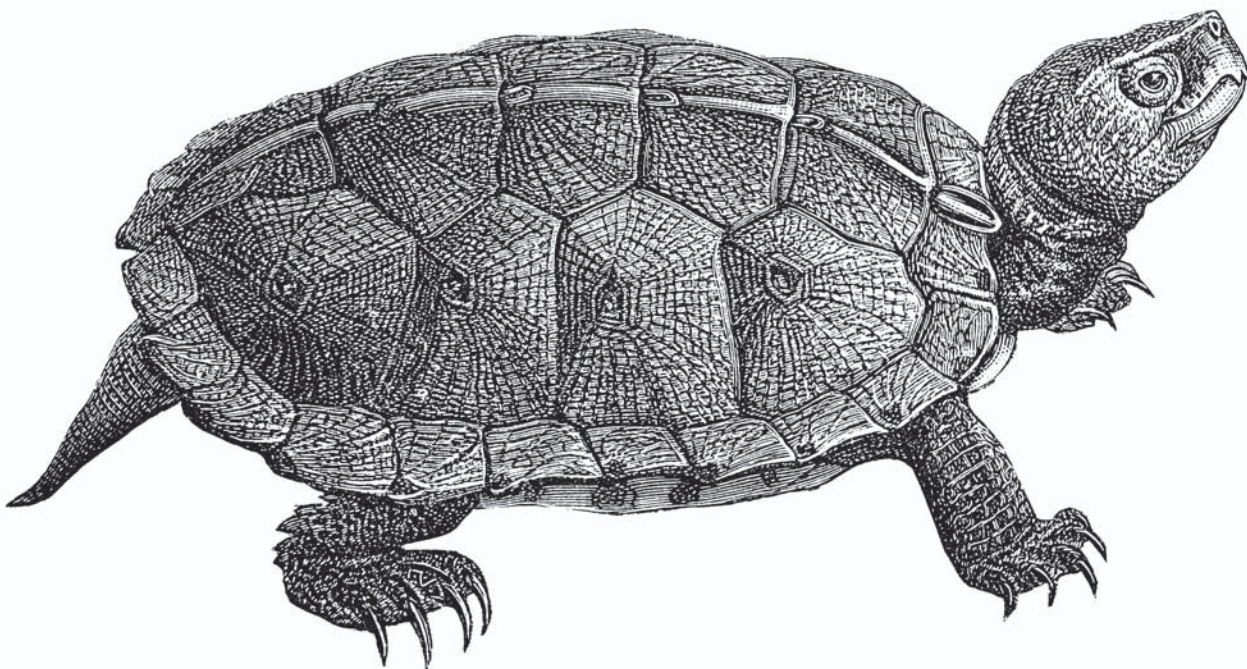


Solutions and Examples for bash Users

bash Cookbook™



O'REILLY®

*Carl Albing, JP Vossen &
Cameron Newham*

bash CookbookTM

Carl Albing, JP Vossen, and Cameron Newham

O'REILLY[®]

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

bash Cookbook™

by Carl Albing, JP Vossen, and Cameron Newham

Copyright © 2007 O'Reilly Media, Inc. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Mike Loukides

Production Editor: Laurel R.T. Ruma

Copyeditor: Derek Di Matteo

Production Services: Tolman Creek Design

Cover Designer: Karen Montgomery

Interior Designer: David Futato

Illustrators: Robert Romano and Jessamyn Read

Printing History:

May 2007: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *bash Cookbook*, the image of a wood turtle, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.



This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN 10: 0-596-52678-4

ISBN 13: 978-0-596-52678-8

[M]

Table of Contents

Preface	xiii
1. Beginning bash	1
1.1 Decoding the Prompt	4
1.2 Showing Where You Are	5
1.3 Finding and Running Commands	6
1.4 Getting Information About Files	8
1.5 Showing All Hidden (dot) Files in the Current Directory	10
1.6 Using Shell Quoting	12
1.7 Using or Replacing Built-ins and External Commands	13
1.8 Determining If You Are Running Interactively	15
1.9 Setting bash As Your Default Shell	16
1.10 Getting bash for Linux	17
1.11 Getting bash for xBSD	20
1.12 Getting bash for Mac OS X	21
1.13 Getting bash for Unix	22
1.14 Getting bash for Windows	23
1.15 Getting bash Without Getting bash	24
1.16 Learning More About bash Documentation	25
2. Standard Output	28
2.1 Writing Output to the Terminal/Window	29
2.2 Writing Output but Preserving Spacing	30
2.3 Writing Output with More Formatting Control	31
2.4 Writing Output Without the Newline	32
2.5 Saving Output from a Command	33
2.6 Saving Output to Other Files	34

