SAMS Teach Yourself

Shell Programming

in 24 Hours

Sriranga Veeraraghavan

Sriranga Veeraraghavan

SAMS
Teach Yourself

Shell Programming in 24 Hours

SECOND EDITION

SAMS
800 East 96th St., Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Shell Programming in 24 Hours, Second Edition

International Standard Book Number: 0-672-32358-3

Library of Congress Catalog Card Number: 2001096631

Printed in the United States of America

First Printing: April 2002

06 05 13 12 11 10 9 8 7

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearsoned.com

Acquisitions Editor
Katie Purdum

Development Editor
Steve Rowe

Technical Editor
Michael Watson

Managing Editor
Charlotte Clapp

Project Editor
Natalie Harris

Copy Editors
Kezia Endsley
Rhonda Tinch-Mize

Indexer
Kelly Castell

Proofreaders
Linda Seifert
Karen Whitehouse

Interior Design
Gary Adair

Cover Design
Aren Howell

Page Layout
Stacey Richwine-DeRome

Contents at a Glance

Contents

About the Author

SRIRANGA VEERARAGHAVAN is a material scientist by training and a software engineer by trade. He has several years of software development experience in C, Java, Perl, and Bourne Shell and has contributed to several books, including *Solaris 8: Complete Reference*, *UNIX Unleashed* and *Special Edition Using UNIX*. Sriranga graduated from the University of California at Berkeley in 1997 and is presently pursuing further studies. He is currently employed in the Server Appliance group at Sun Microsystems, Inc. Before joining Sun, Sriranga was employed at Cisco Systems, Inc. Among other interests, Sriranga enjoys mountain biking, classical music, and playing Marathon with his brother Srivathsa. Sriranga can be reached via e-mail at `ranga@soda.berkeley.edu`.

Dedication

For my grandmother, who taught me to love the English language.

For my mother, who taught me to love programming languages.

Acknowledgments

Writing a book on shell programming is a daunting task, due to the myriad UNIX versions and shell versions that are available. Thanks to the hard work of my development editor Steve Rowe, my technical editor Michael Watson, and my copy editor Kezia Endsley, I was able to make sure the book covered the material completely and correctly. Their suggestions and comments have helped enormously.

In addition to the technical side of the book, the task of coordinating and managing the publishing process is a difficult one. The assistance of my acquisitions editor, Kathryn Purdum, in handling all of the editorial issues and patiently working with me to keep this book on schedule was invaluable.

Working on a book takes a lot of time and makes it difficult to concentrate on work and family activities. Thanks to the support of my manager, Larry Coryell, my parents, my brother Srivathsa, and my uncle and aunt Srinvasa and Suma, I was able to balance work, family, and authoring.

Thanks to everyone else on the excellent team at Sams who worked on this book. Without their support, this book would not exist.

Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: `opensource@samspublishing.com`

Mail: Mark Taber
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Introduction

In recent years, the UNIX operating system has seen a huge boost in its popularity, especially with the emergence of Linux. For programmers and users of UNIX, this comes as no surprise: UNIX was designed to provide an environment that's powerful yet easy to use.

One of the main strengths of UNIX is that it comes with a large collection of standard programs. These programs perform a wide variety of tasks from listing your files to reading e-mail. Unlike other operating systems, one of the key features of UNIX is that these programs can be combined to perform complicated tasks and solve your problems.

One of the most powerful standard programs available in UNIX is the shell. The *shell* is a program that provides a consistent and easy-to-use environment for executing programs in UNIX. If you have ever used a UNIX system, you have interacted with the shell.

The main responsibility of the shell is to read the commands you type and then ask the UNIX kernel to perform these commands. In addition to this, the shell provides several sophisticated programming constructs that enable you to make decisions, repeatedly execute commands, create functions, and store values in variables.

