Perl语言编程（影印版）

第四版
上册

Programming

Perl

Tom Christiansen,
brian d foy, Larry Wall,
Jon Orwant 著

O'REILLY®
東南大學出版社

（第四版／上册）

Perl语言编程（影印版）

Programming Perl

Tom Christiansen, brian d foy & Larry Wall
with Jon Orwant 著

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo
O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社
·南 京·

图书在版编目（CIP）数据

Perl语言编程：第四版：英文/（美）克里斯蒂安森(Christiansen, T.)等著. —影印本. —南京：东南大学出版社，2012.6

书名原文：Programming Perl

ISBN 978-7-5641-3412-9

I. ①P… II. ①克… III. ①Perl语言－程序设计－英文 IV. ①TP312

中国版本图书馆CIP数据核字（2012）第065809号

江苏省版权局著作权合同登记

图字：10-2012-6号

Perl语言编程 第四版（影印版）

出版发行：东南大学出版社
地　　址：南京四牌楼2号　　邮编：210096
出 版 人：江建中
责任编辑：张烨
网　　址：http://www.seupress.com
电子邮件：press@seupress.com
印　　刷：扬中市印刷有限公司
开　　本：787毫米×980毫米　16开本
印　　张：75
字　　数：1469千字
版　　次：2012年6月第1版
印　　次：2012年6月第1次印刷
书　　号：ISBN 978-7-5641-3412-9
定　　价：128.00元（上下册）

本社图书若有印装质量问题，请直接与营销部联系。电话（传真）：025-83791830

Preface

The Pursuit of Happiness

Perl is a language for getting your job done.

Of course, if your job is programming, you can get your job done with any "complete" computer language, theoretically speaking. But we know from experience that computer languages differ not so much in what they make *possible*, but in what they make *easy*. At one extreme, the so-called "fourth generation languages" make it easy to do some things, but nearly impossible to do other things. At the other extreme, so-called "industrial-strength" languages make it equally difficult to do almost everything.

Perl is different. In a nutshell, Perl is designed to make the easy jobs easy, without making the hard jobs impossible.

And what are these "easy jobs" that ought to be easy? The ones you do every day, of course. You want a language that makes it easy to manipulate numbers and text, files and directories, computers and networks, and especially programs. It should be easy to run external programs and scan their output for interesting tidbits. It should be easy to send those same tidbits off to other programs that can do special things with them. It should be easy to develop, modify, and debug your own programs, too. And, of course, it should be easy to compile and run your programs, and do it portably, on any modern operating system.

Perl does all that, and a whole lot more.

Initially designed as a glue language for Unix, Perl has long since spread to most other operating systems. Because it runs nearly everywhere, Perl is one of the most portable programming environments available today. To program C or C ++ portably, you have to put in all those strange `#ifdef` markings for different operating systems. To program Java portably, you have to understand the idiosyncrasies of each new Java implementation. To program a shell script portably, you have to remember the syntax for each operating system's version of each

command, and somehow find the common factor that (you hope) works everywhere. And to program Visual Basic portably, you just need a more flexible definition of the word "portable". :-)

Perl happily avoids such problems while retaining many of the benefits of these other languages, with some additional magic of its own. Perl's magic comes from many sources: the utility of its feature set, the inventiveness of the Perl community, and the exuberance of the open source movement in general. But much of this magic is simply hybrid vigor; Perl has a mixed heritage, and has always viewed diversity as a strength rather than a weakness. Perl is a "give me your tired, your poor" language. If you feel like a huddled mass longing to be free, then Perl is for you.

Perl reaches out across cultures. Much of the explosive growth of Perl was fueled by the hankerings of *former* Unix systems programmers who wanted to take along with them as much of the "old country" as they could. For them, Perl is the portable distillation of Unix culture, an oasis in the wilderness of "can't get there from here". On the other hand, it also works in the other direction: Windows-based web designers are often delighted to discover that they can take their Perl programs and run them unchanged on the company's Unix servers.