2.7	Saving Output from the ls Command	35
2.8	Sending Both Output and Error Messages to Different Files	37
2.9	Sending Both Output and Error Messages to the Same File	37
2.10	Appending Rather Than Clobbering Output	39
2.11	Using Just the Beginning or End of a File	39
2.12	Skipping a Header in a File	40
2.13	Throwing Output Away	41
2.14	Saving or Grouping Output from Several Commands	41
2.15	Connecting Two Programs by Using Output As Input	43
2.16	Saving a Copy of Output Even While Using It As Input	44
2.17	Connecting Two Programs by Using Output As Arguments	46
2.18	Using Multiple Redirects on One Line	47
2.19	Saving Output When Redirect Doesn't Seem to Work	48
2.20	Swapping STDERR and STDOUT	50
2.21	Keeping Files Safe from Accidental Overwriting	52
2.22	Clobbering a File on Purpose	53
3.	Standard Input	55
3.1	Getting Input from a File	55
3.2	Keeping Your Data with Your Script	56
3.3	Preventing Weird Behavior in a Here-Document	57
3.4	Indenting Here-Documents	59
3.5	Getting User Input	60
3.6	Getting Yes or No Input	61
3.7	Selecting from a List of Options	64
3.8	Prompting for a Password	65
4.	Executing Commands	67
4.1	Running Any Executable	67
4.2	Telling If a Command Succeeded or Not	69
4.3	Running Several Commands in Sequence	71
4.4	Running Several Commands All at Once	72
4.5	Deciding Whether a Command Succeeds	74
4.6	Using Fewer if Statements	75
4.7	Running Long Jobs Unattended	76
4.8	Displaying Error Messages When Failures Occur	77
4.9	Running Commands from a Variable	78
4.10	Running All Scripts in a Directory	79

5. Basic Scripting: Shell Variables	80
5.1 Documenting Your Script	82
5.2 Embedding Documentation in Shell Scripts	83
5.3 Promoting Script Readability	85
5.4 Separating Variable Names from Surrounding Text	86
5.5 Exporting Variables	87
5.6 Seeing All Variable Values	89
5.7 Using Parameters in a Shell Script	90
5.8 Looping Over Arguments Passed to a Script	91
5.9 Handling Parameters with Blanks	92
5.10 Handling Lists of Parameters with Blanks	94
5.11 Counting Arguments	96
5.12 Consuming Arguments	98
5.13 Getting Default Values	99
5.14 Setting Default Values	100
5.15 Using null As a Valid Default Value	101
5.16 Using More Than Just a Constant String for Default	102
5.17 Giving an Error Message for Unset Parameters	103
5.18 Changing Pieces of a String	105
5.19 Using Array Variables	106
6. Shell Logic and Arithmetic	108
6.1 Doing Arithmetic in Your Shell Script	108
6.2 Branching on Conditions	111
6.3 Testing for File Characteristics	114
6.4 Testing for More Than One Thing	117
6.5 Testing for String Characteristics	118
6.6 Testing for Equal	119
6.7 Testing with Pattern Matches	121
6.8 Testing with Regular Expressions	122
6.9 Changing Behavior with Redirections	125
6.10 Looping for a While	126
6.11 Looping with a read	128
6.12 Looping with a Count	130
6.13 Looping with Floating-Point Values	131
6.14 Branching Many Ways	132
6.15 Parsing Command-Line Arguments	134
6.16 Creating Simple Menus	137