This book concentrates on the standard UNIX shell called the Bourne shell. When Dennis Ritche and Ken Thompson were developing much of UNIX in the early 1970s, they used a very simple shell. The first real shell, written by Stephen Bourne, appeared in the mid 1970s. The original Bourne shell has changed slightly over the years; some features were added and others were removed, but its syntax and its resulting power have remained the same.

The most attractive feature of the shell is that it enables you to create scripts. *Scripts* are files that contain a list of commands you want to run. Because every script is contained in a file and every file has a name, scripts enable you to combine existing programs to create completely new programs that solve your problems. This book teaches you how to create, execute, modify, and debug shell scripts quickly and easily. After you get used to writing scripts, you will find yourself solving more and more problems with them.

How This Book Is Organized

This book assumes that you have some familiarity with UNIX and know how to log in, create, and edit files, as well as how to work with files and directories to a limited extent. If you haven't used UNIX in a while or you aren't familiar with one of these topics, don't worry; the first part of this book reviews this material thoroughly.

This book is divided into three parts:

- Part I is an introduction to UNIX, the shell, and some common tools.
- Part II covers programming using the shell.
- Part III covers advanced topics in shell programming.

Part I consists of Chapters 1 through 7. The following material is covered in the individual chapters:

- Chapter 1, "Shell Basics," discusses several important concepts related to the shell and describes the different versions of the shell.
- Chapter 2, "Script Basics," describes the process of creating and running a shell script. It also covers the login process and the different modes in which the shell executes.
- Chapters 3, "Working with Files," and 4, "Working with Directories," provide an overview of the commands used when working with files and directories. These chapters show you how to list the contents of a directory, view the contents of a file, and manipulate files and directories.
- Chapter 5, "Input and Output" covers the `echo`, `printf`, and `read` commands along with the < and > input redirection operators. This chapter also covers using file descriptors.
- Chapter 6, "Manipulating File Attributes," introduces the concept of file attributes. It covers the different types of files along with how to modify a file's permissions.
- Chapter 7, "Processes," shows you how to start and stop a process. It also explains the term *process ID* and how you can view them.

By this point, you should have a good foundation in the UNIX basics. This will enable you to start writing shell scripts that solve real problems using the concepts covered in Part II. Part II is the heart of this book, consisting of Chapters 8 through 18. It teaches you about all the tools available when programming in the shell. The following material is covered in these chapters:

- Chapter 8, "Variables," explains the use of variables in shell programming, shows you how to create and delete variables, and explains the concept of environment variables.
- Chapters 9, "Substitution," and 10, "Quoting," cover the topics of substitution and quoting. Chapter 9 shows you the four main types of substitution: filename, variable, command, and arithmetic substitution. Chapter 10 shows you the behavior of the different types of quoting and its affect on substitution.

- Chapters 11, “Flow Control,” and 12, “Loops,” provide complete coverage of flow control and looping. The flow control constructs `if` and `case` are covered along with the loop constructs `for` and `while`.
- Chapter 13, “Parameters,” shows you how to write scripts that use command-line arguments. The special variables and the `getopts` command are covered in detail.
- Chapter 14, “Functions,” discusses shell functions. Functions provide a mapping between a name and a set of commands. Learning to use functions in a shell script is a powerful technique that helps you solve complicated problems.
- Chapters 15, “Text Filters,” 16, “Filtering Text with Regular Expressions,” and 17, “Filtering Text with `awk`,” cover text filtering. These chapters show you how to use a variety of UNIX commands including `grep`, `tr`, `sed`, and `awk`.
- Chapter 18, “Other Tools,” provides an introduction to some tools that are used in shell programming. Some of the commands that are discussed include `type`, `find`, `bc`, and `expr`.

