Mac OS X Server

Introduction to Command-Line Administration

Version 10.6 Snow Leopard

Apple
1 Infinite Loop
Cupertino, CA 95014
408-996-1010
www.apple.com

019-1398/2009-08-01

Contents

About This Guide

This guide provides a starting point for administering Mac OS X Server using command-line tools.

Introduction to Command-Line Administration supplements the information in the other advanced administration guides. It provides information useful to building workflows and remote administration practices beyond the use of Server Admin and Workgroup Manager. The information in this guide isn't specific to any particular technology, but is relevant to many server technologies.

What's in This Guide

This guide includes the following sections:

- Chapter 1, "Introduction to the Command-Line Environment," provides an overview of the command-line environment in Mac OS X Server—for administrators who are new to the command line or who are coming from the command line on other platforms.
- Chapter 2, "Using the Command-Line Shell Interactively," explains how shells work and provides information about the shells in Mac OS X Server.
- Chapter 3, "Scripting the Command Line," explains what shell scripts are and why you would use them in Mac OS X Server.
- Chapter 4, "Connecting to Remote Computers," provides information about various ways to access remote computers.
- Chapter 5, "Common Command-Line Tasks," provides examples of frequently used command-line tasks.
- P7-8Chapter 6, "Accessing Apple Hardware from the Command Line," provides information about accessing hardware-specific Mac attributes from the command line.
- Appendix , "Command-Line Tools Specific to Mac OS X," provides a list of the command-line tools that are unique to Mac OS X and Mac OS X Server.

Note: Because Apple periodically releases new versions and updates to its software, images shown in this book may be different from what you see on your screen.

Using Onscreen Help

You can get task instructions onscreen in Help Viewer while you're managing Snow Leopard Server. You can view help on a server, or on an administrator computer. (An administrator computer is a Mac OS X computer with Snow Leopard Server administrator software installed on it.)

To get the most recent onscreen help for Mac OS X Snow Leopard Server:

- Open Server Admin or Workgroup Manager and then:
  - Use the Help menu to search for a task you want to perform.
  - Choose Help > Server Admin Help or Help > Workgroup Manager Help to browse and search the help topics.

  The onscreen help contains instructions taken from *Advanced Server Administration* and the other administration guides.

To see the most recent server help topics:

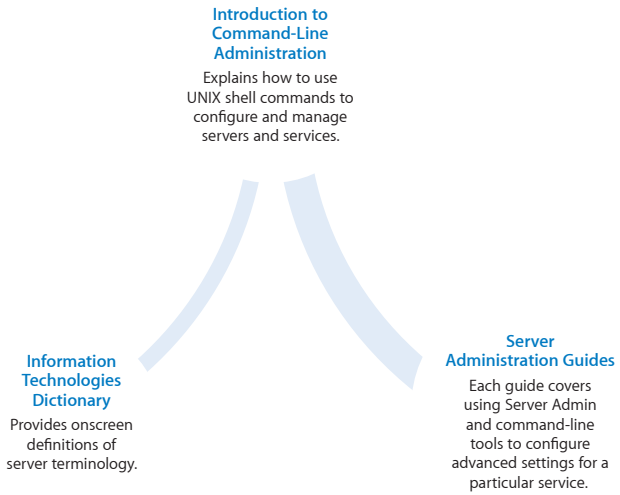
- Make sure the server or administrator computer is connected to the Internet while you're getting help.

  Help Viewer automatically retrieves and caches the most recent server help topics from the Internet. When not connected to the Internet, Help Viewer displays cached help topics.

Documentation Map

Snow Leopard has a suite of guides that cover management of individual services. Each service may be dependent on other services for maximum utility. The documentation map below shows some related documentation that you may need in order to fully configure your desired service to your specifications. You can get these guides in PDF format from the Mac OS X Server Resources website:

http://www.apple.com/server/macosx/resources/



Viewing PDF Guides Onscreen

While reading the PDF version of a guide onscreen:

- Show bookmarks to see the guide's outline, and click a bookmark to jump to the corresponding section.
- Search for a word or phrase to see a list of places where it appears in the document. Click a listed place to see the page where it occurs.
- Click a cross-reference to jump to the referenced section. Click a web link to visit the website in your browser.

Printing PDF Guides

If you want to print a guide, you can take these steps to save paper and ink:

- Save ink or toner by not printing the cover page.
- Save color ink on a color printer by looking in the panes of the Print dialog for an option to print in grays or black and white.
- Reduce the bulk of the printed document and save paper by printing more than one page per sheet of paper. In the Print dialog, change Scale to 115% (155% for *Getting Started*). Then choose Layout from the untitled pop-up menu. If your printer supports two-sided (duplex) printing, select one of the Two-Sided options. Otherwise, choose 2 from the Pages per Sheet pop-up menu, and optionally choose Single Hairline from the Border menu. (If you're using Mac OS X v10.4 or earlier, the Scale setting is in the Page Setup dialog and the Layout settings are in the Print dialog.)

You may want to enlarge the printed pages even if you don't print double sided, because the PDF page size is smaller than standard printer paper. In the Print dialog or Page Setup dialog, try changing Scale to 115% (155% for *Getting Started,* which has CD-size pages).

Getting Documentation Updates

Periodically, Apple posts revised help pages and new editions of guides. Some revised help pages update the latest editions of the guides.

- To view new onscreen help topics for a server application, make sure your server or administrator computer is connected to the Internet and click "Latest help topics" or "Staying current" in the main help page for the application.
- To download the latest guides in PDF format, go to the Mac OS X Server Resources website at:

  www.apple.com/server/macosx/resources/
- An RSS feed listing the latest updates to Mac OS X Server documentation and onscreen help is available. To view the feed use an RSS reader application, such as Safari or Mail:

  feed://helposx.apple.com/rss/snowleopard/serverdocupdates.xml

Getting Additional Information

For more information, consult these resources:

- *Read Me documents*—get important updates and special information. Look for them on the server discs.
- *Mac OS X Server website* (www.apple.com/server/macosx/)—enter the gateway to extensive product and technology information.
- *Mac OS X Server Support website* (www.apple.com/support/macosxserver/)—access hundreds of articles from Apple's support organization.
- *Apple Discussions website* (discussions.apple.com/)—share questions, knowledge, and advice with other administrators.
- *Apple Mailing Lists website* (www.lists.apple.com/)—subscribe to mailing lists so you can communicate with other administrators using email.
- *Apple Training and Certification website* (www.apple.com/training/)—hone your server administration skills with instructor-led or self-paced training, and differentiate yourself with certification.

Introduction to the Command-Line Environment

1

Use this chapter to determine when to use command-line tools and to understand the fundamentals of how to use them.

A command-line interface (CLI) is an alternative to graphical applications for interacting with and controlling your computer. Mac OS X Server provides graphical applications—primarily, Server Admin and Workgroup Manager—to address common administration tasks. There are situations, though, where using a command-line interface might be appropriate. These situations include:

- Configuring advanced options that aren't supported by the graphical applications.
- Configuring remotely from a computer that doesn't have the Server Admin tools installed—for example, a computer with Windows, Linux, or another UNIX-based operating system.
- Performing tasks that are repetitive or that need to be run at predefined times.
- Editing text files, usually in order to change advanced configuration settings and preferences.

The primary way to access the CLI in Mac OS X is with the Terminal application. Other ways to access the CLI are discussed in "Accessing the Shell" on page 11, and in Chapter 4, "Connecting to Remote Computers."

Each window in Terminal contains an execution context, called a *shell,* which is separate from all other execution contexts. The shell is an interactive programming language interpreter, with a specialized syntax for executing commands and writing structured programs (shell scripts). Different shells have slightly different capabilities and programming syntax. Although you can use any shell, the examples in this book use `bash`, the startup shell for Mac OS X and the default user shell.

The Command-Line Environment

This section gives some background information about UNIX and shells. Both are important for understanding the command-line environment in Mac OSX Server.

UNIX

Mac OS X and Mac OS X Server are built on the foundation of the UNIX operating system. UNIX-based operating systems include BSD, GNU/Linux, AIX, and Solaris. The shared heritage of these operating systems means that many programs are compatible across this larger family with minimal changes.

The unique underpinnings of each brand of UNIX are what distinguish them from each other. To support programs and utilities that work across multiple flavors of UNIX, there are some standard specifications set by various regulatory bodies. One such specification is The Open Group's "Single UNIX Specification." Mac OS X versions 10.5 and later conform to version 3 of this specification, which implies conformance to the SUSv3 and POSIX 1003.1 specifications for the C API, shell utilities, and threads. Code that complies with the UNIX-03 specification works not only on Mac OS X Server, but on any other compliant system.

For more information about the The Single UNIX Specification, Version 3, see http://www.unix.org/version3/.

The Shell

In UNIX-based operating systems, the shell is the fundamental user interface. The shell is an environment that presents a simple textual prompt to the user and accepts keyboard input from the user.

In Mac OS X, the shell is easily accessed through Terminal, but there are other options as well. The shell can be invoked interactively, or by a text file with commands to the shell given in a standard format. There are several shells available in Mac OS X, each with its own strengths and capabilities. Shells included in Mac OS X include `bash`, `csh`, `ksh`, `sh`, `tcsh`, and `zsh`.