Although Perl is especially popular with systems programmers and web developers, that's just because they discovered it first; Perl appeals to a much broader audience. From its small start as a text-processing language, Perl has grown into a sophisticated, general-purpose programming language with a rich software development environment complete with debuggers, profilers, cross-referencers, compilers, libraries, syntax-directed editors, and all the rest of the trappings of a "real" programming language—if you want them. But those are all about making hard things possible; and lots of languages can do that. Perl is unique in that it never lost its vision for keeping easy things easy.

Because Perl is both powerful and accessible, it is being used daily in every imaginable field, from aerospace engineering to molecular biology, from mathematics to linguistics, from graphics to document processing, from database manipulation to client-server network management. Perl is used by people who are desperate to analyze or convert lots of data quickly, whether you're talking DNA sequences, web pages, or pork belly futures.

There are many reasons for the success of Perl. Perl was a successful open source project long before the open source movement got its name. Perl is free, and it will always be free. You can use Perl however you see fit, subject only to a very liberal licensing policy. If you are in business and want to use Perl, go right ahead. You can embed Perl in the commercial applications you write without fee or

restriction. And if you have a problem that the Perl community can't fix, you have the ultimate backstop: the source code itself. The Perl community is not in the business of renting you their trade secrets in the guise of "upgrades". The Perl community will never "go out of business" and leave you with an orphaned product.

It certainly helps that Perl is free software. But that's not enough to explain the Perl phenomenon, since many freeware packages fail to thrive. Perl is not just free; it's also fun. People feel like they can be creative in Perl, because they have freedom of expression: they get to choose what to optimize for, whether that's computer speed or programmer speed, verbosity or conciseness, readability or maintainability or reusability or portability or learnability or teachability. You can even optimize for obscurity, if you're entering an Obfuscated Perl Contest.

Perl can give you all these degrees of freedom because it's a language with a split personality. It's simultaneously a very simple language and a very rich language. Perl has taken good ideas from nearly everywhere, and installed them into an easy-to-use mental framework. To those who merely like it, Perl is the *Practical Extraction and Report Language*. To those who love it, Perl is the *Pathologically Eclectic Rubbish Lister*. And to the minimalists in the crowd, Perl seems like a pointless exercise in redundancy. But that's okay. The world needs a few reductionists (mainly as physicists). Reductionists like to take things apart. The rest of us are just trying to get it together.

There are many ways in which Perl is a simple language. You don't have to know many special incantations to compile a Perl program—you can just execute it like a batch file or shell script. The types and structures used by Perl are easy to use and understand. Perl doesn't impose arbitrary limitations on your data—your strings and arrays can grow as large as they like (so long as you have memory), and they're designed to scale well as they grow. Instead of forcing you to learn new syntax and semantics, Perl borrows heavily from other languages you may already be familiar with (such as C, and *awk*, and BASIC, and Python, and English, and Greek). In fact, just about any programmer can read a well-written piece of Perl code and have some idea of what it does.

Most important, you don't have to know everything there is to know about Perl before you can write useful programs. You can learn Perl "small end first". You can program in Perl Baby-Talk, and we promise not to laugh. Or more precisely, we promise not to laugh any more than we'd giggle at a child's creative way of putting things. Many of the ideas in Perl are borrowed from natural language, and one of the best ideas is that it's okay to use a subset of the language as long as you get your point across. Any level of language proficiency is acceptable in

Perl culture. We won't send the language police after you. A Perl script is "correct" if it gets the job done before your boss fires you.

Though simple in many ways, Perl is also a rich language, and there is much to be learned about it. That's the price of making hard things possible. Although it will take some time for you to absorb all that Perl can do, you will be glad to have access to Perl's extensive capabilities when the time comes that you need them.

Because of its heritage, Perl was a rich language even when it was "just" a data-reduction language designed for navigating files, scanning large amounts of text, creating and obtaining dynamic data, and printing easily formatted reports based on that data. But somewhere along the line, Perl started to blossom. It also became a language for filesystem manipulation, process management, database administration, client-server programming, secure programming, Web-based information management, and even for object-oriented and functional programming. These capabilities were not just slapped onto the side of Perl—each new capability works synergistically with the others, because Perl was designed to be a glue language from the start.