At this point, you will know enough about the shell and the external tools available in UNIX that you can solve most problems. The last part of the book, Part III, is designed to help you solve the most difficult problems encountered in shell programming. Part III spans Chapters 19 through 24 and covers the following material:

- Chapter 19, “Signals,” explains the concept of signals and shows you how to deliver a signal and how to deal with a signal using the `trap` command.
- Chapter 20, “Debugging,” discusses the shell’s built-in debugging tools. It shows you how to use syntax checking and shell tracing to track down bugs and fix them.
- Chapters 21, “Problem Solving with Functions,” and 22, “Problem Solving with Shell Scripts,” cover problem solving. Chapter 21 covers problems that can be solved using functions. Chapter 22 introduces some real-world problems and shows you how to solve them using a shell script.
- Chapter 23, “Scripting for Portability,” covers the topic of portability. In this chapter, you will rewrite several scripts from previous chapters to be portable to different versions of UNIX.
- Chapter 24, “Shell Programming FAQs,” is a question-and-answer chapter. Several common programming questions are presented along with detailed answers and examples.

Each chapter in this book includes complete syntax descriptions for the various commands along with several examples to illustrate the use of commands. The examples are designed to show you how to apply the commands to solve real problems. At the end of

each chapter are a few questions that you can use to check your progress. Some of the questions are short answers, whereas others require you to write scripts.

After Chapter 24, four appendixes are available for your reference:

- Appendix A, "Command Quick Reference," provides a complete command reference.
- Appendix B, "Glossary," contains the terms used in this book.
- Appendix C, "Answers to Questions," contains the answers to all the questions in the book.
- Appendix D, "Shell Function Library," contains a listing of the shell function library discussed in Chapter 21, "Problem Solving with Functions."

About the Examples

As you work through the chapters, try typing in the examples to get a better feeling for how the computer responds and how each command works. After you get an example working, try experimenting with the example by changing commands. Don't be afraid to experiment. Experiments (both successes and failures) teach you important things about UNIX and the shell.

Many of the examples and the answers to the questions are available for downloading from the following URL:

```
http://www.csua.berkeley.edu/~ranga/downloads/tysp2.tar.Z
```

After you have downloaded this file, change to the directory where the file was saved and execute the following commands:

```
$ uncompress tysp2.tar.Z
$ tar -xvf tysp2.tar
```

This creates a directory named `tysp2` that contains the examples from this book.

There is no warranty of any kind on the examples in this book. Much effort has been placed into making the examples as portable as possible. To this end the examples have been tested on the following versions of UNIX:

- Sun Solaris versions 2.5.1 to 8
- Hewlett-Packard HP-UX versions 10.10 to 11.0
- OpenBSD versions 2.6 to 2.9
- Apple MacOS X 10.0 to 10.1.2
- Red Hat Linux versions 4.2, 5.1, 5.2, 6.0, and 6.2
- FreeBSD versions 2.2.6 and 4.0 to 4.3

It is possible that some of the examples might not work on other versions of UNIX. If you encounter a problem or have a suggestion about improvements to the examples or the content of the book, please feel free to contact me at the following e-mail address:

`ranga@soda.berkeley.edu`

I appreciate any suggestions and feedback you have regarding this book.

Conventions Used in This Book

Features in this book include the following:

Notes give you comments and asides about the topic at hand, as well as full explanations of certain concepts.

Tips provide great shortcuts and hints on how to program in shell more effectively.

Cautions warn you against making your life miserable and avoiding the pitfalls in programming.

NEW TERM New terms appear in *italic*. Each of the new terms covered in a chapter is listed at the end of that chapter in the "Terms" section.

At the end of each chapter, you'll find the handy Summary and Quiz sections (with answers found in Appendix C).

In addition, you'll find various typographic conventions throughout this book:

- Commands, variables, directories, and files appear in text in a special `monospaced font`.
- Commands and such that you type appear in **boldface type**.
- Placeholders in syntax descriptions appear in a *`monospaced italic`* typeface. This indicates that you will replace the placeholder with the actual filename, parameter, or other element that it represents.