6.17	Changing the Prompt on Simple Menus	138
6.18	Creating a Simple RPN Calculator	139
6.19	Creating a Command-Line Calculator	142
7.	Intermediate Shell Tools I	144
7.1	Sifting Through Files for a String	145
7.2	Getting Just the Filename from a Search	147
7.3	Getting a Simple True/False from a Search	148
7.4	Searching for Text While Ignoring Case	149
7.5	Doing a Search in a Pipeline	149
7.6	Paring Down What the Search Finds	151
7.7	Searching with More Complex Patterns	152
7.8	Searching for an SSN	153
7.9	Grepping Compressed Files	154
7.10	Keeping Some Output, Discarding the Rest	155
7.11	Keeping Only a Portion of a Line of Output	156
7.12	Reversing the Words on Each Line	157
7.13	Summing a List of Numbers	158
7.14	Counting String Values	159
7.15	Showing Data As a Quick and Easy Histogram	161
7.16	Showing a Paragraph of Text After a Found Phrase	163
8.	Intermediate Shell Tools II	165
8.1	Sorting Your Output	165
8.2	Sorting Numbers	166
8.3	Sorting IP Addresses	167
8.4	Cutting Out Parts of Your Output	170
8.5	Removing Duplicate Lines	171
8.6	Compressing Files	172
8.7	Uncompressing Files	174
8.8	Checking a tar Archive for Unique Directories	175
8.9	Translating Characters	176
8.10	Converting Uppercase to Lowercase	177
8.11	Converting DOS Files to Linux Format	178
8.12	Removing Smart Quotes	179
8.13	Counting Lines, Words, or Characters in a File	180
8.14	Rewrapping Paragraphs	181
8.15	Doing More with less	181

9. Finding Files: find, locate, slocate	184
9.1 Finding All Your MP3 Files	184
9.2 Handling Filenames Containing Odd Characters	186
9.3 Speeding Up Operations on Found Files	187
9.4 Finding Files Across Symbolic Links	188
9.5 Finding Files Irrespective of Case	188
9.6 Finding Files by Date	189
9.7 Finding Files by Type	191
9.8 Finding Files by Size	192
9.9 Finding Files by Content	192
9.10 Finding Existing Files and Content Fast	194
9.11 Finding a File Using a List of Possible Locations	195
10. Additional Features for Scripting	199
10.1 “Daemon-izing” Your Script	199
10.2 Reusing Code with Includes and Sourcing	200
10.3 Using Configuration Files in a Script	202
10.4 Defining Functions	203
10.5 Using Functions: Parameters and Return Values	205
10.6 Trapping Interrupts	207
10.7 Redefining Commands with alias	211
10.8 Avoiding Aliases, Functions	213
11. Working with Dates and Times	216
11.1 Formatting Dates for Display	217
11.2 Supplying a Default Date	218
11.3 Automating Date Ranges	220
11.4 Converting Dates and Times to Epoch Seconds	222
11.5 Converting Epoch Seconds to Dates and Times	223
11.6 Getting Yesterday or Tomorrow with Perl	224
11.7 Figuring Out Date and Time Arithmetic	225
11.8 Handling Time Zones, Daylight Saving Time, and Leap Years	227
11.9 Using date and cron to Run a Script on the Nth Day	228
12. End-User Tasks As Shell Scripts	230
12.1 Starting Simple by Printing Dashes	230
12.2 Viewing Photos in an Album	232
12.3 Loading Your MP3 Player	237
12.4 Burning a CD	242
12.5 Comparing Two Documents	244

13. Parsing and Similar Tasks	248
13.1 Parsing Arguments for Your Shell Script	248
13.2 Parsing Arguments with Your Own Error Messages	251
13.3 Parsing Some HTML	253
13.4 Parsing Output into an Array	255
13.5 Parsing Output with a Function Call	256
13.6 Parsing Text with a read Statement	257
13.7 Parsing with read into an Array	258
13.8 Getting Your Plurals Right	259
13.9 Taking It One Character at a Time	260
13.10 Cleaning Up an SVN Source Tree	261
13.11 Setting Up a Database with MySQL	262
13.12 Isolating Specific Fields in Data	264
13.13 Updating Specific Fields in Data Files	266
13.14 Trimming Whitespace	268
13.15 Compressing Whitespace	271
13.16 Processing Fixed-Length Records	273
13.17 Processing Files with No Line Breaks	275
13.18 Converting a Data File to CSV	277
13.19 Parsing a CSV Data File	278
14. Writing Secure Shell Scripts	280
14.1 Avoiding Common Security Problems	282
14.2 Avoiding Interpreter Spoofing	283
14.3 Setting a Secure \$PATH	283
14.4 Clearing All Aliases	285
14.5 Clearing the Command Hash	286
14.6 Preventing Core Dumps	287
14.7 Setting a Secure \$IFS	287
14.8 Setting a Secure umask	288
14.9 Finding World-Writable Directories in Your \$PATH	289
14.10 Adding the Current Directory to the \$PATH	291
14.11 Using Secure Temporary Files	292
14.12 Validating Input	296
14.13 Setting Permissions	298
14.14 Leaking Passwords into the Process List	299
14.15 Writing setuid or setgid Scripts	300
14.16 Restricting Guest Users	301
14.17 Using chroot Jails	303