For information about these shells, see their man pages.

Accessing the Shell

To enter shell commands or run server command-line tools, you need access to the UNIX shell prompt on the local server or on a remote server.

Local Access

There are multiple ways to access the shell on your local computer. Under normal circumstances you can use Terminal, but for advanced troubleshooting or configuration, you may want to use a different way to access the command line.

Logging In from Terminal

To open Terminal, click the Terminal icon in the dock or double-click the application icon in the Finder (in /Applications/Utilities/). Each window in Terminal represents another instance of a shell process.

Terminal presents a prompt when it's ready to accept a command. The prompt you see depends on your Terminal and shell preferences, but it often includes the name of the host you're logged in to, your current working folder, your user name, and a prompt symbol.

For example, if you're using the default `bash` shell, the prompt appears as:

```
server1:~ mariah$
```

This indicates that you're logged in to a computer named *server1* as the user named *mariah,* and your current folder is Mariah's home folder (~).

Logging In from the Console

You can log in to a command-line version of Mac OS X without running the window manager. This mode is more advanced than single-user mode because the entire system is running.

To log in without the window manager:

1. In the Accounts pane of System Preferences, select Login Options.
2. Make sure the settings for "Display login window as:" is set to "Name and password."
3. Log out any logged in users.
4. In the login window, type ">console" and press Return. Don't enter a password.

   You'll be prompted to log in with the user name and password of a user on the system.

   Logging in to the console at this level can help you troubleshoot issues that are graphics-related or that are triggered by users logging in to the system through the GUI.

Single-User Mode

To debug a computer problem, you can restart the computer and hold down Command-S as the computer boots. The computer boots up verbosely from the command line to a certain point, and won't continue booting without your intervention. The window server won't be running, and many services won't be started. Onscreen instructions guide you through mounting and verifying the attached volumes. This is a useful way to boot if you want to troubleshoot hardware-related issues or determine what's happening in software before higher-level processes and applications are running. At this point, very few processes are running.

The following processes and services aren't running if you boot into single-user mode:

- Directory Services
- Kerberos
- `syslogd`
- mDNSResponder
- `securityd` (and many related security processes)
- Spotlight
- Any other server services (such as Mail Server, Web Server, or Wiki Server) you may have configured

X11

X11 is a window manager traditionally used in UNIX-based operating systems. Although Mac OS X Server is a UNIX operating system, it doesn't use X Windows as its window manager. X11 is available to provide compatibility with other UNIX-based operating systems. All normal Mac OS X Server tasks are performed with tools that don't rely on X11. To connect to the X11 server remotely, see "Configuring and Running X11 Applications on Mac OS X" on the Apple Developer Connection website.

Serial Console

Xserve hardware includes a 9-pin serial port. To access the Xserve, you can connect a terminal or use terminal emulation software on a computer connected by a serial-to-USB cable. No other Apple hardware includes a serial port.

Remote Access

Various ways of accessing the command-line interface on remote computers are using are discussed in Chapter 4, "Connecting to Remote Computers."

Closing the Shell

To quit a shell session, enter the command `exit`. This ensures that any commands the shell is actively running are closed. If anything's still in progress, the shell warns you.

Executing Commands and Running Tools

To execute a command in the shell, enter the complete pathname of the tool's executable file, followed by arguments, and then press Return.

If a command is located in one of the shell's known folders, you can omit path information and enter just the command name.

The list of known folders is stored in the shell's PATH environment variable and includes the folders containing most command-line tools.

For example, to run the `ls` command in the current user's home folder, you could enter the following at the command line and press Return:

```
host:~ mariah$ ls
```

The shell looks through the list of folders in the PATH variable until it finds a program named `ls`; in this case, it finds `ls` in `/bin`, and runs `/bin/ls`.

To run a command in the current user's home folder, precede it with the folder specifier. For example, to run MyCommandLineProg, use the following:

```
host:~ mariah$ ~/MyCommandLineProg
```

To open an application, use the `open` command:

```
open -a MyProg.app
```

When entering commands, if you get the message `command not found`, check your spelling. Here's an example:

```
server:/ mariah$ opne -a TextEdit.app
-bash: opne: command not found
```

If this error recurs, the command you're trying to run might not be in your default search path. You can add the path before the command name:

```
server:/ mariah$ sudo /System/Library/ServerSetup/serversetup
     -getHostname
server.example.com
```

or change your working folder to the folder that contains the tool:

```
server:/ mariah$ cd /System/Library/ServerSetup
server:/System/Library/ServerSetup mariah$ sudo ./serversetup
     -getHostname
server.example.com
```

or define the path for this session and then run the command:

```
server:/ mariah$ PATH="$PATH:/System/Library/ServerSetup"
server:/ mariah$ sudo serversetup -getHostname
server.example.com
```

Terminating Commands

To terminate the currently running command, press Control-C. This keyboard shortcut sends an abort signal to the command. In most cases this causes the command to terminate, although commands can install signal handlers to trap this signal and respond differently.

Specifying Files and Folders

Most commands operate on files and folders, whose locations are identified by paths. The folder names that make up a path are separated by slashes. For example, the path to the Terminal application is /Applications/Utilities/Terminal.app.

Standard shortcuts used to represent specific folders are shown in the following table. They are specified relative to the current folder, and can eliminate the need to enter full paths.

Shortcut	Description
.	A single period represents the current folder. For example, the string "./Test.c" represents the Test.c file in the current folder.
..	Two periods represent the parent folder of the current folder. For example, the string "../Test" represents a sibling folder (named Test) of the current folder.
~[username]	The tilde character represents the home folder of the logged-in user. For example, to specify the Documents folder, of the current user, you would specify ~/Documents. To specify another user's Document folder you would use their short name preceded by the tilde (~) character—for example, ~jsmith/Documents. In Mac OS X, this folder is in the local /Users folder or on a network server. For a list of all the short names on your system, type `dscl . -list /Users`. Most of these users aren't traditional user accounts with home directories, but you should be able to find the short name of known users on the computer.

File and folder names can include letters, numbers, a period, or the underscore character. Avoid most other characters, including space characters. Although some Mac OS X file systems permit the use of these other characters, including spaces, you might need to add single or double quotation marks around pathnames that contain them.

For individual characters, you can also "escape" the character—that is, put a backslash character immediately before the character in your string. For example, the pathname My Disk is "My Disk" or My\ Disk.

Commands Requiring Root or Administrator Privileges

Many commands used to manage a server must be executed by an administrator user or the root user. For example, entering:

```
server:~ mariah$ shutdown
```

gives you the following error:

```
shutdown: NOT super-user
```

This is because the `shutdown` command can be run only by the root user or by an administrative user with special privileges. To run commands in this "super user" mode, use the `sudo` command. `sudo` stands for "super user do." The following command does work, (so don't run it unless you really want to restart your computer):

```
server:~ mariah$ sudo shutdown
```

You'll be prompted for the password of the currently logged in user. Only users that you have designated as admin users are able to execute commands with `sudo`. If you're logged in as a user who isn't an admin user, you can change "substitute users" by typing `su` *`adminUsername`*, where *`adminUsername`* is the name of a user in the Admin group. After you enter that user's password, a new shell is launched from the existing shell, as that user. If a command requires it, you can use `su` to log in as the root user. Under normal circumstances you don't need to use the root user account. If you do `su` to the root user, be especially careful, as you have sufficient privileges to make changes that can cause your server to stop working.

For more information about the `sudo` and `su` commands, see their man pages.

Getting Help for Command-Line Tools

Command-line tools provide multiple mechanism for getting help while using them. This section describes three ways that you can get help from the command-line.

Using Help Built Into Command-Line Tools

Most command-line tools include a parameter to invoke onscreen help directly. Command-line tools do not always follow the same conventions so if one parameter doesn't work try another.

To access command help:

Enter the command followed by the `-help`,`-h`,`--help`, or `help` parameter:

```
$ hdiutil help
$ dig -h
$ diff --help
```

To view a list of options and parameters you can use with the command:
Enter the command without options or parameters:

```
$ sudo serveradmin
```

Some commands don't have onscreen help.

Using Man Pages

Most command-line documentation comes in man pages. Man pages provide reference information for shell commands, tools, and high-level concepts.

To access a man page entry:

```
$ man command
```

Replace *command* with the name of the command you want to find information about. The man page contains detailed information about the command, its options and parameters, and proper use.

For help using the `man` command itself, enter:

```
$ man man
```

You can press the Space bar to go to the next page, the B key to go back a page, or the Return key to scroll forward one line at a time. Press the Q key to exit the man page.

You can search within the contents of a man page by pressing the / key followed by the word you're looking for. If multiple instances are found, the P and N keys let you access the previous and next instances of the term.

If you don't know the name of the particular man page, you can search the topics by entering:

```
$ man -k topic
```

Replace *topic* with a word that would be contained in the description of the man page you might be looking for. For example:

```
$ man -k "directory service"
```

Returns references to the `dscacheutil`, `dscl`, and `whois` man pages. You can also find links to related man pages at the bottom of a given man page in the "SEE ALSO" section.