This page intentionally left blank

HOUR 3

Working with Files

In UNIX there are two basic types of files: ordinary and special. An *ordinary* file contains data, text, or program instructions. Almost all of the files on a UNIX system are ordinary files. This chapter covers operations on ordinary files.

Special files are mainly used to provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Some special files are similar to aliases or shortcuts and enable you to access a single file using different names. Special files are covered in Chapter 6, "Manipulating File Attributes."

Both ordinary and special files are stored in directories. *Directories* are similar to folders in the Mac OS or Windows, and they are covered in detail in Chapter 4, "Working with Directories."

In this chapter, we will examine ordinary files, concentrating on the following topics:

- Listing files
- File contents
- Manipulating files

Listing Files

We'll start by using the `ls` (short for **ls**t) command to list the contents of the current directory:

```
$ ls
```

The output will be similar to the following:

```
Desktop         Icon            Music           Sites
Documents       Library         Pictures        Temporary Items
Downloads       Movies          Public
```

We can tell that several items are in the current directory, but this output does not tell us whether these items are files or directories. To find out which of the items are files and which are directories, we can specify the `-F` *option* to `ls`. An option is an argument that starts with the hyphen or dash character, '`-`'.

The following example illustrates the use of the `-F` option of `ls`:

```
$ ls -F
```

Now the output for the directory is slightly different:

```
Desktop/        Icon            Music/          Sites/
Documents/      Library/        Pictures/       Temporary Items/
Downloads/      Movies/         Public/
```

As you can see, some of the items now have a `/` at the end, incicating each of these items is a directory. The other items, such as `icon`, have no character appended to them. This indicates that they are ordinary files.

When the `-F` option is specified to `ls`, it appends a character indicating the file type of each of the items it lists. The exact character depends on your version of `ls`. For ordinary files, no character is appended. For special files, a character such as `!`, `@`, or `#` is appended to the filename. For more information on the `-F` options, check the UNIX manual page for the `ls` command. You can do this as follows:

```
$ man ls
```

> **Options Are Case Sensitive**
>
> The options that can be specified to a command, such as `ls`, are case sensitive. When specifying an option, you need to make sure that you have specified the correct case for the option. For example, the output from the `-F` option to `ls` is different from the output produced when the `-f` option is specified.

So far, you have seen `ls` list more than one file on a line. Although this is fine for humans reading the output, it is hard to manipulate in a shell script. Shell scripts are geared toward dealing with lines of text, not the individual words on a line. Although external tools, such as the `awk` language covered in Chapter 17, "Filtering Text with `awk`," can be used to deal with multiple words on a line, it is much easier to manipulate the output when each file is listed on a separate line. You can modify the output of `ls` to this format by using the `-1` option. For example,

```
$ ls -1
```

produces the following listing:

```
Desktop
Documents
Downloads
Icon
Library
Movies
Music
Pictures
Public
Sites
Temporary Items
```


Hidden Files

In the examples you have seen thus far, the output has listed only the visible files and directories. You can also use `ls` to list invisible or hidden files and directories. An *invisible* or *hidden* file is one whose first character is a dot or period (`.`). Many programs, including the shell, use such files to store configuration information. Some common examples of invisible files include

- `.profile`, the Bourne shell (`sh`) initialization script
- `.kshrc`, the Korn Shell (`ksh`) initialization script
- `.cshrc`, the C Shell (`csh`) initialization script
- `.rhosts`, the remote shell configuration file

All files that do not start with the `.` character are considered *visible*.

To list invisible files, specify the `-a` option to `ls`:

```
$ ls -a
```

The directory listing now resembles this:

```
.                   .FBCLockFolder      Icon                Public
..                  .ssh                Library             Sites
.CFUserTextEncoding Desktop             Movies              Temporary Items
```

```
.DS_Store           Documents           Music
.FBCIndex           Downloads           Pictures
```

As you can see, this directory contains several invisible files.