14.18	Running As a Non-root User	305
14.19	Using sudo More Securely	305
14.20	Using Passwords in Scripts	307
14.21	Using SSH Without a Password	308
14.22	Restricting SSH Commands	316
14.23	Disconnecting Inactive Sessions	318
15.	Advanced Scripting	320
15.1	Finding bash Portably for #!	321
15.2	Setting a POSIX \$PATH	322
15.3	Developing Portable Shell Scripts	324
15.4	Testing Scripts in VMware	326
15.5	Using for Loops Portably	327
15.6	Using echo Portably	329
15.7	Splitting Output Only When Necessary	332
15.8	Viewing Output in Hex	333
15.9	Using bash Net-Redirection	334
15.10	Finding My IP Address	335
15.11	Getting Input from Another Machine	340
15.12	Redirecting Output for the Life of a Script	342
15.13	Working Around “argument list too long” Errors	343
15.14	Logging to syslog from Your Script	345
15.15	Sending Email from Your Script	345
15.16	Automating a Process Using Phases	348
16.	Configuring and Customizing bash	352
16.1	bash Startup Options	353
16.2	Customizing Your Prompt	353
16.3	Change Your \$PATH Permanently	361
16.4	Change Your \$PATH Temporarily	362
16.5	Setting Your \$CDPATH	367
16.6	Shortening or Changing Command Names	369
16.7	Adjusting Shell Behavior and Environment	371
16.8	Adjusting readline Behavior Using .inputrc	371
16.9	Keeping a Private Stash of Utilities by Adding ~/bin 373	
16.10	Using Secondary Prompts: \$PS2, \$PS3, \$PS4	374
16.11	Synchronizing Shell History Between Sessions	376
16.12	Setting Shell History Options	377

16.13	Creating a Better cd Command	380
16.14	Creating and Changing into a New Directory in One Step	381
16.15	Getting to the Bottom of Things	383
16.16	Adding New Features to bash Using Loadable Built-ins	384
16.17	Improving Programmable Completion	389
16.18	Using Initialization Files Correctly	394
16.19	Creating Self-Contained, Portable RC Files	398
16.20	Getting Started with a Custom Configuration	400
17.	Housekeeping and Administrative Tasks	411
17.1	Renaming Many Files	411
17.2	Using GNU Texinfo and Info on Linux	413
17.3	Unzipping Many ZIP Files	414
17.4	Recovering Disconnected Sessions Using screen	415
17.5	Sharing a Single bash Session	417
17.6	Logging an Entire Session or Batch Job	418
17.7	Clearing the Screen When You Log Out	420
17.8	Capturing File Metadata for Recovery	421
17.9	Creating an Index of Many Files	422
17.10	Using diff and patch	422
17.11	Counting Differences in Files	426
17.12	Removing or Renaming Files Named with Special Characters	428
17.13	Prepending Data to a File	429
17.14	Editing a File in Place	432
17.15	Using sudo on a Group of Commands	434
17.16	Finding Lines in One File But Not in the Other	436
17.17	Keeping the Most Recent N Objects	439
17.18	Grepping ps Output Without Also Getting the grep Process Itself	442
17.19	Finding Out Whether a Process Is Running	443
17.20	Adding a Prefix or Suffix to Output	444
17.21	Numbering Lines	446
17.22	Writing Sequences	448
17.23	Emulating the DOS Pause Command	450
17.24	Commifying Numbers	450
18.	Working Faster by Typing Less	453
18.1	Moving Quickly Among Arbitrary Directories	453
18.2	Repeating the Last Command	455
18.3	Running Almost the Same Command	456