If you have the Xcode tools installed, you can view man pages from within Xcode by selecting "Open man page…" from the Help menu. There are also several third-party graphical Mac OS X applications available for viewing man pages. You can find one by choosing Mac OS X Software from the Apple menu and then searching for "man page."

Not all commands and tools have man pages. Some tools use `info` pages instead, and some have no documentation at all. For more information about `info` pages, see "Using Info Pages" on page 18.

You can also access command information using the `help` command, and sometimes information is displayed if you enter the command without options or parameters.

Using Info Pages

Some commands use info pages to display their documentation. Primarily these are software packages that come from the GNU project. `info` is a tool for reading Texinfo files from the command line. To use an `info` page, enter the `info` command followed by the name of the tool:

```
server:/ mariah$ info emacs
```

You can navigate to nodes with the cursor and then press Return to go to them, or type `menu` followed by the node name. The following commands provide basic navigation between info nodes:

Key Command	Results
n	Navigates to the next page
p	Returns to the previous page
u	Navigates up one level of nodes
l	Returns to the last node visited
q	Quits the `info` program

Using the Command-Line Shell Interactively

2

Use this chapter to learn about using the command-line by typing in commands.

You can use the command-line environment in Mac OS X and Mac OS X Server interactively by typing a command and waiting for a result, or you can use the shell to compose scripts that run without direct interaction. This chapter discusses using the command-line environment interactively.

For more information about using a particular shell interactively, see the man page for that shell.

Standard Pipes

Many commands can receive text input from the user and print text to the console. They do so using *standard pipes,* which are automatically created by the shell and passed to the command.

Standard pipes include:

- `stdin`—The standard input pipe is where command input enters a command. By default, the user enters input from the command-line interface. You can redirect the output from files or other commands to `stdin`.
- `stdout`—The standard output pipe is where command output is sent. By default, command output is sent to the command line. You can redirect the output from the command line to other commands and tools.
- `stderr`—The standard error pipe is where error messages are sent. By default, errors are displayed on the command line along with standard output.

Redirecting Input and Output

From the command line, you can redirect input and output from a command to a file, or to another command.

Redirect output from the command if you want to capture the results of running the command and store it in a file for later use. Similarly, redirect input from a file to the command if you want to provide the command with preset input data, instead of needing to enter that data.

Use the following characters to redirect input and output:

Redirect	Description
>	Use a right angle bracket to redirect command output to a file.
<	Use a left angle bracket to use the contents of a file as input to the command.
>>	Use two right angle brackets to append output from a command to a file.

In addition to using file redirection, you can also redirect the output of one command to the input of another using the vertical bar character, or *pipe*. You can combine commands in this manner to implement more sophisticated versions of the same commands.

For example, the command `man bash | grep commands` passes the formatted contents of the `bash` man page to the `grep` tool, which searches those contents for lines containing the word "commands." The result is a list of lines with the specified text, instead of the entire man page.

For more information about redirection, see the `bash` man page.

Correcting Typing Errors

You can use the Left and Right Arrow keys to correct typing errors before you press Return to execute a command.

To correct a typing error:

1. Press the Left or Right Arrow key to skip backward or forward over parts of the command you don't want to change.
2. Press Delete to remove characters.
3. Type regular characters to insert them.
4. Press Return to execute the command.

To ignore what you entered and start again, press Control-U.

Using Environment Variables

The shell uses environment variables to store information, such as the name of the current user, the name of the host computer, and the default paths to any commands. Environment variables are inherited by all commands executed in the shell's context, and some commands depend on environment variables.

You can create environment variables and use them to control the behavior of a command without modifying the command itself. For example, you can use an environment variable to have a command print debug information to the console.

To set the value of an environment variable, use the appropriate shell command to associate a variable name with a value. For example, to set the variable `PATH` to the value `/bin:/sbin:/user/bin:/user/sbin:/system/Library/`, you would enter the following command in a Terminal window:

```
$ PATH=/bin:/sbin:/user/bin:/user/sbin:/system/Library/ export PATH
```

This modifies the environment variable `PATH` with the value assigned.

To view all environment variables, enter:

```
$ env
```

When you launch an application from a shell, the application inherits much of the shell's environment, including exported environment variables. This form of inheritance can be a useful way to configure the application dynamically. For example, your application can check for the presence (or value) of an environment variable and change its behavior accordingly.

Different shells support different semantics for exporting environment variables. For information, see the man page for your preferred shell.

Although child processes of a shell inherit the environment of that shell, shells are separate execution contexts that don't share environment information with each other. Variables you set in one Terminal window aren't set in other Terminal windows.

After you close a Terminal window, variables you set in that window are gone. If you want the value of a variable to persist across sessions and in all Terminal windows, you must set it in a shell startup script. For information about modifying your bash shell startup script (~bashrc) to keep variables and other settings across multiple sessions, see the "Invocation" section of the `bash` man page.

Another way to set environment variables in Mac OS X is with a property list file in your home folder. When you log in, the computer looks for a ~/.MacOSX/environment.plist file. If the file is present, the computer registers the environment variables in the property list file.

Repeating Commands

To repeat a command, press the Up Arrow key until you see the command, then make any modifications and press Return.

Including Paths Using Drag and Drop

To include a fully qualified filename or folder path in a command, you can drag the file or folder from a Finder window to the Terminal window.

Scripting the Command Line

3

Instead of entering commands and waiting for their responses, you can compose scripts that are run without direct interaction.

This chapter discusses some basics of shell scripting on Mac OS X, including automation and scheduling, as well as a brief overview of what a shell script is. It does not provide information on writing shell scripts in general.

For information about how to write shell scripts, see the *Shell Scripting Primer* on the Apple Developer Connection website.

What is a Shell Script?

A shell script is a text file that contains one or more UNIX commands. You run a shell script to perform commands you might otherwise run interactively at the command line.

Shell scripts are useful because you can combine many common tasks into one script, saving you time and possible errors when running similar tasks over and over. They can also be easily automated using tools such as `launchd` or Apple Remote Desktop.

A shell script begins with a character combination that identifies it as a shell script, the characters '#' and '!' (together called a "shebang") followed by a reference to the specific shell that the script should be run with. For example, here's the first line of a shell script that would be run with `sh`:

```
#!/bin/sh
```

You should document your shell scripts with comments. To make a comment, start the line with the number sign (#). Every line of a comment needs to begin with the number sign:

```
#This program returns the
#contents of my Home folder
```

You can put blank lines in a shell script to help visually distinguish different sections of the script.

You need to use the `chmod` tool to indicate to the operating system that the text file is executable (that is, its contents can be run as a program). To make a shell script executable:

```
chmod 755 YourScriptName.sh
```

After making the shell script executable, you can run it by entering its pathname. For example:

```
~/Documents/Dev/YourScriptName.sh
```

or

```
cd ~/Documents/Dev/
./YourScriptName.sh
```

For more information about using `chmod`, see its man page. For more information about running your shell scripts, see "Executing Commands and Running Tools" on page 13.

Monitoring and Restarting Critical Services with launchd

Mac OS X includes a system for monitoring and running critical service, which you may want to use to run various shell scripts. This system is uses a daemon named `launchd`. During system startup, `launchd` is the first process the kernel runs to set up the computer. In Mac OS X Server, your daemon should be started by `launchd`. Other mechanisms for starting daemons and services are subject to removal at Apple's discretion.

You can get an idea of the various processes run by `launchd` by looking at the following configuration files:

Folder	Usage
/System/Library/LaunchDaemons/	Apple-supplied system daemons
/System/Library/LaunchAgents/	Apple-supplied agents that apply to all users on a per-user basis
/Library/LaunchDaemons/	Third-party system daemons
/Library/LaunchAgents/	Third-party agents that apply to all users on a per-user basis
~/Library/LaunchAgents/	Third-party agents that apply to the logged-in user only

You do not interact with `launchd` directly—instead you use `launchctl` to load or unload `launchd` daemons and agents.

Note: In earlier versions of Mac OS X and Mac OS X Server, system administrators used the `watchdog` daemon to monitor critical services or modified the `rc` scripts. These are no longer supported and should be replaced with calls using `launchd`.

In earlier versions of Mac OS X and Mac OS X Server, system administrators used the `watchdog` daemon to monitor critical services or modified the `rc` scripts. These are no longer supported and should be replaced with calls using `launchd`.

For more information about `launchd`, see the `launchd` and `launchctl` man pages. Also see Technical Note TN2083: Daemons and Agents on the Apple Developer Connection.

Scheduling a Shell Script to Run at Specific Times

To schedule a shell script to run at predefined times, use either `launchd` or the `cron` tool. `cron` is a daemon that executes scheduled commands defined in crontab files.

Using cron to schedule a task

The `cron` tool searches the /var/cron/tabs/ folder for crontab files named after accounts in /etc/passwd, and loads the files into memory. The `cron` tool also searches for crontab files in the /etc/crontab/ folder, which are in a different format. `cron` then cycles every minute, examining stored crontab files and checking each command to see if it should be run in the current minute.

When commands execute, output is mailed to the owner of the crontab file or to the user named in the optional MAILTO environment variable in the crontab file.