Notice that in this output, the file type information is missing. To get the file type information, specify the `-F` and the `-a` options as follows:

```
$ ls -a -F
```

The output changes to the following:

```
./                      .ssh/                   Movies/
../                     Desktop/                Music/
.CFUserTextEncoding     Documents/              Pictures/
.DS_Store               Downloads/              Public/
.FBCIndex               Icon?                   Sites/
.FBCLockFolder/         Library/                Temporary Items/
```

With the file type information, you see that there are two invisible directories (`.` and `..`). These directories are special entries present in all directories. The first one, `.`, represents the current directory, whereas the second one, `..`, represents the parent directory. These concepts are discussed in greater detail in Chapter 4.

Option Grouping

In the previous example, you specified the options to `ls` separately. You could have grouped the options together, as follows:

```
$ ls -aF
$ ls -Fa
```

Both of these commands are equivalent to the following command:

```
$ ls -a -F
```

The order of the options does not matter to `ls`. As an example of option grouping, consider the following equivalent commands:

```
ls -1 -a -F
ls -1aF
ls -a1F
ls -Fa1
```

All permutations of the options `-1`, `-a`, and `-F` produce the same output:

```
./
../
.CFUserTextEncoding
.DS_Store
```

```
.FBCIndex
.FBCLockFolder/
.ssh/
Desktop/
Documents/
Downloads/
Icon?
Library/
Movies/
Music/
Pictures/
Public/
Sites/
Temporary Items/
```


File Contents

In the last section we looked at listing files and directories with the `ls` command. In this section we will look at the `cat` and `wc` commands. The `cat` command lets you view the contents of a file. The `wc` command gives you information about the number of words and lines in a file.

cat

To view the contents of a file, we can use the `cat` (short for con**cat**enate) command as follows:

```
cat [opts] file1 ... fileN
```

Here `opts` are one or more of the options understood by `cat`, and `file1...fileN` are the names of the files whose contents should be printed. The options, `opts`, are optional and can be omitted. Two commonly used options are discussed later in this section.

The following example illustrates the use of `cat`:

```
$ cat fruits
```

This command prints the contents of a file called `fruits`:

```
Fruit           Price/lbs       Quantity
Banana          $0.89           100
Peach           $0.79           65
Kiwi            $1.50           22
Pineapple       $1.29           35
Apple           $0.99           78
```

If more than one file is specified, the output includes the contents of both files concatenated together. For example, the following command outputs the contents of the files `fruits` and `users`:

```
$ cat fruits users
Fruit           Price/lbs       Quantity
Banana          $0.89           100
Peach           $0.79           65
Kiwi            $1.50           22
Pineapple       $1.29           35
Apple           $0.99           78

ranga
vathsa
amma
```

Numbering Lines

The `-n` option of `cat` will number each line of output. It can be used as follows:

```
$ cat -n fruits
```

This produces the output

```
     1  Fruit           Price/lbs       Quantity
     2  Banana          $0.89           100
     3  Peach           $0.79           65
     4  Kiwi            $1.50           22
     5  Pineapple       $1.29           35
     6  Apple           $0.99           78
     7
```

From this output, you can see that the last line in this file is blank. We can ask `cat` to skip numbering blank lines using the `-b` option as follows:

```
$ cat -b fruits
```

Now the output resembles the following:

```
     1  Fruit           Price/lbs       Quantity
     2  Banana          $0.89           100
     3  Peach           $0.79           65
     4  Kiwi            $1.50           22
     5  Pineapple       $1.29           35
     6  Apple           $0.99           78
```

The blank line is still presented in the output, but it is not numbered. If the blank line occurs in the middle of a file, it is printed but not numbered:

```
$ cat -b hosts
     1  127.0.0.1       localhost          loopback

     2  128.32.43.52    soda.berkeley.edu  soda
```