But Perl can glue together more than its own features. Perl is designed to be modularly extensible. Perl allows you to rapidly design, program, debug, and deploy applications, and it also allows you to easily extend the functionality of these applications as the need arises. You can embed Perl in other languages, and you can embed other languages in Perl. Through the module importation mechanism, you can use these external definitions as if they were built-in features of Perl. Object-oriented external libraries retain their object-orientedness in Perl.

Perl helps you in other ways, too. Unlike a strictly interpreted language such as command files or shell scripts, which compile and execute a program one command at a time, Perl first compiles your whole program quickly into an intermediate format. Like any other compiler, it performs various optimizations, and gives you instant feedback on everything from syntax and semantic errors to library binding mishaps. Once Perl's compiler frontend is happy with your program, it passes off the intermediate code to the interpreter to execute (or optionally to any of several modular backends that can emit C or bytecode). This all sounds complicated, but the compiler and interpreter are quite efficient, and most of us find that the typical compile-run-fix cycle is measured in mere seconds. Together with Perl's many fail-soft characteristics, this quick turnaround capability makes Perl a language in which you really can do rapid prototyping. Then later, as your program matures, you can tighten the screws on yourself, and make yourself program with less flair but more discipline. Perl helps you with that, too, if you ask nicely.

Perl also helps you to write programs more securely. In addition to all the typical security interfaces provided by other languages, Perl also guards against accidental security errors through a unique data tracing mechanism that automatically determines which data came from insecure sources and prevents dangerous operations before they can happen. Finally, Perl lets you set up specially protected compartments in which you can safely execute Perl code of dubious origin, masking out dangerous operations.

But, paradoxically, the way in which Perl helps you the most has almost nothing to do with Perl, and everything to do with the people who use Perl. Perl folks are, frankly, some of the most helpful folks on earth. If there's a religious quality to the Perl movement, then this is at the heart of it. Larry wanted the Perl community to function like a little bit of heaven, and by and large he seems to have gotten his wish, so far. Please do your part to keep it that way.

Whether you are learning Perl because you want to save the world, or just because you are curious, or because your boss told you to, this handbook will lead you through both the basics and the intricacies. And although we don't intend to teach you how to program, the perceptive reader will pick up some of the art, and a little of the science, of programming. We will encourage you to develop the three great virtues of a programmer: *laziness*, *impatience*, and *hubris*. Along the way, we hope you find the book mildly amusing in some spots (and wildly amusing in others). And if none of this is enough to keep you awake, just keep reminding yourself that learning Perl will increase the value of your resume. So keep reading.

What's New in This Edition

What's not new? It's been a long time since we've updated this book. Let's just say we had a couple of distractions, but we're all better now.

The third edition was published in the middle of 2000, just as Perl v5.6 was coming out. As we write this, it's 12 years later and Perl v5.16 is coming out soon. A lot has happened in those years, including several new releases of Perl 5, and a little thing we call Perl 6. That 6 is deceptive though; Perl 6 is really a "kid sister" language to Perl 5, and not just a major update to Perl 5 that version numbers have trained you to expect. This book isn't about that other language. It's still

about Perl 5, the version that most people in the world (even the Perl 6 folks!) are still using quite productively.[1]

To tell you what's new in this book is to tell you what's new in Perl. This isn't just a facelift to spike book sales. It's a long anticipated major update for a language that's been very active in the past five years. We won't list everything that's changed (you can read the *perldelta* pages), but there are some things we'd like to call out specifically.

In Perl 5, we started adding major new features, along with a way to shield older programs from new keywords. For instance, we finally relented to popular demand for a `switch`-like statement. In typical Perl fashion, though, we made it better and more fancy, giving you more control to do what you need to do. We call it `given-when`, but you only get that feature if you ask for it. Any of these statements enable the feature:

```
use v5.10;
use feature qw(switch);
use feature qw(:5.10);
```

and once enabled, you have your super-charged `switch`:

```
given ($item) {
  when (/a/)   { say "Matched an a"  }
  when (/bee/) { say "Matched a bee" }
}
```

You'll see more about that in Chapter 4, along with many of the other new features as they appear where they make the most sense.