If you modify a `crontab` file, you must restart `cron`.

You use `crontab` to install, deinstall, or list the tables used to drive the `cron` daemon. Users can have their own crontab file.

To configure your crontab file, use the `crontab -e` command. This displays an empty crontab file.

Here's an example of a configured crontab file:

```
SHELL=/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin
HOME=/var/log
#min hour mday month wday command
30 18 * * 1-5 diskutil repairPermissions /Volumes/MacHD
50 23 * * 0 diskutil repairVolume /Volumes/MacHD
```

The first crontab entry repairs disk permissions for the MacHD volume at 18:30 every day, Monday through Friday:

```
30 18 * * 1-5 diskutil repairPermissions /Volumes/MacHD
```

The second crontab entry schedules a repair volume operation to run at 23:50 every Sunday:

```
50 23 * * 0 diskutil repairVolume /Volumes/MacHD
```

Scheduling tasks with launchd

You can use `launchd` instead of `cron` to schedule tasks. With `launchd`, if a task is skipped because the computer is shut off or asleep, the task is added to the queue when the computer comes back online. To use `launchd` to schedule timer-based jobs, use the *`StartCalendarInterval`* or *`StartInterval`* key.

For more information about `launchd`, see the `launchd` man page.

Connecting to Remote Computers 4

Learn about using the command-line on computers remotely.

If you need to run command-line tools on remote computers, there are tools to help you. This chapter discusses some of the most commonly used tools and provides some tips for getting started. It also describes three methods for connecting to the command-line environment of a remote computer:

- SSH
- Apple Remote Desktop
- X11

SSH

SSH (Secure Shell) lets you send secure, encrypted commands to a computer remotely, as if you were sitting at the computer. You use the `ssh` tool in Terminal to open a command-line connection to a remote computer, and while the connection is open, you enter commands to be performed on the remote computer.

You can also use any other application that supports SSH to connect to a computer running Mac OS X or Mac OS X Server.

How SSH Works

SSH works by setting up encrypted tunnels using public and private keys. Here's a description of an SSH session:

- The local and remote computers exchange public keys.

  If the local computer has never encountered a given public key, SSH and your web browser prompt you to accept the unknown key.
- The two computers use the public keys to negotiate a session key used to encrypt subsequent session data.

- The remote computer attempts to authenticate the local computer using RSA or DSA certificates. If this isn't possible, the local computer is prompted for a local username and password.
- After successful authentication, the session begins. A remote shell, a secure file transfer, a remote command, or other action can take place through the encrypted tunnel.

The following are SSH tools:

- `sshd`—A daemon that acts as a server to all other commands
- `ssh`—The primary user tool, which includes a remote shell, remote command, and port-forwarding sessions
- `scp`—Secure copy, a tool for automated file transfers
- `sftp`—Secure FTP, a replacement for FTP

Generating Key Pairs for Key-Based SSH Connections

By default, SSH supports the use of password, key, and Kerberos authentication. The standard method of SSH authentication is to supply a user name and password as login credentials. Identity key-based authentication lets you log in to the server without supplying a password.

Key-based authentication is more secure than password authentication, because it requires that you have the private key file and know the password that lets you access that key file. A key must be generated for each user account that needs to use `ssh`.

How SSH key-based authentication works:

1. A private and a public key are generated, each associated with a user name to establish that user's authenticity.
2. When you attempt to log in as that user, the user name is sent to the remote computer.
3. The remote computer looks in the user's .ssh/ folder for the user's public key.

   This folder is created when using SSH the first time.
4. A challenge is sent to the user based on his or her public key.
5. The user verifies his or her identity by using the private portion of the key pair to decode the challenge.
6. After the key is decoded, the user is logged in without a password.

   This is especially useful when automating remote scripts.

   Note: If the server uses FileVault to encrypt the home folder of the user you want to use SSH to connect as, you must be logged in on the server to use SSH. Alternatively, you can store the keys for the user in a location that isn't protected by FileVault, but this isn't secure.

To generate the identity key pair:

1. Enter the following command on the local computer:

```
$ ssh-keygen -t dsa
```

2. When prompted, enter a filename in the user's home folder to save the keys in; then enter a password and password verification. For no password, don't enter anything when prompted. Just press Return.

   For example:

```
Generating public/private dsa key pair.
Enter file in which to save the key (/Users/mariah/.ssh/id_dsa): frog
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in frog.
Your public key has been saved in frog.pub.
The key fingerprint is:
4a:5c:6e:9f:3e:35:8b:e5:c9:5a:ac:00:e6:b8:d7:96 mariahjohnson1@mac.com
```

   This creates two files. Your identification or private key is saved in one file (*frog* in our example) and your public key is saved in the other (*frog*.pub in our example).

   The key fingerprint, which is derived cryptographically from the public key value, also appears. This secures the public key, making it computationally infeasible for duplication.

3. Copy the resulting public file, which contains the local computer's public key, to the .ssh/authorized_keys file in the user's home folder on the remote computer (~/.ssh/authorized_keys).

   The next time you log in to the remote computer from the local computer, you won't need to enter a password.

   If you need to establish two-way communication between servers, repeat this process on the second computer.

   This process must be repeated for each user who needs to be able to open a key-based SSH session. This includes the root user, whose home folder on Mac OS X Server is at /var/root/.

   Note: If you're using an Open Directory user account and have logged in using the account, you don't need to supply a password for SSH login. On computers with Mac OS X Server, SSH uses Kerberos for single sign-on authentication with any user account that has an Open Directory password. (Kerberos must be running on the Open Directory server.) For more information, see *Open Directory Administration*.

A Key-Based SSH Scripting Example

A cluster of servers is an ideal environment for using key-based SSH. The following Perl script is a trivial scripting example, and it shouldn't be implemented. It demonstrates connecting over an SSH tunnel to each server defined in the variable serverList, running softwareupdate, installing available updates, and restarting each server if necessary. The script assumes that key-based SSH has been properly set up for the root user on all servers to be updated.

```
#!/usr/bin/perl
# \@ is the escape sequence for the "@" symbol.
my @serverList = ('root\@exampleserver1.example.com',
'root\@exampleserver2.example.com');
foreach $server (@serverList) {
open SBUFF, "ssh $server -x -o batchmode=yes 'softwareupdate -i -a' |";
while(<SBUFF>) {
my $flag = 0;
chop($_);
#check for restart text in $_
my $match = "Please restart immediately";
$count = @{[$_ =~ /$match/g]};
if($count > 0) {
$flag = 1;
}
}
close SBUFF;
if($flag == 1) {
\Qssh $server -x -o batchmode=yes shutdown -r now\Q
}
}
```

Updating SSH Key Fingerprints

The first time you connect to a remote computer using SSH, the local computer prompts for permission to add the remote computer's fingerprint (or encrypted public key) to a list of known remote computers. You might see a message like this:

```
The authenticity of host "server1.example.com" can't be established.
RSA key fingerprint is a8:0d:27:63:74:f1:ad:bd:6a:e4:0d:a3:47:a8:f7.
Are you sure you want to continue connecting (yes/no)?
```

The first time you connect, you have no way of knowing whether this is the correct host key. Most people respond "yes." The host key is then inserted into the ~/.ssh/known_hosts file so it can be verified in later sessions.

Be sure this is the correct key before accepting it. If possible, provide users with the encryption key through FTP, mail, or a download from the web, so they can be sure of the identity of the server.

If you later see a warning message about a *man-in-the-middle* attack (see below) when you try to connect, it might be because the key on the remote computer no longer matches the key stored on the local computer. This can happen if you:

- Change your SSH configuration on the local or remote computer
- Perform a clean installation of the server software on the computer you're attempting to log in to using SSH
- Start up from a Mac OS X Server disc on the computer you're attempting to log in to using SSH
- Attempt to use SSH to access a computer that has the same IP address as a computer that you used SSH with on another network

To connect again, delete the entries corresponding to the remote computer (which are stored by name and IP address) in the file ~/.ssh/known_hosts.

An SSH Man-in-the-Middle Attack

Sometimes an attacker can access your network and compromise routing information, so that packets intended for a remote computer are routed to the attacker, who then impersonates the remote computer to the local computer and the local computer to the remote computer.

Here's a typical scenario: A user connects to the remote computer using SSH. By using spoofing techniques, the attacker poses as the remote computer and receives information from the local computer. The attacker then relays the information to the remote computer, receives a response, and then relays the remote computer's response to the local computer.

Throughout the process, the attacker is privy to all information that goes back and forth, and can modify it.

If you see the following message when connecting to the remote computer using SSH, it may indicate a man-in-the-middle attack.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Protect against this type of attack by verifying that the host key sent back is the correct host key for the computer you're trying to reach. Be watchful for the warning message, and alert your users to its meaning.

Important: Removing a host key from the known_hosts file bypasses a security mechanism that would help you avoid imposters and man-in-the-middle attacks. Before you delete a host key from the known_hosts file, be sure you understand why the key on the remote computer has changed.

Controlling Access to SSH Service

You can use Server Admin to control which users can open a command-line connection using the `ssh` tool in Terminal. Users with administrator privileges can always open a connection using SSH.