If multiple files are specified, the contents of the files are concatenated in the output, but line numbering is restarted at 1 for each file. As an illustration, the following command,

```
$ cat -b fruits users
```

produces the output

```
     1  Fruit           Price/lbs       Quantity
     2  Banana          $0.89           100
     3  Peach           $0.79           65
     4  Kiwi            $1.50           22
     5  Pineapple       $1.29           35
     6  Apple           $0.99           78

     1  ranga
     2  vathsa
     3  amma
```


wc

Now let's look at getting some information about the contents of a file. Using the `wc` command (short for **w**ord **c**ount), we can get a count of the total number of lines, words, and characters contained in a file. The basic syntax of this command is:

```
wc [opts] files
```

Here `opts` are one or more of the options given in Table 3.1, and `files` are the files you want examined. The options, *`opts`*, are optional and can be omitted.

TABLE 3.1 `wc` Options

Option	*Description*
`-l`	Count of the number of lines.
`-w`	Count of the number of words.
`-m`	Count of the number of characters. This option is available on Mac OS X, OpenBSD, Solaris, and HP-UX. This option is not available on FreeBSD and Linux systems.
`-c`	Count of the number of characters. This option is the Linux and FreeBSD equivalents of the `-m` option.

When no options are specified, the default behavior of `wc` is to print out a summary of the number of lines, words, and characters contained in a file. For example, the command

```
$ wc fruits
```

produces the following output:

```
       8       18     219 fruits
```

The first number, in this case `8`, is the number of lines in the file. The second number, in this case `18`, is the number of words in the file. The third number, in this case `219`, is the number of characters in the file. At the end of the line, the filename is listed. When multiple files are specified, the filename helps to identify the information associated with a particular file.

If more than one file is specified, `wc` gives the counts for each file along with a total. For example, the command

```
$ wc fruits users
```

produces output similar to the following:

```
       8      18     219 fruits
       3       3      18 users
      11      21     237 total
```

```
The output on your system might be slightly different.
```

Counting Lines

To count the number of lines, the `-l` (as in lines) option can be used. For example, the command

```
$ wc -l fruits
```

produces the output

```
       8 fruits
```

The first number, in this case `8`, is the number of lines in the file. The name of the file is listed at the end of the line.

When multiple files are specified, the number of lines in each file is listed along with the total number of lines in all of the specified files. As an example, the command

```
$ wc -l fruits users
```

produces the output

```
       8 fruits
       3 users
      11 total
```

Counting Words

To count the number of words in a file, the `-w` (as in words) option can be used. For example, the command

```
$ wc -w fruits
```

produces the output

```
      18 hosts
```

The first number, in this case `18`, is the number of words in the file. The name of the file is listed at the end of the line.

When multiple files are specified, the number of words in each file is listed along with the total number of words in all of the specified files. As an example, the command

```
$ wc -w fruits users
```

produces the output

```
      18 fruits
       3 users
      21 total
```


Counting Characters

To count the number of characters, we need to use either the `-m` or the `-c` option. The `-m` option is available on Mac OS X, OpenBSD, Solaris, and HP-UX. On FreeBSD and Linux systems, the `-c` option should be used instead.

For example, on Solaris the command

```
$ wc -m fruits
```

produces the output

```
       219 fruits
```

The same output is produced on Linux and FreeBSD systems using the command

```
$ wc -c fruits
```

The first number, in this case `219`, is the number of characters in the file. The name of the file is listed at the end of the line.

When multiple files are specified, the number of characters in each file is listed along with the total number of characters in all the specified files. As an example, the command

```
$ wc -m fruits users
```

produces the output

```
219 hosts
 18 users
237 total
```

Combining Options

The options to `wc` can be grouped together and specified in any order. For example, to obtain a count of the number of lines and words in the file `fruits`, we can use any of the following commands:

```
$ wc -w -l fruits
$ wc -l -w fruits
$ wc -wl fruits
$ wc -lw fruits
```

The output from each of these commands is identical:

```
8      18 fruits
```

The output lists the number of words in the files, followed by the number of lines in the file. The filename is specified at the end of the line. When multiple files are specified, the information for each file is listed along with the appropriate total values.

