

ADVANCE PRAISE FOR *PERL ONE-LINERS*

“One of the slogans used by Perl is ‘Easy things should be easy and hard things should be possible.’ This book illustrates just how easy things can be—and how much can be done with so little code.”

—DAVID PRECIOUS, CONTRIBUTOR TO THE PERL DANCER PROJECT AND VARIOUS CPAN MODULES

“By reading this book you can make a step toward becoming the local computer wizard, even without learning how to program.”

—GABOR SZABO, FOUNDER AND EDITOR OF THE *PERL WEEKLY* NEWSLETTER

“A set of exercises for deepening your understanding of Perl.”

—JOHN D. COOK, SINGULAR VALUE CONSULTING

“The author is enthusiastic about the material and uses an easy writing style. Highly recommended.”

—THRIG (JEREMY MATES), INTERNET PLUMBER

“These one-liners are great. Simple. Clear. Concise.”

—JONATHAN SCOTT DUFF, PERL GURU

“A quick read full of useful command-line Perl programs.”

—CHRIS FEDDE, SYSTEMS ENGINEER AND PERL ENTHUSIAST

“Handy for anyone who does a lot of one-off text processing: system administrators, coders, or anyone with large amounts of data they need shifted, filtered, or interpreted.”

—JIM DAVIS, PERL DEVELOPER



PERL ONE-LINERS



PERL ONE-LINERS

**130 Programs
That Get Things Done**

by Peteris Krumins



**no starch
press**

San Francisco

PERL ONE-LINERS. Copyright © 2014 by Peteris Kruminis.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Printed in USA

First printing

17 16 15 14 13 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-520-X

ISBN-13: 978-1-59327-520-4

Publisher: William Pollock

Production Editor: Riley Hoffman

Cover Illustration: Tina Salameh

Interior Design: Octopod Studios

Developmental Editor: William Pollock

Technical Reviewer: Alastair McGowan-Douglas

Copyeditor: LeeAnn Pickrell

Compositor: Riley Hoffman

Proofreader: Elaine Merrill

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

phone: 415.863.9900; fax: 415.863.9950; info@nostarch.com; www.nostarch.com

Library of Congress Cataloging-in-Publication Data

Krumins, Peteris.

Perl one-liners : 130 programs that get things done / by Peteris Kruminis.
pages cm

Summary: "Snappy Perl programs to streamline tasks and sharpen coding skills"-- Provided by publisher.

ISBN 978-1-59327-520-4 (paperback) -- ISBN 1-59327-520-X (paperback)

1. Perl (Computer program language) I. Title.

QA76.73.P22K78 2013

005.13'3--dc23

2013030613

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

About the Author

Peteris Krumins is a programmer, systems administrator, blogger, and all-around hacker. He is currently running his own company, Browserling, which focuses on cross-browser testing. He has self-published three books on essential UNIX tools, and he enjoys open-sourcing hundreds of small projects on GitHub.

Find his website and blog at <http://www.catonmat.net/>, follow @pkrumins on Twitter, and see his open source projects at <http://github.com/pkrumins/>.



About the Technical Reviewer

Alastair McGowan-Douglas lives in Rugby in the UK. He has been a Perl developer since 2008 and is now stuck writing PHP for a living. His favorite pastime at work is writing Perl scripts for internal use to encourage others to embrace the language. Also a JavaScript developer and Git aficionado, his rantings and musings on these various subjects can be found at <http://altreus.blogspot.com/>.



BRIEF CONTENTS

Acknowledgments	xvii
Chapter 1: Introduction to Perl One-Liners	1
Chapter 2: Spacing	7
Chapter 3: Numbering	17
Chapter 4: Calculations	29
Chapter 5: Working with Arrays and Strings	49
Chapter 6: Text Conversion and Substitution	59
Chapter 7: Selectively Printing and Deleting Lines	69
Chapter 8: Useful Regular Expressions	83
Appendix A: Perl's Special Variables	95
Appendix B: Using Perl One-Liners on Windows	105
Appendix C: perl1line.txt	117
Index	139