For information about controlling access to the SSH service, see *Open Directory Administration*.

Connecting to a Remote Computer Using SSH

Use the `ssh` tool to create a secure shell connection to a remote computer.

To access a remote computer using ssh:

1. Open Terminal.
2. Log in to the remote computer by entering:

   ```
   $ ssh -l username server
   ```

   Replace *username* with the name of an administrator user on the remote computer. Replace *server* with the name or IP address of the remote computer. For example:

   ```
   $ ssh -l mariah 10.0.1.2
   ```

   If this is the first time you're connecting to the remote computer, you're prompted to continue connecting after the remote computer's RSA fingerprint appears.

   Enter `yes`.
3. When prompted, enter the user's password for the remote computer.

   The command prompt changes to show that you're connected to the remote computer. In the previous example, the prompt might look like this:

   ```
   10.0.1.2:~ mariah$
   ```
4. To send a command to the remote computer, enter the command.
5. To close the remote connection, enter `logout`.

   You can authenticate and send a command using a single line, by appending the command to the basic `ssh` tool. For example, to delete a file you could enter:

   ```
   $ ssh -l mariah server1.example.com rm /Users/mariah/Documents/report
   ```

   or

   ```
   $ ssh -l mariah@server1.example.com "rm /Users/mariah/Documents/report"
   ```

   You're prompted for the user's password.

Apple Remote Desktop

Apple Remote Desktop is a software package that's available separately from Mac OS X Server. Apple Remote Desktop provides a command for sending a shell script or command to client computers, which lets you easily distribute and automate shell scripts. For more information, see the "UNIX Shell Commands" section of the *Apple Remote Desktop Administrator Guide*.

X11

X11 is the traditional windowing system of UNIX systems. If you're working in an environment where you need to support X11-based applications, you can use them with Mac OS X Server, but you first need to install the X11 package. The X11 server and an application to access X windows from the Finder are available as an optional installation in the Optional Installs folder of your installation disc (X11 is in the Applications package). Once the package is installed, you can access an X-based terminal by launching the X11 application in /Applications/Utilities/.

The X11 implementation in Mac OS X Server is based on the X.org foundation release, and is X11R7 compatible.

X11 uses a different security model than the default model in Mac OS X Server. For more information, see the X11 Preferences Security pane and this article on the Apple Developer Connection website:

"Configuring and Running X11 Applications on Mac OS X"

Common Command-Line Tasks 5

This chapter discusses some of the most frequently used command-line task.

If you're new to the command-line environment, it helps to understand some common scenarios in which people frequently use the shell. This section explores some of those areas and provides some guidance on getting started using the shell in these situations.

Editing Configuration Files

A common use of the command line is to manually edit configuration files to enable functionality that isn't exposed in Server Admin or Workgroup Manager. In server documentation, for example, you may be instructed to modify Property Lists (plists) or other regular text files to incorporate additional functionality or enforce enhanced security settings. If you're unfamiliar with using the command line to edit text files, there are a few things to understand:

- How to choose an appropriate text editor
- How to edit property list (plist) files
- How to save text files so they can be used by the UNIX subsystem of Mac OS X

These topics are discussed below.

Text Editors

To edit a plain text file, use a text editor. Text editors are among the oldest programs available on any operating system, and come in a wide variety—from completely automatic text editors, where you essentially write a recipe for what actions should be taken on text and then let the computer do the work, to much more interactive text editors that can edit (and save) text in a wide variety of formats.

For general-purpose work, it's easiest to deal with one of the text editors included with Mac OS X. If you want to use a graphical text editor, use TextEdit (in /Applications/); otherwise, use one of the many command-line editors provided. The three most full-featured command-line text editors included with Mac OS X are:

nano Nano is a simple command-line based editor. It's a replacement for the Pico editor, so instructions for using the Pico editor can be used with nano. If you invoke the pico editor, you actually run nano. Nano is a good introduction to using a command-line based editor as it includes easy-to-follow on-screen help.

vim Vim is a vi-compatible text editor. It has many powerful enhancements for moving around, searching, and editing documents. Basic editing is simple to learn and there is much additional functionality to explore. Most functionality is accessed by typing combinations of keystrokes that trigger certain behavior. Vim, or the editor it's modeled after, vi, is found in most UNIX-based operating systems. If you'll be doing lots of editing from the command line, it's a good editor to learn to use, but if you only use a command-line based editor occasionally, you can get by without learning it.

Emacs Like vim, Emacs is an extremely full-featured editor found on most UNIX-based systems. In addition to its editing power, Emacs is extremely customizable, with additional functionality available in modules that let the Emacs interface do much more than just text editing. It's relatively easy to do basic editing with, and has an incredible depth of functionality for the dedicated user to explore. Like vim, Emacs uses keystroke combinations to access its many different functional behaviors. These require memorization to be most useful, so Emacs is most useful for people who use the command line very often.

If you're new to using the command line and don't anticipate using it much for editing, nano is probably your best choice. If you expect to spend a lot of time using the command-line environment, it's probably worth learning either vim or Emacs. They have very different design philosophies, so spend some time with each of them to determine which works best for you. For more information about using nano, vim, or Emacs, see their man pages.

You invoke a command-line editor by typing the name of the editor, followed by a space and then the name of the file you want to open. If you want to create a new file, type a name for the file. Designate where the file is located, as described in "Specifying Files and Folders" on page 15. Here's an example of using nano to open a new file named "myFile.conf" in your Documents folder:

```
$ nano ~/Documents/myFile.conf
```

Saving Text Files for UNIX Execution

When you edit text files for execution by UNIX utilities, you need to save the files properly so that they can be used (or executed) by their calling program. It's especially important to use plain text and ensure that the privileges are correct.

Using plain text

Many graphical text editors, including TextEdit, save text files in a more complex format than most UNIX programs expect. If you're using TextEdit to edit text-based configuration files, save them as Plain Text, not the default Rich Text Format. To change the default format of text documents in TextEdit you have two options:

- To save all documents as plain text, select "Plain text" under Format in the New Document pane of TextEdit preferences.
- To change the format of an individual document, choose "Make Plain Text" from the Format menu.

Although Rich Text Format may appear to be simple text in an editor, it's actually a full specification that describes formatting, colors, fonts, and other information that isn't contained in the plain text files that most UNIX programs expect. To see what's actually contained in a Rich Text Format document, save one in TextEdit, and then open the same file in a command-line text editor.

Editing Property Lists

Many preference and configuration files in Mac OS X use property lists (plists) to specify the attributes, or properties, of an application or process. An example is the Finder's preferences plist in the Library/Preferences/ folder of a user's home folder. The file is named com.apple.Finder.plist. The default naming convention for a plist includes the distributor's reverse DNS name prepended to the application or process name, followed by a".plist" extension.

Property lists are binary files that you can edit using the following tools:

- Property List Editor is a graphical application that's a part of the Xcode developer tools. You can get the Xcode tools from developer.apple.com. Property List Editor is most useful if you already understand property lists and their conventions.
- `PlistBuddyPlistBuddy` is a command-line tool for directly reading and modifying values inside a property list without the need to convert the property list to an intermediary format.
- `defaults` is a command-line tool that you can use to edit property lists.

  The `defaults` command is a powerful tool, with functionality beyond simple editing of property lists. When you know the specific key and value in a property list that you need to change, it's very efficient.

- `plutil` is a command-line tool that you can use to change a property list into a format you can edit with a text editor, and then change back to its binary format.

Using PlistBuddy to edit property lists

The `PlistBuddy` command is designed to easily read and modify values in a property list. If you know the values to set or read, you can quickly make changes with `PlistBuddy`. `PlistBuddy` works on specific property list files.

This example shows how to use the `PlistBuddy` command interactively to change the orientation of the Dock for a local user:

1. Determine the names of the appropriate property list, key, and values. In this case, the name for the Dock's property list is `com.apple.Dock.plist`. If you were editing the Dock property list for the user alecjones, the path would be:

   ```
   /Users/alecjones/Library/Preferences/com.apple.Dock.plist
   ```

2. Enter in the following command to enter the `PlistBuddy` interactive mode:

   ```
   PlistBuddy /Users/alecjones/Library/Preferences/com.apple.Dock.plist
   ```

   If the path to `PlistBuddy` isn't in your default paths, you need to add it or explicitly call it as follows:

   ```
   /usr/libexec/PlistBuddy ~/Library/Preferences/com.apple.Dock.plist
   ```

   See "Executing Commands and Running Tools" on page 13.

   If the file you're trying to edit doesn't exist, `PlistBuddy` creates the file in the designated location.

3. In interactive mode, you can choose from many commands. To set or change the orientation of the Dock to the left side of the screen, enter:

   ```
   Set :orientation left
   ```

4. Save and exit:

   ```
   Save
   Exit
   ```

   `PlistBuddy` can also be run non-interactively. To make the same change without invoking interactive mode:

   ```
   /usr/libexec/PlistBuddy -c "Set :orientation left" ~/Library/Preferences/
   com.apple.Dock.plist
   ```

   Both examples above assume the *`orientation`* key already exists. This isn't necessarily true for a new user in Mac OS X version 10.6. Don't assume that a value exists. First, confirm it with the `Print` command. Otherwise, you need to use the `Add` command, which also requires designating a type.