Manipulating Files

In the preceding sections, you looked at listing files and viewing their content. In this section, you will look at copying, renaming, and removing files using the `cp`, `mv`, and `rm` commands.

Copying Files (cp)

The `cp` command (short for copy) is used to make a copy of a file. The basic syntax of the command is

```
cp src dest
```

Here `src` is the name of the file to be copied (the source) and `dest` is the name of the copy (the destination). For example, the following command creates a copy of the file `fruits` in a file named `fruits.sav`:

```
$ cp fruits fruits.sav
```

If *`dest`* is the name of a directory, a copy with the same name as *`src`* is created in *`dest`*. For example, the command

```
$ cp fruits Documents/
```

creates a copy of the file `fruits` in the directory `Documents`.

It is also possible to specify multiple source files to `cp`, provided that the destination, `dest`, is a directory. The syntax for copying multiple files is

```
$ cp src1 ... srcN dest
```

Here `src1 ... srcN` are the source files and `dest` is the destination directory. As an example, the following command

```
$ cp fruits users Documents/
```

creates a copy the files `fruits` and `users` in the directory `Documents`.

Interactive Mode

The default behavior of `cp` is to automatically overwrite the destination file if it exists. This behavior can lead to problems. The `-i` option (short for interactive) can be used to prevent such problems. In interactive mode, `cp` prompts for confirmation before overwriting any files.

Assuming that the file `fruits.sav` exists, the following command

```
$ cp -i fruits fruits.sav
```

results in a prompt similar to the following:

```
overwrite fruits.sav? (y/n)
```

If `y` (yes) is chosen, the file `fruits.sav` is overwritten; otherwise the file is untouched. The actual prompt varies among the different versions of UNIX.

Common Errors

When an error is encountered, `cp` generates a message. Some common error conditions follow:

- The source, *`src`*, is a directory.
- The source, *`src`*, does not exist.
- The destination, *`dest`*, is not a directory when multiple sources, *`src1 ... srcN`*, are specified.
- A non-existent destination, *`dest`*, is specified along with multiple sources, *`src1 ... srcN`*.
- One of the sources in *`src1 ... srcN`* is not a file.

The first error type is illustrated by the following command:

```
$ cp Downloads/ fruits
```

Because *`src`* (`Downloads` in this case) is a directory, an error message similar to the following is generated:

```
cp: Downloads: is a directory
```

In this example, `dest` was the file `fruits`; the same error would have been generated if `dest` was a directory.

The second error type is illustrated by the following command:

```
$ cp fritus fruits.sav
cp: cannot access fritus: No such file or directory
```

Here the filename `fruits` has been misspelled `fritus`, resulting in an error. In this example `dest` was the file `fruits.sav`; the same error would have been generated if *`dest`* was a directory.

The third error type is illustrated by the following command:

```
$ cp fruits users fruits.sav
usage: cp [-R [-H | -L | -P]] [-f | -i] [-p] src target
       cp [-R [-H | -L | -P]] [-f | -i] [-p] src1 ... srcN directory
```

Because *`dest`*, in this case `fruits.sav`, is not a directory, a usage statement that highlights the proper syntax for a `cp` command is presented. The output might be different on your system because some versions of `cp` do not display the usage information.

If the file `fruits.sav` does not exist, the error message is

```
cp: fruits.sav: No such file or directory
```

This illustrates the fourth error type.

The fifth error type is illustrated by the following command:

```
$ cp fruits Downloads/ users Documents/
cp: Downloads is a directory (not copied).
```

Although `cp` reports an error for the directory `Downloads`, the other files are correctly copied to the directory `Documents`.

Renaming Files (`mv`)

The `mv` command (short for move) can be used to change the name of a file. The basic syntax is

```
mv src dest
```

Here `src` is the original name of the file and `dest` is the new name of the file. For example, the command

```
$ mv fruits fruits.sav
```

changes the name of the file `fruits` to `fruits.sav`. There is no output from `mv` if the name change is successful.