18.4	Substituting Across Word Boundaries	457
18.5	Reusing Arguments	458
18.6	Finishing Names for You	459
18.7	Playing It Safe	460
19.	Tips and Traps: Common Goofs for Novices	462
19.1	Forgetting to Set Execute Permissions	462
19.2	Fixing “No such file or directory” Errors	463
19.3	Forgetting That the Current Directory Is Not in the \$PATH	465
19.4	Naming Your Script Test	466
19.5	Expecting to Change Exported Variables	467
19.6	Forgetting Quotes Leads to “command not found” on Assignments	468
19.7	Forgetting That Pattern Matching Alphabetizes	470
19.8	Forgetting That Pipelines Make Subshells	470
19.9	Making Your Terminal Sane Again	473
19.10	Deleting Files Using an Empty Variable	474
19.11	Seeing Odd Behavior from printf	474
19.12	Testing bash Script Syntax	476
19.13	Debugging Scripts	477
19.14	Avoiding “command not found” When Using Functions	479
19.15	Confusing Shell Wildcards and Regular Expressions	480
A.	Reference Lists	482
	bash Invocation	482
	Prompt String Customizations	483
	ANSI Color Escape Sequences	484
	Built-in Commands and Reserved Words	485
	Built-in Shell Variables	487
	set Options	491
	shopt Options	492
	Adjusting Shell Behavior Using set, shopt, and Environment Variables	494
	Test Operators	505
	I/O Redirection	506
	echo Options and Escape Sequences	508
	printf	509
	Date and Time String Formatting with strftime	513
	Pattern-Matching Characters	514
	extglob Extended Pattern-Matching Operators	515
	tr Escape Sequences	515

Readline Init File Syntax	516
emacs Mode Commands	518
vi Control Mode Commands	520
Table of ASCII Values	522
B. Examples Included with bash	524
Startup-Files Directory Examples	524
C. Command-Line Processing	532
Command-Line Processing Steps	532
D. Revision Control	538
CVS	539
Subversion	545
RCS	550
Other	557
E. Building bash from Source	559
Obtaining bash	559
Unpacking the Archive	559
What's in the Archive	560
Who Do I Turn To?	564
Index	567

Preface

Every modern operating system has at least one shell and some have many. Some shells are command-line oriented, such as the shell discussed in this book. Others are graphical, like Windows Explorer or the Macintosh Finder. Some users will interact with the shell only long enough to launch their favorite application, and then never emerge from that until they log off. But most users spend a significant amount of time using the shell. The more you know about your shell, the faster and more productive you can be.

Whether you are a system administrator, a programmer, or an end user, there are certainly occasions where a simple (or perhaps not so simple) shell script can save you time and effort, or facilitate consistency and repeatability for some important task. Even using an alias to change or shorten the name of a command you use often can have a significant effect. We'll cover this and much more.

As with any general programming language, there is more than one way to do a given task. In some cases, there is only one *best* way, but in most cases there are at least two or three equally effective and efficient ways to write a solution. Which way you choose depends on your personal style, creativity, and familiarity with different commands and techniques. This is as true for us as authors as it is for you as the reader. In most cases we will choose a single method and implement it. In a few cases we may choose a particular method and explain why we think it's the best. We may also occasionally show more than one equivalent solution so you can choose the one that best fits your needs and environment.

There is also sometimes a choice between a clever way to write some code, and a readable way. We will choose the readable way every time because experience has taught us that no matter how transparent you think your clever code is now, six or eighteen months and 10 projects from now, you will be scratching your head asking yourself what you were thinking. Trust us, write clear code, and document it—you'll thank yourself (and us) later.

Who Should Read This Book

This book is for anyone who uses a Unix or Linux system, as well as system administrators who may use several systems on any given day. With it, you will be able to create scripts that allow you to accomplish more, in less time, more easily, consistently, and repeatably than ever before.

Anyone? Yes. New users will appreciate the sections on automating repetitive tasks, making simple substitutions, and customizing their environment to be more friendly and perhaps behave in more familiar ways. Power users and administrators will find new and different solutions to common tasks and challenges. Advanced users will have a collection of techniques they can use at a moment's notice to put out the latest fire, without having to remember every little detail of syntax.

Ideal readers include:

- New Unix or Linux users who don't know much about the shell, but want to do more than point and click
- Experienced Unix or Linux users and system administrators looking for quick answers to shell scripting questions
- Programmers who work in a Unix or Linux (or even Windows) environment and want to be more productive
- New Unix or Linux sysadmins, or those coming from a Windows environment who need to come up to speed quickly
- Experienced Windows users and sysadmins who want a more powerful scripting environment

This book will only briefly cover basic and intermediate shell scripting—see *Learning the bash Shell* by Cameron Newham (O'Reilly) and *Classic Shell Scripting* by Nelson H.F. Beebe and Arnold Robbins (O'Reilly) for more in-depth coverage. Instead, our goal is to provide solutions to common problems, with a strong focus on the “how to” rather than the theory. We hope this book will save you time when figuring out solutions or trying to remember syntax. In fact, that's why we wrote this book. It's one we wanted to read through to get ideas, then refer to practical working examples when needed. That way we don't have to remember the subtle differences between the shell, Perl, C, and so forth.