There are many other options for `PlistBuddy` that are invoked in a similar manner. For information about `PlistBuddy`, see its man page.

Using the defaults command to edit property lists

The `defaults` tools works directly with the Mac OS X preferences subsystem and is used by many applications in Mac OS X to manage preferences and other settings. It can be built into shell scripts and allows you to access preferences in the multiple domains that exist on a given computer.

1. Determine the names of the appropriate property list, key, and values. For example, the name for the Dock's property list is `com.apple.Dock.plist`. (When invoking the `defaults` command, omit the `.plist` extension.)
2. Using the values you have determined or been given, enter their values following the `defaults` command:

```
defaults write com.apple.dock orientation left
```

3. In most cases, you need to restart the application or process. A simple way to do this is to use Activity Monitor to select the appropriate process, and then click Quit Process. For this example, you would choose the process named Dock.

For information about `defaults`, see its man page.

Using plutil and a text editor to edit property lists

In Mac OS X v10.6, plist files are stored in a binary format. If you want to edit them with a text editor, you must first convert them to plain text. To convert a plist file to plain text, use the `plutil` command:

```
plutil -convert xml1 com.apple.dock.plist
```

This results in an XML text file that you can edit. When you're done, convert the file back to binary format:

```
plutil -convert binary1 com.apple.dock.plist
```

Before making any changes to plist files using `plutil`, make a backup copy of the files. Do this in the Finder, or use the `cp` command:

```
cp com.apple.finder.plist com.apple.dock.plist.bak
```

For information about Property Lists, see the `plist` man page. For the basics of using command-line tools, see Chapter 1, "Introduction to the Command-Line Environment."

Moving and Copying Files

You can move and copy files locally or remotely using the `mv`, `cp`, and `scp` commands.

Moving a file or folder locally

To move files or folders from one location to another on the same computer, use the `mv` command. The `mv` command moves the file or folder from its old location and puts it in the new location.

For example, to move a file from your Downloads folder to a Work folder in your Documents folder:

```
mv ~/Downloads/MyFile.txt ~/Documents/Work/MyFile.txt
```

You can also change the name of the file as it's moved:

```
mv ~/Downloads/MyFile.txt ~/Documents/Work/NewFileName.txt
```

For more information about the `mv` command, see its man page.

Copying a file or folder locally

To make a copy of a file, use the `cp` command.

For example, to copy a folder named "Expenses" in your Documents folder to another volume named "Data":

```
cp ~/Documents/Expenses /Volumes/Data/Expenses
```

You can also change the name of the folder as it's being moved:

```
cp ~/Documents/Expenses /Volumes/Data/Current_Expenses
```

For more information about the `cp` command, see its man page.

Copying a file or folder remotely

To copy a file or folder to or from a remote computer, use the `scp` command. `scp` uses the same underlying protocols as `ssh`. For more information about SSH, see "Controlling Access to SSH Service" on page 22.

For example, to copy a compressed file from your home folder to the ladmin user's home folder on a remote server:

```
scp -E ~/ImportantPapers.tgz ladmin@remoteserver.com:/Users/ladmin/
    Desktop/ImportantPapers.tgz
```

You're prompted for the ladmin user's password.

The '-E' flag preserves extended attributes, resource forks, and ACL information.

For more information about the `scp` command, see its man page.

Compressing and Uncompressing File Archives

Mac OS X and Mac OS X Server use the GNU `tar` utility to compress and uncompress files and folders. When sending folders and multiple files between computers, it's helpful to compress them into a single archive. This saves space, allows you to transfer just one item instead of many, and makes it easier to resume in case the task is suspended for some reason.

The `tar` utility has many options, but for a basic compression of a folder named "LotsOfFiles," you could simply enter:

```
tar -czf LotsOfFiles.tgz LotsOfFiles
```

If it's a large folder, you may want to monitor the process by adding the 'v' flag:

```
tar -czvf LotsOfFiles.tgz LotsOfFiles
```

To open an archive, use the 'x' flag. The 'v' flag is useful to watch what's going on:

```
tar -xzvf LotsOfFiles.tgz
```

The 'z' flag indicates that the archive is being compressed, as well as being combined into one file. Usually you'll use this option, but you aren't required to. The traditional file extension for a compressed archive is `.tgz`, although you might also see files ending in `.tar.gz`. If the archive isn't compressed, it usually just ends in `.tar`.

Files created with `tar` can be opened in the Finder by double-clicking them. Also, if you use the File > Compress menu command in the Finder to compress a folder or file, the tar file can be opened using `tar` from the command line.

For more information about the `tar` command, see its man page.

Viewing File Contents

If you want to look at the contents of a text-based configuration file, you can use `cat` or `less`. Generally, you'll use `less` because it has more options (like searching).

To use `less`, type the command name followed by the name of the file you want to view. The first page of text fills the window. To view the next page, press the Space bar.

`less` also lets you search in a file. Type '/' followed by the phrase you're searching for. If the phrase has spaces in it, precede each space with '\':

```
/I\ read\ the\ other\ day
```

The following table lists some other useful keys for navigating the output from `less`.

Key Command	Action
J or Down Arrow	Scroll down a line
K or Up Arrow	Scroll up a line
N	Find the next occurrence of a search term
P	Find the previous occurrence of a search term
Q	Quit `less`

For more information about the `less` command, see its man page.

Searching for Text in a File

To locate a string within a file, use the `grep` tool. The `grep` tool searches the named input files for lines containing a match to the given pattern. By default, `grep` prints the matching lines.

To search for a unique string in a file:

```
$ grep search_string filename
```

Replace *search_string* with the the string to search for, and replace *filename* with the name of the file whose contents you want to search.

Backing Up and Restoring

Time Machine is extremely useful for user backups, but server administrators might have different backup requirements. Mac OS X Server provides several command-line tools for backing up and restoring data:

- `rsync`—Use this command to keep a backup copy of your data in sync with the original. `rsync` copies only the files that have changed.
- `ditto`—Use this command to perform full backups.
- `asr`—Use this command to back up and restore an entire volume.

For more information about these commands, see their man pages.

Note: You can use these commands with the `launchctl` command to automate data backup.

Accessing Apple Hardware from the Command Line

6

Learn how to access hardware-level controls like restarting, shutting down, powering up, and selecting boot options from the command line.

This chapter introduces commands for shutting down or restarting a local or remote computer. Computers need to be shut down or restarted, whether locally or remotely, when installing tools or making computer repairs.

Restarting a Computer

To restart a computer at a specific time, use the `reboot` or `shutdown -r` command. For more information, see their man pages.

To restart the local computer:

```
$ shutdown -r now
```

To restart a remote computer immediately:

```
$ ssh -l root computer shutdown -r now
```

To restart a remote computer at a specific time:

```
$ ssh -l root computer shutdown -r hhmm
```

Parameter	Description
computer	The IP address or DNS name of the computer
hhmm	The hour and minute when the computer restarts

Automatic Restart

You can also use the `systemsetup` tool to set the computer to start up after a power failure or system freeze, by specifying a number of seconds:

```
systemsetup -setwaitforstartupafterpowerfailure seconds
```

Parameter	Description
seconds	The number of seconds before the computer starts after a power failure. This value must be a multiple of 30.

Changing a Remote Computer's Startup Disk

You can change a remote computer's startup disk using SSH.

To determine available startup volumes:

Log in to the remote computer using SSH, and enter:

```
systemsetup -liststartupdisks
```

To change the startup disk:

Log in to the remote computer using SSH, and enter:

```
CodeLinesystemsetup -setstartupdisks /Volumes/SnowLeopardServerHD/System/
     Library/CoreServices
```

For information about using SSH to log in to a remote computer, see "SSH" on page 27.

Shutting Down a Computer

To shut down a computer at a specific time, use the `shutdown` tool. For more information, see the `shutdown` man page.

To shut down a remote computer immediately:

```
$ ssh -l root computer shutdown -h now
```

To shut down the local computer in 30 minutes:

```
$ shutdown -h +30
```

Parameter	Description
computer	The IP address or DNS name of the computer

Shutting Down While Leaving the Computer On and Powered

To support UPS restart after power failure, the `shutdown` tool provides the `-u` option. This option halts system shutdown before the `shutdown` tool instructs the power manager to turn off the power supply.

The `-u` option keeps the system halted and waits for 5 minutes before removing power so an external UPS can forcibly remove power.

Using the `-u` option simulates a dirty shutdown, which allows a later automatic power-on. The operating system uses the `-u` option with supported UPS devices in emergency shutdowns.

Manipulating Open Firmware NVRAM Variables

To manipulate Open Firmware NVRAM variables, use the `nvram` tool. If you change a value with `nvram`, the value is saved only if the computer cleanly restarts or shuts down.

To view NVRAM variables:

```
$ nvram -p
```

For more information, see the `nvram` man page.

Remotely Controlling the Xserve Front Panel

You can use the `ipmitool` command to remotely control the front panel of an Xserve.