If `src` does not exist, an error will be generated. For example,

```
$ mv cp fritus fruits.sav
mv: fritus: cannot access: No such file or directory
```

Similar to `cp`, `mv` does not report an error if `dest` already exists. The old file is automatically overwritten. This problem can be avoided by specifying the `-i` option (short for interactive). In interactive mode, `mv` prompts for confirmation before overwriting any files.

Assuming that the file `fruits.sav` already exists, the command

```
$ mv -i fruits fruits.sav
```

results in a confirmation prompt similar to the following:

```
overwrite fruits.sav?
```

If `y` (yes) is chosen, the file `fruits.sav` is overwritten; otherwise the file is untouched. The actual prompt varies among the different versions of UNIX.

Removing Files (`rm`)

The `rm` command (short for remove) can be used to remove or delete files. Its syntax is

```
rm file1 ... fileN
```

Here `file1 ... fileN` is a list of one or more files to remove. For example, the command

```
$ rm fruits users
```

removes the files `fruits` and `users`.

Because there is no way to recover files that have been removed using `rm`, you should make sure that you specify only those files you really want removed. One way to ensure this is by specifying the `-i` option (short for interactive). In interactive mode, `rm` prompts before removing every file. For example, the command

```
$ rm -i fruits users
```

produces confirmation prompts similar to the following:

```
fruits: ? (n/y) y
users: ? (n/y) n
```

In this case, you answered `y` (yes) to removing `fruits` and `n` (no) to removing `users`. Thus, the file `fruits` was removed, but the file `users` was untouched.

Common Errors

The two most common errors when using `rm` are

- One of the specified files does not exist.
- One of the specified files is a directory.

The first error type is illustrated by the following command:

```
$ rm users fritus hosts
rm: fritus non-existent
```

Because the file `fruits` is misspelled as `fritus`, it cannot be removed. The other two files are removed correctly.

The second error type is illustrated by the following command:

```
$ rm fruits users Documents/
rm: Documents directory
```

The `rm` command is unable to remove directories and presents an error message stating this fact. It removes the two other files correctly.

Summary

In this chapter, the following topics were discussed:

- Listing files using `ls`
- Viewing the content of a file using `cat`
- Counting the words, lines, and characters in a file using `wc`
- Copying files using `cp`
- Renaming files using `mv`
- Removing files using `rm`

Knowing how to perform each of these basic tasks is essential to becoming a good shell programmer. In the chapters ahead, you will use these basics to create scripts for solving real-world problems.

Questions

1. What are invisible files? How can they be listed with `ls`?
2. Is there any difference in the output of the following commands?
   a. `$ ls -a1`
   b. `$ ls -1 -a`
   c. `$ ls -1a`
3. Which options should be specified to `wc` to count just the number of lines and characters in a file?
4. Given that `hw1`, `hw2`, `ch1`, and `ch2` are files and `book` and `homework` are directories, which of the following commands generates an error message?
   a. `$ cp hw1 ch2 homework`
   b. `$ cp hw1 homework hw2 book`
   c. `$ rm hw1 homework ch1`
   d. `$ rm hw2 ch2`

Terms

Directories Directories are used to hold ordinary and special files. Directories are similar to folders in Mac OS or Windows.

Invisible Files An invisible file is one whose first character is a dot or period (`.`). Many programs (including the shell) use such files to store configuration information. Invisible files are also referred to as hidden files.

Option An option is an argument that starts with the hyphen or dash character, '`-`'.

Ordinary File An ordinary file is a file that contains data, text, or program instructions. Almost all the files on a UNIX system are ordinary files.

Special Files Special files are mainly used to provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Some special files are similar to aliases or shortcuts and enable you to access a single file using different names.

This page intentionally left blank

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P-Q

R

S

T

U

V

W-Y

Z