This book assumes you have access to a Unix or Linux system (or see Recipe 1.15, “Getting bash Without Getting bash” and Recipe 15.4, “Testing Scripts in VMware”) and are familiar with logging in, typing basic commands, and using a text editor. You do not have to be root to use the vast majority of the recipes, though there are a few, particularly dealing with installing bash, where root access will be needed.

About This Book

This book covers *bash*, the GNU Bourne Again Shell, which is a member of the Bourne family of shells that includes the original Bourne shell *sh*, the Korn shell *ksh*, and the Public Domain Korn Shell *pdksh*. While these and other shells such as *dash*, and *zsh* are not specifically covered, odds are that most of the scripts will work pretty well with them.

You should be able to read this book cover to cover, and also just pick it up and read anything that catches your eye. But perhaps most importantly, we hope that when you have a question about how to do something or you need a hint, you will be able to easily find the right answer—or something close enough—and save time and effort.

A great part of the Unix philosophy is to build simple tools that do one thing well, then combine them as needed. This combination of tools is often accomplished via a shell script because these commands, called pipelines, can be long or difficult to remember and type. Where appropriate, we'll cover the use of many of these tools in the context of the shell script as the glue that holds the pieces together to achieve the goal.

This book was written using OpenOffice.org Writer running on whatever Linux or Windows machine happened to be handy, and kept in Subversion (see Appendix D). The nature of the Open Document Format facilitated many critical aspects of writing this book, including cross-references and extracting code see Recipe 13.17, “Processing Files with No Line Breaks.”

GNU Software

bash, and many of the tools we discuss in this book, are part of the GNU Project (<http://www.gnu.org/>). GNU (pronounced guh-noo, like canoe) is a recursive acronym for “GNU’s Not Unix” and the project dates back to 1984. Its goal is to develop a free (as in freedom) Unix-like operating system.

Without getting into too much detail, what is commonly referred to as *Linux* is, in fact, a kernel with various supporting software as a core. The GNU tools are wrapped around it and it has a vast array of other software possibly included, depending on your distribution. However, the Linux kernel itself is not GNU software.

The GNU project argues that Linux should in fact be called “GNU/Linux” and they have a good point, so some distributions, notably Debian, do this. Therefore GNU’s goal has arguably been achieved, though the result is not exclusively GNU.

The GNU project has contributed a vast amount of superior software, notably including *bash*, but there are GNU versions of practically every tool we discuss in this book. And while the GNU tools are more rich in terms of features and (usually) friendliness, they are also sometimes a little different. We discuss this in Recipe 15.3,

“Developing Portable Shell Scripts,” though the commercial Unix vendors in the 1980s and 1990s are also largely to blame for these differences.

Enough (several books this size worth) has already been said about all of these aspects of GNU, Unix, and Linux, but we felt that this brief note was appropriate. See <http://www.gnu.org> for much more on the topic.

A Note About Code Examples

When we show an executable piece of shell scripting in this book, we typically show it in an offset area like this:

```
$ ls
a.out  cong.txt  def.conf  file.txt  more.txt  zebra.list
$
```

The first character is often a dollar sign (\$) to indicate that this command has been typed at the *bash* shell prompt. (Remember that you can change the prompt, as in Recipe 16.2, “Customizing Your Prompt,” so your prompt may look very different.) The prompt is printed by the shell; you type the remainder of the line. Similarly, the last line in such an example is often a prompt (the \$ again), to show that the command has ended execution and control has returned to the shell.

The pound or hash sign (#) is a little trickier. In many Unix or Linux files, including *bash* shell scripts, a leading # denotes a comment, and we have used it that way in some of our code examples. But as the trailing symbol in a *bash* command prompt (instead of \$), # means you are logged in as root. We only have one example that is running anything as root, so that shouldn’t be confusing, but it’s important to understand.

When you see an example without the prompt string, we are showing the contents of a shell script. For several large examples we will number the lines of the script, though the numbers are not part of the script.