To display the list of supported virtual front panel commands:

```
$ ipmitool chassis bootdev
bootdev <device> [clear-cmos=yes|no]
none : Do not change boot device order
pxe : Force PXE boot (LOM: Force boot NetBoot server)
disk : Force boot from default Hard-drive
safe : Force boot from default Hard-drive, request Safe Mode (LOM: Not
     used)
diag : Force boot from Diagnostic Partition (LOM: Force boot diagnostic
     mode from NetBoot server)
cdrom : Force boot from CD/DVD
bios : Force boot into BIOS Setup (LOM: Not used)
Lights-out Management additional options
nvram : Force reset of NVRAM
tdm : Force boot into Target Disk Mode
other : Skip current startup disk selection, and boot from other
```

Mac OS X Server v10.6 supports the following commands: `none`, `pxe`, `disk`, `diag`, `cdrom`, `nvram`, `tdm`, and `other`.

For example, enter the following command and then restart the Xserve system to start the system in Target Disk Mode:

```
$ ipmitool chassis bootdev tdm
```

After the system starts, the `ipmitool` command reverts to the default setting (none). Restarting the Xserve system without running the `ipmitool` command doesn't change the boot device order.

For more information about `ipmitool`, see its man page.

Command-Line Tools Specific to Mac OS X

The following command line tools are unique to Mac OS X or substantially different from implementations on other UNIX platforms. See their man pages for more details.

An online version of the man pages in Mac OS X and Mac OS X server is available at:

http://developer.apple.com/documentation/Darwin/Reference/ManPages/

Section 1 Man Pages

Man pages in section 1 refer to general command-line tools and utilities.

See the `intro(1)` man page for more information about this section.

`afconvert(1)`	Audio file converter
`afinfo(1)`	Audio file information
`afplay(1)`	Audio file player
`afscexpand(1)`	Decompress files compressed with HFS+ compression
`amlint(1)`	Check Automator actions for problems
`applesingle(1)`	Encode and decode files
`ar(1)`	Create and maintain library archives
`arch(1)`	Print architecture type or run selected architecture of a universal binary
`authopen(1)`	Open file with authorization
`automator(1)`	Runs Automator workflow
`auval(1)`	AudioUnit validation
`auvaltool(1)`	AudioUnit validation
`binhex(1)`	Encode and decode files
`BuildStrings(1)`	Generate header (.h) or resource (.r) file from text files
`compileHelp(1)`	Command-line utility to merge contextual help rtf snippets into one resource

`configureLocalKDC(1)`	Generate a LocalKDC
`CPlusTestRig(1)`	Runs CPlusTest unit test bundles
`CpMac(1)`	Copy files preserving metadata and forks
`createhomedir(1)`	Create and populate home directories on the local computer
`defaults(1)`	Access the Mac OS X user defaults system
`desdp(1)`	Scripting definition generator
`ditto(1)`	Copy directory hierarchies, create and extract archives
`dns-sd(1)`	Multicast DNS (mDNS) & DNS Service Discovery (DNS-SD) Test Tool
`drutil(1)`	Interact with CD/DVD burners
`dscacheutil(1)`	Gather information, statistics and initiate queries to the Directory Service cache
`dsimport(1)`	Tool for importing records into an Open Directory source
`dsmemberutil(1)`	Various operations for the membership APIs, including state dump, check memberships, UUIDs, etc
`dsymutil(1)`	Manipulate archived DWARF debug symbol files
`dwarfdump(1)`	Dump DWARF debug information
`dyldinfo(1)`	Displays information used by dyld in an executable
`emacs-undumped(1)`	Basic emacs with no ELisp libraries loaded
`FixupResourceForks(1)`	Join AppleDouble files into two-fork HFS resource files
`fs_usage(1)`	Report system calls and page faults related to filesystem activity in real-time
`fwkdp(1)`	FireWire KDP Tool
`fwkpfv(1)`	FireWire kprintf viewer
`gatherheaderdoc(1)`	Header documentation processor
`genstrings(1)`	Generate string table from source code
`GetFileInfo(1)`	Get attributes of files and directories
`hdiutil(1)`	Manipulate disk images (attach, verify, burn, etc)
`hdxml2manxml(1)`	HeaderDoc XML to MPGL translator
`headerdoc(1)`	Header documentation processor
`headerdoc2html(1)`	Header documentation processor
`hiutil(1)`	Utility for creating and examining Help Viewer indices
`hwprefs(1)`	Inspect and control low-level system and processor parameters

`javaconfig(1)`	Get Java configuration information
`javatool(1)`	Tool used in building older Java software projects
`languagesetup(1)`	Set the primary language
`latency(1)`	Monitors scheduling and interrupt latency
`launchctl(1)`	Interfaces with launchd
`ld(1)`	Linker
`locale(1)`	Display locale settings
`localedef(1)`	Define locale environment
`lookupd(1)`	Gather information, statistics and initiate queries to the Directory Service cache
`mDNS(1)`	Multicast DNS (mDNS) & DNS Service Discovery (DNS-SD) Test Tool
`macbinary(1)`	Encode and decode files
`mdcheckschema(1)`	Simple mdimporter schema validation tool
`mdfind(1)`	Finds files matching a given query
`mdimport(1)`	Import file hierarchies into the metadata datastore
`mdls(1)`	Lists the metadata attributes for the specified file
`mdutil(1)`	Manage the metadata stores used by Spotlight
`mediastreamsegmenter(1)`	Create segments from MPEG-2 Transport streams for HTTP Live Streaming
`memberd(1)`	Various operations for the membership APIs, including state dump, check memberships, UUIDs, etc.
`MergePef(1)`	Merge multiple PEF containers into one file
`migrateLocalKDC(1)`	Migrates a LocalKDC
`mnthome(1)`	Mount an AFP (AppleShare) home directory with the correct privileges
`mpgl(1)`	MPGL to mdoc (man page) translator
`MvMac(1)`	Move files while preserving metadata and forks
`netstat(1)`	Show network status
`notificationconf(1)`	
`notifyutil(1)`	Notification command line utility
`ocspd(1)`	OCSP and CRL Daemon
`open(1)`	Open files and directories

`opendiff(1)`	Use FileMerge to graphically compare or merge file or directories
`osacompile(1)`	Compile AppleScripts and other OSA language scripts
`osadecompile(1)`	Display compiled AppleScripts or other OSA language scripts
`osalang(1)`	Information about installed OSA languages
`osascript(1)`	Execute AppleScripts and other OSA language scripts
`packagemaker(1)`	Installation-package creation tool
`passwd(1)`	Modify a user's password
`pl(1)`	ASCII property list utility Extract translatable strings from source
`plutil(1)`	Property list utility
`pmset(1)`	Manipulate power management settings
`podcast(1)`	Podcast Producer command line tool
`PPCExplain(1)`	Verbose description of PowerPC mnemonics
`projectInfo(1)`	Identify build tool for software development project
`pubsub(1)`	Utility for managing RSS/Atom subscriptions via the PubSub framework
`qlmanage(1)`	Quick Look Server debug and management tool
`rebase(1)`	Changes base address of dylibs and bundles
`reggie_se(1)`	Read and modify hardware registers
`ResMerger(1)`	Merges resource forks or files into one resource file
`RezWack(1)`	Combines resource and data forks of a file into a flattened file
`RunTargetUnitTests(1)`	Run unit tests for the current target
`RunUnitTests(1)`	Run unit tests for the current target
`sandbox-exec(1)`	Execute within a sandbox
`sandbox-simplify(1)`	Simplify a sandbox profile created by a trace directive
`sar(1)`	System activity reporter
`sc_usage(1)`	Show system call usage statistics
`sdef(1)`	Scripting definition extractor
`sdp(1)`	Scripting definition (sdef) processor
`security(1)`	Command line interface to keychains and Security framework

securityd(1)	Security context daemon for Authorization and cryptographic operations
SetFile(1)	Set attributes of files and directories
sips(1)	Scriptable image processing system
SplitForks(1)	Divide a two-fork HFS file into AppleDouble format resource and data files
stackshot(1)	Capture user and kernel space stack traces, using a kernel stack trace facility
sw_vers(1)	Print Mac OS X operating system version information
symstacks.rb(1)	Capture user and kernel space stack traces, using a kernel stack trace facility
syslog(1)	Apple System Log utility
tconf(1)	TargetConfig command line tool
textutil(1)	Text utility
tiff2icns(1)	Converts TIFF to icns format
tiffutil(1)	Manipulates tiff files
UnRezWack(1)	Split a RezWack file into separate data and resource files
unwinddump(1)	Displays compact unwind information in an executable
update_dyld_shared_cache(1)	Updates dyld's shared cache
uuidgen(1)	Generates new UUID strings
vm_stat(1)	Show Mach virtual memory statistics
wai(1)	Wait for process termination
xcodebuild(1)	Build Xcode projects
xcodeindex(1)	Indexes Xcode projects
xgrid(1)	Submit and monitor xgrid jobs
xm(1)	Script to get information about the installed version of GNOME-XML
yacc(1)	Parser generator

Section 4 Man Pages

Man pages in section 4 refer to descriptions of special files and devices.