Although Perl has had Unicode support since v5.6, that support is greatly improved in recent versions, including better regular expression support than any other language at the moment. Perl's better-and-better support is even acting as a testbed for future Unicode developments. In the previous edition of this book, we had all of that Unicode stuff in one chapter, but you'll find it throughout this book when we need it.

Regular expressions, the feature that many people associate with Perl, are even better. Other languages stole Perl's pattern language, calling it Perl Compatible Regular Expressions, but also adding some features of their own. We've stolen back some of those features, continuing Perl's tradition of taking the best ideas from everywhere and everything. You'll also find powerful new features for dealing with Unicode in patterns.

---

1. Since we're lazy, and since by now you already know this book is about Perl 5, we should mention that we won't always spell out "Perl v5.*n*"—for the rest of this book, if you see a bare version number that starts with "v5", just assume we're talking about that version of Perl.

Threads are much different today, too. Perl used to support two thread models: one we called `5005threads` (because that's when we added them), and interpreter threads. As of v5.10, it's just the interpreter threads. However, for various reasons, we didn't think we could do the topic justice in this edition since we dedicated our time to many of the other features. If you want to learn about threads, see the *perlthrtut* manpage, which is approximately the same thing as our "Threads" chapter would have been. Maybe we can provide a bonus chapter later, though.

Other things have come or gone. Some experiments didn't work out and we took them out of Perl, replacing them with other experiments. Pseudohashes, for instance, were deprecated, removed, and forgotten. If you don't know what those are, don't worry about it, but don't look for them in this edition either.

And, since we last updated this book, there's been a tremendous revolution (or two) in Perl programming practice as well as its testing culture. CPAN (the Comprehensive Perl Archive Network) continues to grow exponentially, making it Perl's killer feature. This isn't a book about CPAN, though, but we tell you about those modules when they are important. Don't try to do everything with just vanilla Perl.

We've also removed two chapters, the list of modules in the Standard Library (Chapter 32 in the previous edition) and the diagnostic messages (Chapter 33 in the previous edition). Both of these will be out of date before the book even gets on your bookshelf. We'll show you how to get that list yourself. For the diagnostic messages, you can find all of them in the *perldiag* manpage, or turn warnings into longer messages with the `diagnostics` pragma.

Part I, Overview

Getting started is always the hardest part. This part presents the fundamental ideas of Perl in an informal, curl-up-in-your-favorite-chair fashion. Not a full tutorial, it merely offers a quick jump-start, which may not serve everyone. See the section on "Offline Documentation" below for other books that might better suit your learning style.

Part II, The Gory Details

This part consists of an in-depth, no-holds-barred discussion of the guts of the language at every level of abstraction, from data types, variables, and regular expressions, to subroutines, modules, and objects. You'll gain a good sense of how the language works, and in the process, pick up a few hints on good software design. (And if you've never used a language with pattern matching, you're in for a special treat.)

Part III, Perl As Technology

You can do a lot with Perl all by itself, but this part will take you to a higher level of wizardry. Here you'll learn how to make Perl jump through whatever hoops your computer sets up for it, everything from dealing with Unicode, interprocess communication and multithreading, through compiling, invoking, debugging, and profiling Perl, on up to writing your own external extensions in C or C++, or interfaces to any existing API you feel like. Perl will be quite happy to talk to any interface on your computer—or, for that matter, on any other computer on the Internet, weather permitting.

Part IV, Perl As Culture

Everyone understands that a culture must have a language, but the Perl community has always understood that a language must have a culture. This part is where we view Perl programming as a human activity, embedded in the real world of people. We'll cover how you can improve the way you deal with both good people and bad people. We'll also dispense a great deal of advice on how you can become a better person yourself, and on how to make your programs more useful to other people.