CONTENTS IN DETAIL

ACKNOWLEDGMENTS	xvii
1	
INTRODUCTION TO PERL ONE-LINERS	1
2	
SPACING	7
2.1 Double-space a file	7
2.2 Double-space a file, excluding the blank lines	11
2.3 Triple-space a file	11
2.4 N-space a file	12
2.5 Add a blank line before every line	12
2.6 Remove all blank lines	12
2.7 Remove all consecutive blank lines, leaving only one	14
2.8 Compress/expand all blank lines into N consecutive lines	14
2.9 Double-space between all words	15
2.10 Remove all spacing between words	15
2.11 Change all spacing between words to one space.	16
2.12 Insert a space between all characters	16
3	
NUMBERING	17
3.1 Number all lines in a file	17
3.2 Number only non-empty lines in a file	18
3.3 Number and print only non-empty lines in a file (drop empty lines)	19
3.4 Number all lines but print line numbers only for non-empty lines	20
3.5 Number only lines that match a pattern; print others unmodified	20
3.6 Number and print only lines that match a pattern	21
3.7 Number all lines but print line numbers only for lines that match a pattern	21
3.8 Number all lines in a file using a custom format	22
3.9 Print the total number of lines in a file (emulate wc -l)	22
3.10 Print the number of non-empty lines in a file	24
3.11 Print the number of empty lines in a file	25
3.12 Print the number of lines in a file that match a pattern (emulate grep -c)	25
3.13 Number words across all lines	26
3.14 Number words on each individual line	26
3.15 Replace all words with their numeric positions	27

4
CALCULATIONS **29**

4.1	Check if a number is a prime	29
4.2	Print the sum of all fields on each line	30
4.3	Print the sum of all fields on all lines	31
4.4	Shuffle all fields on each line	32
4.5	Find the numerically smallest element (minimum element) on each line	33
4.6	Find the numerically smallest element (minimum element) over all lines	33
4.7	Find the numerically largest element (maximum element) on each line	35
4.8	Find the numerically largest element (maximum element) over all lines	35
4.9	Replace each field with its absolute value	36
4.10	Print the total number of fields on each line	36
4.11	Print the total number of fields on each line, followed by the line	37
4.12	Print the total number of fields on all lines	37
4.13	Print the total number of fields that match a pattern	38
4.14	Print the total number of lines that match a pattern	38
4.15	Print the number π	39
4.16	Print the number e	39
4.17	Print UNIX time (seconds since January 1, 1970, 00:00:00 UTC)	39
4.18	Print Greenwich Mean Time and local computer time	40
4.19	Print yesterday's date	41
4.20	Print the date 14 months, 9 days, and 7 seconds ago.	41
4.21	Calculate the factorial.	41
4.22	Calculate the greatest common divisor	42
4.23	Calculate the least common multiple	43
4.24	Generate 10 random numbers between 5 and 15 (excluding 15)	43
4.25	Generate all permutations of a list	44
4.26	Generate the powerset	45
4.27	Convert an IP address to an unsigned integer	45
4.28	Convert an unsigned integer to an IP address	47

5
WORKING WITH ARRAYS AND STRINGS **49**

5.1	Generate and print the alphabet	49
5.2	Generate and print all the strings from "a" to "zz"	50
5.3	Create a hex lookup table	51
5.4	Generate a random eight-character password	51
5.5	Create a string of specific length	52
5.6	Create an array from a string	52
5.7	Create a string from the command-line arguments	53
5.8	Find the numeric values for characters in a string	53
5.9	Convert a list of numeric ASCII values into a string	55
5.10	Generate an array with odd numbers from 1 to 100	55
5.11	Generate an array with even numbers from 1 to 100	56
5.12	Find the length of a string	56
5.13	Find the number of elements in an array	56

6 **TEXT CONVERSION AND SUBSTITUTION** **59**

6.1	ROT13 a string	59
6.2	Base64-encode a string	60
6.3	Base64-decode a string	61
6.4	URL-escape a string	61
6.5	URL-unescape a string	61
6.6	HTML-encode a string	62
6.7	HTML-decode a string	62
6.8	Convert all text to uppercase	62
6.9	Convert all text to lowercase	62
6.10	Uppercase only the first letter of each line	63
6.11	Invert the letter case	63
6.12	Title-case each line	63
6.13	Strip leading whitespace (spaces, tabs) from the beginning of each line	64
6.14	Strip trailing whitespace (spaces, tabs) from the end of each line	64
6.15	Strip whitespace (spaces, tabs) from the beginning and end of each line	64
6.16	Convert UNIX newlines to DOS/Windows newlines	65
6.17	Convert DOS/Windows newlines to UNIX newlines	65
6.18	Convert UNIX newlines to Mac newlines	65
6.19	Substitute (find and replace) "foo" with "bar" on each line	66
6.20	Substitute (find and replace) "foo" with "bar" on lines that match "baz"	66
6.21	Print paragraphs in reverse order	66
6.22	Print all lines in reverse order	67
6.23	Print columns in reverse order	67