`dummynet(4)`	Traffic shaper, bandwidth manager and delay emulator
`ipfirewall(4)`	IP packet filter and traffic accounting
`random(4)`	Better random number generator; routines for changing generators
`urandom(4)`	Better random number generator; routines for changing generators

Section 5 Man Pages

Man pages in section 5 give information about file formats and conventions.

See the `intro(5)` man page for more information about this section.

`asl.conf(5)`	Configuration file for syslogd(8) and aslmanager(8)
`auto_master(5)`	Automounter master map
`autofs.conf(5)`	automount(8) and automountd(8) configuration file
`bom(5)`	Bill of materials
`bootparams(5)`	Boot parameter database
`bootptab(5)`	Internet Bootstrap Protocol server database
`compat(5)`	Manipulate compatibility settings
`fstab(5)`	Static information about the filesystems
`group(5)`	Format of the group permissions file
`launchd.conf(5)`	launchd(8) configuration file
`launchd.plist(5)`	System wide and per-user daemon/agent configuration files
`manpages(5)`	An introduction to manual pages
`plist(5)`	System wide and per-user daemon/agent configuration files property list format
`ranlib(5)`	Archive (library) table-of-contents format
`resolver(5)`	Resolver configuration file format
`sdef(5)`	Scripting definition file
`stab(5)`	Symbol table types
`types(5)`	Mime type description file for cups system data types

Section 7 Man Pages

Man pages in section 7 are miscellaneous pages that don't belong in any other section.

See the `intro(7)` man page for more information about this section.

`sandbox(7)`	Overview of the sandbox facility

Section 8 Man Pages

Man pages in section 8 document commands that system administrators would invoke as well as daemons.

See the `intro(8)` man page for more information about this section.

`agvtool(8)`	Apple-generic versioning tool for Xcode projects
`aosnotifyd(8)`	Apple Online Services notification daemon
`appleprofilepolicyd(8)`	AppleProfileFamily access control daemon
`aslmanager(8)`	Configuration file for syslogd(8) and aslmanager(8) Apple System Log data store file manager
`asr(8)`	Apple Software Restore; copy volumes (e.g. from disk images)
`atrun(8)`	Run jobs queued for later execution
`autodiskmount(8)`	Disk support tool
`autofsd(8)`	Daemon to update autofs mounts on network changes
`automount(8)`	automount(8) and automountd(8) configuration file mount autofs on the appropriate mount points
`automountd(8)`	automount(8) and automountd(8) configuration file automatic mount / unmount daemon for autofs
`bless(8)`	Set volume boot-ability and startup disk options
`blued(8)`	The Mac OS X bluetooth daemon
`bootpd(8)`	DHCP/BOOTP/NetBoot server
`c(8)`	Standard C language compiler standard C language compiler
`chkpasswd(8)`	Verifies user password against various systems
`configd(8)`	System Configuration Daemon

coreaudiod(8)	Core Audio daemon
dirhelper(8)	Helper for special directory creation
diskarbitrationd(8)	Disk arbitration daemon
diskmanagementd(8)	DiskManagement.framework server
disktool(8)	Disk support tool
diskutil(8)	Modify, verify and repair local disks
distnoted(8)	Distributed notification server
dnsextd(8)	BIND Extension Daemon
dsconfigad(8)	Retrieves/changes configuration for Directory Services Active Directory Plugin
dserr(8)	Prints a description for an error code
dumpemacs(8)	Utility to dump pre-loaded emacs with compiled ELisp auto-loads
dynamic_pager(8)	External storage manager for dynamic pager
fibreconfig(8)	Tool for configuring settings for Fibre Channel controllers and targets
firmwaresyncd(8)	Synchronize files used by the system firmware
fsck_hfs(8)	HFS file system consistency check
hdik(8)	Lightweight in-kernel disk image mounting tool
hfs.util(8)	HFS/HFS+ file system utility
hostinfo(8)	Host information
ifcstart(8)	Rebuilds international data caches
installer(8)	System software and package installer tool
InternetSharing(8)	Simple NAT/router configuration daemon
ioalloccount(8)	Summarize IOKit memory usage
ioclasscount(8)	Displays the instance counts of OSObject-based C++ classes in the kernel
ioreg(8)	Show I/O Kit registry
ioupsd(8)	Daemon to track UPS state
ipconfig(8)	View and control IP configuration state
ipfw(8)	IP firewall and traffic shaper control program
kadmin_util(8)	Kerberos -- Open Directory Single Sign On

`kdcsetup(8)`	Kerberos -- Open Directory Single Sign On
`kerberosautoconfig(8)`	Kerberos -- Open Directory Single Sign On
`kext_logging(8)`	Verbose/logging flags for kernel extensions (kexts) in the kernel and command-line utilities
`kextcache(8)`	Create kext cache files
`kextd(8)`	Kernel extension server
`kextfind(8)`	Find kernel extensions (kexts) based on a variety of criteria and print information
`kextlibs(8)`	Find OSBundleLibraries needed by a kext
`kextload(8)`	Load kernel extensions (kexts) into the kernel
`kextstat(8)`	Display status of loaded kernel extensions (kexts)
`kextunload(8)`	Terminate driver I/O Kit driver instances and unload kernel extensions (kexts)
`kextutil(8)`	Load, diagnose problems with, and generate symbols for kernel extensions (kexts)
`krbservicesetup(8)`	Kerberos -- Open Directory Single Sign On
`kuncd(8)`	The Kernel User Notification Center daemon
`launchd(8)`	System wide and per-user daemon/agent manager
`launchproxy(8)`	Inetd job emulation helper
`locate.bigram(8)`	Sorted list compressor
`locate.code(8)`	Sorted list compressor
`lsbom(8)`	List contents of a bom file
`mDNSResponder(8)`	Multicast and Unicast DNS daemon
`mDNSResponderHelper(8)`	MDNS privilege separation helper
`mkbom(8)`	Create a bill-of-materials file
`mkextunpack(8)`	Extract or list the contents of a multikext (mkext) archive
`mount_afp(8)`	Mount an afp (AppleShare) filesystem
`mount_cddafs(8)`	Mount an Audio CD
`mount_ftp(8)`	Mount a FTP filesystem
`mount_hfs(8)`	Mount an HFS/HFS+ file system
`mount_ntfs(8)`	Mount an NTFS file system
`mount_url(8)`	Mount a remote file system given a URL

`mount_webdav(8)`	Mount a WebDAV filesystem
`msdos.util(8)`	DOS/Windows (FAT) file system utility
`natd(8)`	Network Address Translation daemon
`nbdst(8)`	NetBoot deferred shadow tool
`networksetup(8)`	Configuration tool for network settings in System Preferences
`newfs_hfs(8)`	Construct a new HFS Plus file system
`newfs_hfs(8)`	Construct a new HFS Plus file system
`notifyd(8)`	Notification server
`ntfs.util(8)`	NTFS file system utility
`ntpd-wrapper(8)`	Wrapper for ntpdate/ntpd called by launchd
`path_helper(8)`	Helper for constructing PATH environment variable
`pboard(8)`	Pasteboard server
`pbs(8)`	General helper tool
`pcastagentd(8)`	Captures video, screen, and audio content for Podcast Producer
`pictd(8)`	General helper tool
`PlistBuddy(8)`	Read and write values to plists
`pmap_dump(8)`	Print a list of all registered RPC programs
`pmap_set(8)`	Set the list of registered RPC programs
`pwpolicy(8)`	Gets and sets password policies
`rc(8)`	Command script for boot
`sa1(8)`	Generate a system activity daily data file
`sa2(8)`	Generate a system activity daily data file
`sadc(8)`	System activity data collector
`sandboxd(8)`	Sandbox daemon
`scselect(8)`	Select system configuration location
`scsid(8)`	SCSI subsystem daemon
`scutil(8)`	Manage system configuration parameters
`security_authtrampoline(8)`	

service(8)	Deprecated
service_helper(8)	Helper program for enabling and disabling services
setregion(8)	Set the disc region code for a DVD drive
softwareupdate(8)	Software Update checks for new and updated versions of your software
spindump(8)	Report generation for unresponsive applications helper process for spindump(8)
spindump_ symbolicator(8)	Helper process for spindump(8)
sso_util(8)	Tool for setting up, interrogating and removing Kerberos configurations within the Apple Single Sign On environment
StartupItemContext(8)	Execute a program in StartupItem context
syslogd(8)	Configuration file for syslogd(8) and aslmanager(8) syslogd(8) configuration file Apple System Log server
system_profiler(8)	Reports system hardware and software configuration
systemsetup(8)	Configuration tool for certain machine settings in System Preferences
SystemStarter(8)	Deprecated
taskgated(8)	Task_for_pid access control daemon
tokenadmin(8)	Command-line interface to smartcards and other token-based keychains
ufs.util(8)	UFS file system utility
upsshutdown(8)	UPS emergency low power shutdown script
UserEventAgent(8)	High-level system event handler
vpnd(8)	Mac OS X VPN service daemon
vsdbutil(8)	Manipulates the volume status DB
warmd(8)	Pre-heating daemon
warmd_agent(8)	Pre-heating agent
xgridctl(8)	Xgrid Daemon Control Interface

Index