Part V, Reference Material

Here we've put together all the chapters in which you might want to look something up alphabetically, everything from special variables and functions to standard modules and pragmas. The Glossary will be particularly helpful to those who are unfamiliar with the jargon of computer science. For example, if you don't know what the meaning of "pragma" is, you could look it up right now. (If you don't know what the meaning of "is" is, we can't help you with that.)

The Standard Distribution

The official Perl policy, as noted in *perlpolicy*, is that the last two maintenance releases are officially supported. Since the current release as we write this is v5.14, that means both v5.12 and v5.14 are officially supported. When v5.16 is released, v5.12 won't be supported anymore.

Most operating system vendors these days include Perl as a standard component of their systems, although their release cycles might not track the latest Perl. As of this writing, AIX, BeOS, BSDI, Debian, DG/UX, DYNIX/ptx, FreeBSD, IRIX, LynxOS, Mac OS X, OpenBSD, OS390, RedHat, SINIX, Slackware, Solaris, SuSE, and Tru64 all came with Perl as part of their standard distributions. Some companies provide Perl on separate CDs of contributed freeware or through their customer service groups. Third-party companies like ActiveState offer prebuilt

Perl distributions for a variety of different operating systems, including those from Microsoft.

Even if your vendor does ship Perl as standard, you'll probably eventually want to compile and install Perl on your own. That way you'll know you have the latest version, and you'll be able to choose where to install your libraries and documentation. You'll also be able to choose whether to compile Perl with support for optional extensions such as multithreading, large files, or the many low-level debugging options available through the *-D* command-line switch. (The user-level Perl debugger is always supported.)

The easiest way to download a Perl source kit is probably to point your web browser to Perl's homepage (*http://www.perl.org*), where you'll find download information prominently featured on the start-up page, along with links to precompiled binaries for platforms that have misplaced their C compilers.

You can also head directly to CPAN, described in Chapter 19, using *http://www.cpan.org*. If those are too slow for you (and they might be, because they're *very* popular), you should find a mirror close to you. The MIRRORED.BY (*http://www.cpan.org/SITES.html*) file there contains a list of all other CPAN sites, so you can just get that file and then pick your favorite mirror. Some of them are available through FTP, others through HTTP (which makes a difference behind some corporate firewalls). The *http://www.cpan.org* multiplexor attempts to do this selection for you. You can change your selection if you like later.

Once you've fetched the source code and unpacked it into a directory, you should read the *README* and the *INSTALL* files there to learn how to build Perl. There may also be an *INSTALL.platform* file for you to read there, where *platform* represents your operating system platform.

If your *platform* happens to be some variety of Unix, then your commands to fetch, configure, build, and install Perl might resemble what follows. First, you must choose a command to fetch the source code. You can download via the Web using a browser or a command-line tool:

```
% wget http://www.cpan.org/src/5.0/maint.tar.gz
```

Now unpack, configure, build, and install:

```
% tar zxf latest.tar.gz     # or gunzip first, then tar xf
% cd perl-5.14.2            # or 5.* for whatever number
% sh Configure -des         # assumes default answers
% make test && make install # install typically requires superuser
```

Your platform might already have packages that do this work for you (as well as providing platform-specific fixes or enhancements). Even then, many platforms already come with Perl, so you might not need to do anything.

If you already have Perl but want a different version, you can save yourself some work by using the *perlbrew* tool. It automates all of this for you and installs it where you (should) have permissions to install files so you don't need any administrator privileges. It's on CPAN as `App::perlbrew`, but you can also install it according to the documentation:

```
% curl -L http://xrl.us/perlbrewinstall | bash
```

Once installed, you can let the tool do all the work for you:

```
% ~/perl5/perlbrew/bin/perlbrew install perl-5.14.2
```

There's a lot more that perlbrew can do for you, so see its documentation.

You can also get enhanced versions of the standard Perl distribution. ActiveState offers ActivePerl (*http://www.activestate.com/activeperl/downloads*) for free for Windows, Mac OS X, and Linux, and for a fee for Solaris, HP-UX, and AIX.