We may also occasionally show an example as a session log or a series of commands. In some cases, we may cat one or more files so you can see the script and/or data files we’ll be using in the example or in the results of our operation.

```
$ cat data_file
static header line1
static header line2
1 foo
2 bar
3 baz
```

Many of the longer scripts and functions are available to download as well. See the end of this Preface for details. We have chosen to use `#!/usr/bin/env bash` for these examples, where applicable, as that is more portable than the `#!/bin/bash` you will see on Linux or a Mac. See Recipe 15.1, “Finding *bash* Portably for #!” for more details.

Also, you may notice something like the following in some code examples:

```
# cookbook filename: snippet_name
```

That means that the code you are reading is available for download on our site (<http://www.bashcookbook.com>). The download (.tgz or .zip) is documented, but you'll find the code in something like `./chXX/snippet_name`, where `chXX` is the chapter and `snippet_name` is the name of the file.

Useless Use of `cat`

Certain Unix users take a positively giddy delight in pointing out inefficiencies in other people's code. Most of the time this is constructive criticism gently given and gratefully received.

Probably the most common case is the so-called “useless use of `cat` award” bestowed when someone does something like `cat file | grep foo` instead of simply `grep foo file`. In this case, `cat` is unnecessary and incurs some system overhead since it runs in a subshell. Another common case would be `cat file | tr '[A-Z]' '[a-z]'` instead of `tr '[A-Z]' '[a-z]' < file`. Sometimes using `cat` can even cause your script to fail (see Recipe 19.8, “Forgetting That Pipelines Make Subshells”).

But... (you knew that was coming, didn't you?) sometimes unnecessarily using `cat` actually does serve a purpose. It might be a placeholder to demonstrate the fragment of a pipeline, with other commands later replacing it (perhaps even `cat -n`). Or it might be that placing the file near the left side of the code draws the eye to it more clearly than hiding it behind a `<` on the far right side of the page.

While we applaud efficiency and agree it is a goal to strive for, it isn't as critical as it once was. We are *not* advocating carelessness and code-bloat, we're just saying that processors aren't getting any slower any time soon. So if you like `cat`, use it.

A Note About Perl

We made a conscious decision to avoid using Perl in our solutions as much as possible, though there are still a few cases where it makes sense. Perl is already covered elsewhere in far greater depth and breadth than we could ever manage here. And Perl is generally much larger, with significantly more overhead, than our solutions. There is also a fine line between shell scripting and Perl scripting, and this is a book about shell scripting.

Shell scripting is basically glue for sticking Unix programs together, whereas Perl incorporates much of the functionality of the external Unix programs into the language itself. This makes it more efficient and in some ways more portable, at the expense of being different, and making it harder to efficiently run any external programs you still need.

The choice of which tool to use often has more to do with familiarity than with any other reason. The bottom line is always getting the work done; the choice of tools is secondary. We'll show you many of ways to do things using *bash* and related tools. When you need to get your work done, you get to choose what tools you use.

More Resources

- *Perl Cookbook*, Nathan Torkington and Tom Christiansen (O'Reilly)
- *Programming Perl*, Larry Wall et al. (O'Reilly)
- *Perl Best Practices*, Damian Conway (O'Reilly)
- *Mastering Regular Expressions*, Jeffrey E. F. Friedl (O'Reilly)
- *Learning the bash Shell*, Cameron Newham (O'Reilly)
- *Classic Shell Scripting*, Nelson H.F. Beebe and Arnold Robbins (O'Reilly)

Conventions Used in This Book

The following typographical conventions are used in this book:

Plain text

Indicates menu titles, menu options, menu buttons, and keyboard accelerators (such as Alt and Ctrl).

Italic

Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities.

Constant width

Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

Constant width bold

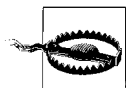
Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values.



This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*bash Cookbook* by Carl Albing, JP Vossen, and Cameron Newham. Copyright 2007 O'Reilly Media, Inc., 978-0-596-52678-8."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

We'd Like to Hear from You

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

<http://www.oreilly.com/catalog/9780596526788>

You can find information about this book, code samples, errata, links, *bash* documentation, and more at the authors' site:

<http://www.bashcookbook.com>

Please drop by for a visit to learn, contribute, or chat. The authors would love to hear from you about what you like and don't like about the book, what *bash* wonders you may have found, or lessons you have learned.

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com