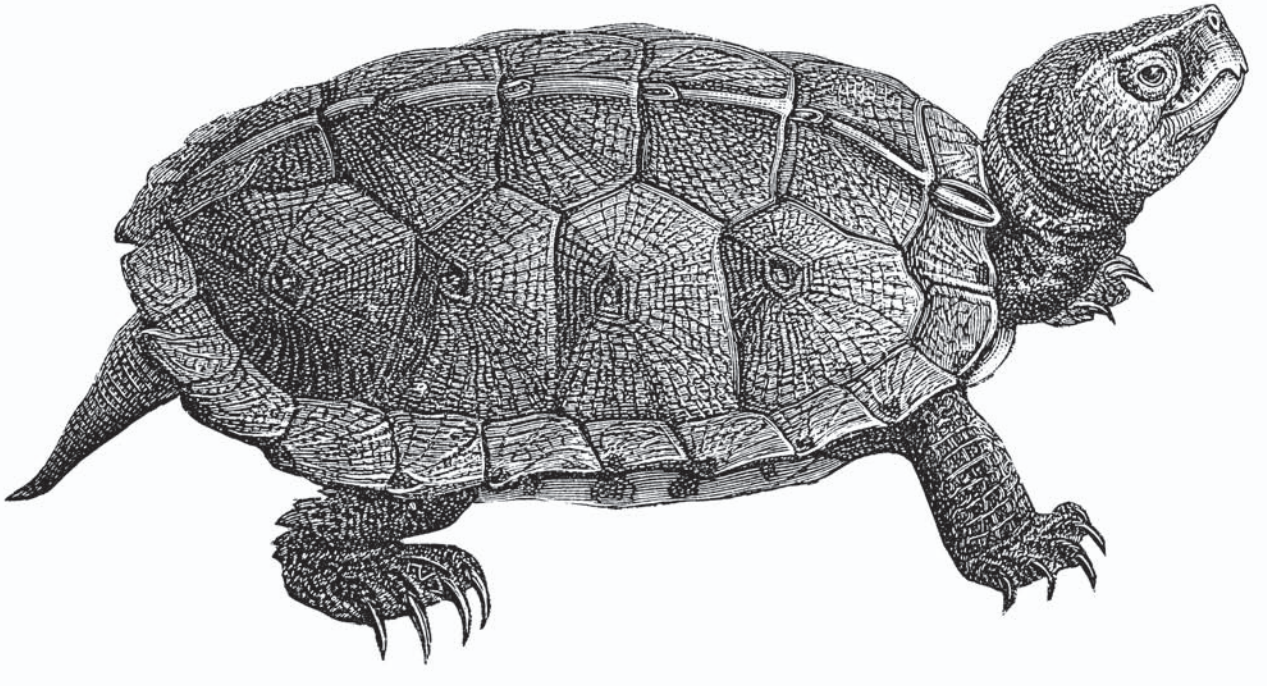
Solutions and Examples for bash Users

bash Cookbook™



O'REILLY®

Carl Albing, JP Vossen & Cameron Newham

bash Cookbook™

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

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.

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

ISBN 10: 0-596-52678-4
ISNB 13: 978-0-596-52678-8
[M]

Table of Contents

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 out 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:

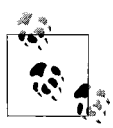
Plain text
: Indicates menu titles, menu options, menu buttons, and keyboard accelerators (such as Alt and Ctrl).

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

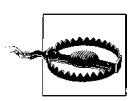
`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

`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