Strawberry Perl (*http://www.strawberryperl.org*) is Windows-only, and it comes with the various tools you need to compile and install third-party Perl modules for CPAN.

Citrus Perl (*http://www.citrusperl.com/*) is a distribution for Windows, Mac OS X, and Linux that bundles wxPerl tools for creating GUIs. It's targeted at people who want to create distributed GUI applications with Perl instead of a general-purpose Perl. Its Cava Packager (*http://www.cava.co.uk/*) tool helps you do that.

Online Documentation

Perl's extensive online documentation comes as part of the standard Perl distribution. (See the next section for offline documentation.) Additional documentation shows up whenever you install a module from CPAN.

When we refer to a "Perl manpage" in this book, we're talking about this set of online Perl manual pages, sitting on your computer. The name *manpage* is purely a convention meaning a file containing documentation—you don't need a Unix-style *man* program to read one. You may even have the Perl manpages installed as HTML pages, especially on non-Unix systems.

The online manpages for Perl have been divided into separate sections so you can easily find what you are looking for without wading through hundreds of pages of text. Since the top-level manpage is simply called *perl*, the Unix command

"`man perl`" should take you to it.[2] That page in turn directs you to more specific pages. For example, "`man perlre`" will display the manpage for Perl's regular expressions. The *perldoc* command often works on systems when the *man* command won't. Your port may also provide the Perl manpages in HTML format or your system's native help format. Check with your local sysadmin, unless you're the local sysadmin. In which case, ask the monks at *http://perlmonks.org*.

Navigating the Standard Manpages

In the Beginning (of Perl, that is, back in 1987), the *perl* manpage was a terse document, filling about 24 pages when typeset and printed. For example, its section on regular expressions was only two paragraphs long. (That was enough, if you knew *egrep*.) In some ways, nearly everything has changed since then. Counting the standard documentation, the various utilities, the per-platform porting information, and the scads of standard modules, we now have thousands of typeset pages of documentation spread across many separate manpages. (And that's not counting any CPAN modules you install, which is likely to be quite a few.)

But in other ways, nothing has changed: there's still a *perl* manpage kicking around. And it's still the right place to start when you don't know where to start. The difference is that once you arrive, you can't just stop there. Perl documentation is no longer a cottage industry; it's a supermall with hundreds of stores. When you walk in the door, you need to find the YOU ARE HERE to figure out which shop or department store sells what you're shopping for. Of course, once you get familiar with the mall, you'll usually know right where to go.

A few of the store signs you'll see are shown in Table P-1.

Table P-1. Selected Perl manpages

Manpage	Covers
perl	What perl manpages are available
perldata	Data types
perlsyn	Syntax
perlop	Operators and precedence
perlre	Regular expressions
perlvar	Predefined variables

2. If you still get a truly humongous page when you do that, you're probably picking up the ancient v4 manpage. Check your `MANPATH` for archaeological sites. (Say "`perldoc perl`" to find out how to configure your `MANPATH` based on the output of "`perl -V:man.dir`".)

Manpage	Covers
perlsub	Subroutines
perlfunc	Built-in functions
perlmod	How perl modules work
perlref	References
perlobj	Objects
perlipc	Interprocess communication
perlrun	How to run Perl commands, plus switches
perldebug	Debugging
perldiag	Diagnostic messages

That's just a small excerpt, but it has the important parts. You can tell that if you want to learn about an operator, that *perlop* is apt to have what you're looking for. And if you want to find something out about predefined variables, you'd check in *perlvar*. If you got a diagnostic message you didn't understand, you'd go to *perldiag*. And so on.

Part of the standard Perl manual is the frequently asked questions (FAQ) list. It's split up into these nine different pages, as shown in Table P-2.