7 **SELECTIVELY PRINTING AND DELETING LINES** **69**

7.1	Print the first line of a file (emulate head -1)	70
7.2	Print the first 10 lines of a file (emulate head -10)	70
7.3	Print the last line of a file (emulate tail -1)	71
7.4	Print the last 10 lines of a file (emulate tail -10)	72
7.5	Print only lines that match a regular expression	72
7.6	Print only lines that do not match a regular expression	73
7.7	Print every line preceding a line that matches a regular expression	73
7.8	Print every line following a line that matches a regular expression	74
7.9	Print lines that match regular expressions AAA and BBB in any order	75
7.10	Print lines that don't match regular expressions AAA and BBB	75
7.11	Print lines that match regular expression AAA followed by BBB followed by CCC	75
7.12	Print lines that are at least 80 characters long	76
7.13	Print lines that are fewer than 80 characters long	76
7.14	Print only line 13	76
7.15	Print all lines except line 27	76
7.16	Print only lines 13, 19, and 67	77
7.17	Print all lines from 17 to 30	77
7.18	Print all lines between two regular expressions (including the lines that match)	78

7.19	Print the longest line	78
7.20	Print the shortest line	79
7.21	Print all lines containing digits	79
7.22	Print all lines containing only digits	79
7.23	Print all lines containing only alphabetic characters	80
7.24	Print every second line	80
7.25	Print every second line, beginning with the second line	80
7.26	Print all repeated lines only once	81
7.27	Print all unique lines	81

8
USEFUL REGULAR EXPRESSIONS **83**

8.1	Match something that looks like an IP address	83
8.2	Test whether a number is in the range 0 to 255	84
8.3	Match an IP address	85
8.4	Check whether a string looks like an email address	86
8.5	Check whether a string is a number	87
8.6	Check whether a word appears in a string twice	88
8.7	Increase all integers in a string by one	89
8.8	Extract the HTTP User-Agent string from HTTP headers	89
8.9	Match printable ASCII characters	90
8.10	Extract text between two HTML tags	90
8.11	Replace all tags with 	91
8.12	Extract all matches from a regular expression	92

A
PERL'S SPECIAL VARIABLES **95**

A.1	Variable \$_	95
	Using \$_ with the -n argument	96
	Using \$_ with the -p argument	97
	Using \$_ explicitly	98
A.2	Variable \$.	99
A.3	Variable \$/	100
A.4	Variable \$\	101
A.5	Variables \$1, \$2, \$3, and so on	101
A.6	Variable \$,	102
A.7	Variable \$"	102
A.8	Variable @F	103
A.9	Variable @ARGV	103
A.10	Variable %ENV	104

B
USING PERL ONE-LINERS ON WINDOWS **105**

B.1	Perl on Windows	105
B.2	Bash on Windows	106

B.3	Perl One-Liners in Windows Bash	107
B.4	Perl One-Liners in the Windows Command Prompt	108
	Converting One-Liners in the Windows Command Prompt	108
	Symbol Challenges	110
	Windows File Paths	111
B.5	Perl One-Liners in PowerShell	111
	Converting One-Liners in PowerShell	112
	One-Liners in PowerShell 3.0+	114

C

PERL1LINE.TXT 117

C.1	Spacing	117
C.2	Numbering	119
C.3	Calculations	121
C.4	Working with Arrays and Strings	127
C.5	Text Conversion and Substitution	130
C.6	Selectively Printing and Deleting Lines	133
C.7	Useful Regular Expressions	136

INDEX 139



ACKNOWLEDGMENTS

I'd like to thank Eric Pement for inspiring me to write this book; Bill Pollock for giving me the opportunity to publish it at No Starch Press; Riley Hoffman and Laurel Chun for working with me to make it perfect; Alastair McGowan-Douglas for his technical review; and David Precious, Gabor Szabo, Jim Davis, Chris Fedde, Andy Lester, John D. Cook, Jonathan Scott Duff, and Jeremy Mates for reviewing the book and making great suggestions for improvements. I'd also like to thank everyone who helped me on the #perl IRC channel on freenode. If I forgot anyone, I'm sorry, but thanks for helping me to get this book written!



1

INTRODUCTION TO PERL ONE-LINERS

Perl one-liners are small and awesome Perl programs that fit in a single line of code. They do one thing really well—like changing line spacing, numbering lines, performing calculations, converting and substituting text, deleting and printing specific lines, parsing logs, editing files in-place, calculating statistics, carrying out system administration tasks, or updating a bunch of files at once. Perl one-liners will make you a shell warrior: what took you minutes (or even hours) to solve will now take you only seconds!

In this introductory chapter, I'll show you what one-liners look like and give you a taste of what's in the rest of the book. This book requires some Perl knowledge, but most of the one-liners can be tweaked and modified without knowing the language in depth.

Let's look at some examples. Here's one:

```
perl -pi -e 's/you/me/g' file
```

This one-liner replaces all occurrences of the text *you* with *me* in the file *file*. Very useful if you ask me. Imagine you're on a remote server and you need to replace text in a file. You can either open the file in a text editor and execute find-replace or simply perform the replacement through the command line and, bam, be done with it.

This one-liner and others in this book work well in UNIX. I'm using Perl 5.8 to run them, but they also work in newer Perl versions, such as Perl 5.10 and later. If you're on a Windows computer, you'll need to change them a little. To make this one-liner work on Windows, swap the single quotes for double quotes. To learn more about using Perl one-liners on Windows, see Appendix B.

