" security. This security mechanism is user-specific, rather than host-specific—you decide which users may connect to the server instead of which machines may connect to the server. In the default mode, the xhost and newshost commands both return empty lists, and state only that security is turned on. You can turn off this security mode (and revert to the security mode of previous OpenWindows server versions) by using the -noauth option with the openwin command.

```
$ openwin -noauth
```

3. **Request the owner of the machine running the OpenWindows software to use the xhost command to give DEC® VAX® permission to have an X11 connection to the OpenWindows server.**
   In order for X11 clients to connect to the OpenWindows server through the DNI software, the DECnet node addresses must be mapped to their DECnet node names. You do this by creating and initializing the NCP database. This must also be done on the DEC VAX system.

```
$ xhost decvax::
```

The double colon specifies the DECNet transport.

Displaying a Remote Client on an OpenWindows Machine

You can run X11 clients from VMS by using the SunLink DNI dnilogin command to log into the VAX system. First, set the environment variable DISPLAY on your local machine to be the X11 server on the remote machine. Then run an X11 client by entering the name of the client, represented here by *x11_client*. (See the *VMS DECwindows User's Guide, Running Applications Across the Network* for more information on using the VMS operating system.)

For example:

```
$ dnilogin decvax
    .
    .
    .
$ define DECW$DISPLAY OW_machine::0
$ spawn/nowait run x11_client
```

Displaying a Remote Client on a VAX

You can run X11 clients on an OpenWindows machine and display them on a DECwindows server by setting the `DISPLAY` variable to the remote VAX system.

Before you can run any of the X11 clients you must compile and install the OpenWindows fonts in the DECwindows server. These fonts are available in the MIT X11R4 release or in the optional font package with the OpenWindows Version 3.2 software. Follow these steps to install the proper fonts in the DECwindows server:

1. **Install the optional OpenWindows font sources (of the font sources from the MIT X11R4 release) on the OpenWindows machine.**
2. **See the** *OpenWindows Version 3.2 Programmer's Guide* **for font installation instructions.**
3. **Copy the font sources to a directory on the VAX system.**

```
$ cd $OPENWINHOME/share/src/fonts/misc
$ dnicp *.bdf 'decvax::[vaxdir]'
```

4. **Compile the cursor fonts on the VAX system.**
   This results in files such as: `olcursor.decw$font;1` `olglyph10.decw$font;1`...

```
$ font olcursor.bdf
$ font olglyph10.bdf
...
```

5. **Copy the fonts to the** `sysfont` **directory:**

```
$ set def sys$sysroot:[sysfont.decw.user_cursor16]
$ copy [vaxdir]olcursor.decw$font;1 *
```

Note – You must be logged in as "system" on the DEC VAX system to copy the fonts to the `sysfont` directory.

---

6. **You must also perform steps 2-4 for the rest of the cursor fonts and for the Lucida fonts in** `$OPENWINHOME/share/src/fonts/75dpi` **and** `$OPENWINHOME/share/src/fonts/100dpi`**.**

Note – The Lucida fonts should be installed in `sys$sysroot:[sysfont.decw.user_75dpi]` and `sys$sysroot:[sysfont.decw.user_100dpi]`.

The following list shows the minimum working set of fonts needed to be installed in order to run the OpenWindows DeskSet tools. If you are using default fonts for the applications you should only have to install these fonts. You can, however, install more fonts as needed.

- `olcursor.bdf`
- `olglyph10.bdf`
- `olglyph12.bdf`
- `olglyph14.bdf`
- `olglyph19.bdf`
- `luBS08.bdf`
- `luBS10.bdf`
- `luBS12.bdf`
- `luBS14.bdf`
- `luRS08.bdf`
- `luRS10.bdf`
- `luFS12.bdf`
- `lutBs12.bdf`
- `lutRS10.bdf`
- `lutRS12.bdf`

7. **Restart the DECwindows server.**

8. **You can verify that the fonts were installed by listing the available fonts in the DECwindows server:**

```
$ DISPLAY=decvax::0
$ xlsfonts | grep Sun     (Cursor fonts)
$ xlsfonts | grep Lucida
```

9. **Make sure you have given the OpenWindows node permission to display on the DECwindows server by using the Security Menu in the DECwindows Session Manager.**

10. **Run an X11 application (for example, an OpenWindows DeskSet tool).**

```
$ DISPLAY=decvax::0
$ mailtool
```

Note – `DNI_X_ENABLE` must be set to the location of the DNI transport library `libdni`. See Step 1 under "Setting Up DECnet Internetworking," in this chapter.

If an error message such as the one below is printed, you need to install that font in the DECwindows server in order to run the application.

```
XView warning: Cannot load font '-b&h-lucida-medium-r-*-*-*-
80-*-*-*-*-*-*' (Font package)
```

This error message means font `luRS10.bdf` needs to be installed.

See the *OpenWindows Version 3.2 Programmer's Guide* for more information on fonts.

Index

801-3660-10
Revision A of May 1993

SunSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue
Mountain View, CA 94043 U.S.A.