Table P-2. The perlfaq manpages

Manpage	Covers
perlfaq1	General questions about Perl
perlfaq2	Obtaining and learning about Perl
perlfaq3	Programming tools
perlfaq4	Data manipulation
perlfaq5	Files and formats
perlfaq6	Regular expressions
perlfaq7	General Perl language issues
perlfaq8	System interaction
perlfaq9	Networking

Some manpages contain platform-specific notes, as listed in Table P-3.

Table P-3. Platform-specific manpages

Manpage	Covers
perlamiga	The Amiga port
perlcygwin	The Cygwin port
perldos	The MS-DOS port
perlhpux	The HP-UX port
perlmachten	The Power MachTen port
perlos2	The OS/2 port
perlos390	The OS/390 port
perlvms	The DEC VMS port
perlwin32	The MS-Windows port

(See also Chapter 22 and the CPAN ports (*http://www.cpan.org/ports/index.html*) directory described earlier for porting information.)

Non-Perl Manpages

When we refer to non-Perl documentation, as in *getitimer*(2), this refers to the *getitimer* manpage from section 2 of the Unix Programmer's Manual.[3] Manpages for syscalls such as *getitimer* may not be available on non-Unix systems, but that's probably okay, because you couldn't use the Unix syscall there anyway. If you really do need the documentation for a Unix command, syscall, or library function, many organizations have put their manpages on the Web—a quick search of Google for `crypt(3) manual` will find many copies.

Although the top-level Perl manpages are typically installed in section 1 of the standard *man* directories, we will omit appending a (1) to those manpage names in this book. You can recognize them anyway because they are all of the form "`perl`*`mumble`*".

3. Section 2 is only supposed to contain direct calls into the operating system. (These are often called "system calls", but we'll consistently call them *syscalls* in this book to avoid confusion with the `system` function, which has nothing to do with syscalls). However, systems vary somewhat in which calls are implemented as syscalls, and which are implemented as C library calls, so you could conceivably find *getitimer*(2) in section 3 instead.

Offline Documentation

If you'd like to learn more about Perl, here are some related publications that we recommend:

- *Perl 5 Pocket Reference*, by Johan Vromans; O'Reilly Media (5th Edition, July 2011). This small booklet serves as a convenient quick-reference for Perl.
- *Perl Cookbook*, by Tom Christiansen and Nathan Torkington; O'Reilly Media (2nd Edition, August 2003). This is the companion volume to the book you have in your hands right now. This cookbook's recipes teach you how to cook with Perl.
- *Learning Perl*, by Randal Schwartz, brian d foy, and Tom Phoenix; O'Reilly Media (6th Edition, June 2011). This book teaches programmers the 30% of basic Perl they'll use 70% of the time, and it is targeted at people writing self-contained programs around a couple of hundred lines.
- *Intermediate Perl*, by Randal Schwartz, brian d foy, and Tom Phoenix; O'Reilly Media (March 2006). This book picks up where *Learning Perl* left off, introducing references, data structures, packages, objects, and modules.
- *Mastering Perl*, by brian d foy; O'Reilly Media (July 2007). This book is the final book in the trilogy along with *Learning Perl* and *Intermediate Perl*. Instead of focusing on language fundamentals, it shifts gears to teaching the Perl programmer about applying Perl to the work at hand.
- *Modern Perl*, by chromatic; Oynx Neon (October 2010). This book provides a survey of modern Perl programming practice and topics, suitable for people who know programming already but haven't paid attention to recent developments in Perl.
- *Mastering Regular Expressions*, by Jeffrey Friedl; O'Reilly Media (3rd Edition, August 2006). Although it doesn't cover the latest additions to Perl regular expressions, this book is an invaluable reference for anyone seeking to learn how regular expressions work.
- *Object Oriented Perl*, by Damian Conway; Manning (August 1999). For beginning as well as advanced OO programmers, this book explains common and esoteric techniques for writing powerful object systems in Perl.
- *Mastering Algorithms with Perl*, by Jon Orwant, Jarkko Hietaniemi, and John Macdonald; O'Reilly Media (1999). All the useful techniques from a CS algorithms course but without the painful proofs. This book covers fundamental and useful algorithms in the fields of graphs, text, sets, and more.