I'll be using Perl's `-e` command-line argument throughout the book. It allows you to use the command line to specify the Perl code to be executed. In the previous one-liner, the code says "do the substitution (`s/you/me/g` command) and replace *you* with *me* globally (`/g` flag)." The `-p` argument ensures that the code is executed on every line of input and that the line is printed after execution. The `-i` argument ensures that *file* is edited in-place. Editing *in-place* means that Perl performs all the substitutions right in the file, overwriting the content you want to replace. I recommend that you always make a backup of the file you're working with by specifying the backup extension to the `-i` argument, like this:

```
perl -pi.bak -e 's/you/me/g' file
```

Now Perl creates a *file.bak* backup file first and only then changes the contents of *file*.

How about doing this same replacement in multiple files? Just specify the files on the command line:

```
perl -pi -e 's/you/me/g' file1 file2 file3
```

Here, Perl first replaces *you* with *me* in *file1* and then does the same in *file2* and *file3*.

You can also perform the same replacement only on lines that match *we*, as simply as this:

```
perl -pi -e 's/you/me/g if /we/' file
```

Here, you use the conditional `if /we/` to ensure that `s/you/me/g` is executed only on lines that match the regular expression `/we/`.

The regular expression can be anything. Say you want to execute the substitution only on lines with digits in them. You could use the `/\d/` regular expression to match numbers:

```
perl -pi -e 's/you/me/g if /\d/' file
```

How about finding all lines in a file that appear more than once?

```
perl -ne 'print if $a{$_}++' file
```

This one-liner records the lines you've seen so far in the `%a` hash and counts the number of times it sees the lines. If it has already seen the line, the condition `$a{$_}++` is true, so it prints the line. Otherwise it “automagically” creates an element that contains the current line in the `%a` hash and increments its value. The `$_` special variable contains the current line. This one-liner also uses the `-n` command-line argument to loop over the input, but unlike `-p`, it doesn't print the lines automatically. (Don't worry about all the command-line arguments right now; you'll learn about them as you work through this book!)

How about numbering lines? Super simple! Perl's `$.` special variable maintains the current line number. Just print it together with the line:

```
perl -ne 'print "$. $_" file
```

You can do the same thing by using the `-p` argument and modifying the `$_` variable:

```
perl -pe '$_ = "$. $_" file
```

Here, each line is replaced by the string `$. $_`, which is equal to the current line number followed by the line itself. (See one-liner 3.1 on page 17 for a full explanation.)

If you omit the filename at the end of the one-liner, Perl reads data from standard input. From now on, I'll assume the data comes from the standard input and drop the filename at the end. You can always put it back if you want to run one-liners on whole files.

You can also combine the previous two one-liners to create one that numbers only the repeated lines:

```
perl -ne 'print "$. $_" if $a{$_}++'
```

Another thing you can do is sum the numbers in each line using the `sum` function from the `List::Util` CPAN module. CPAN (Comprehensive Perl Archive Network; <http://www.cpan.org/>) is an archive of over 100,000

reusable Perl modules. `List::Util` is one of the modules on CPAN, and it contains various list utility functions. You don't need to install this module because it comes with Perl. (It's in Perl core.)

```
perl -MList::Util=sum -alne 'print sum @F'
```

The `-MList::Util` command-line argument imports the `List::Util` module. The `=sum` part of this one-liner imports the `sum` function from the `List::Util` module so that the program can use the function. Next, `-a` enables the automatic splitting of the current line into fields in the `@F` array. The splitting happens on the whitespace character by default. The `-l` argument ensures that `print` outputs a newline at the end of each line. Finally, `sum @F` sums all the elements in the `@F` list, and `print` prints the result followed by a newline (which I added with the `-l` argument). (See one-liner 4.2 on page 30 for a more detailed explanation.)

How about finding the date 1299 days ago? Try this:

```
perl -MPOSIX -le  
'@t = localtime; $t[3] -= 1299; print scalar localtime mktime @t'
```

I explain this example in detail in one-liner 4.19 (page 41), but basically you modify the fourth element of the structure returned by `localtime`, which happens to be days. You simply subtract 1299 days from the current day and then reassemble the result into a new time with `localtime mktime @t` and print the result in the scalar context to display human-readable time.

How about generating an eight-letter password? Here you go:

```
perl -le 'print map { ("a".."z")[rand 26] } 1..8'
```

The `"a".."z"` generates a list of letters from *a* to *z* (for a total of 26 letters). Then you randomly choose a letter eight times! (This example is explained in detail in one-liner 5.4 on page 51.)

Or suppose you want to find the decimal number that corresponds to an IP address. You can use `unpack` to find it really quickly:

```
perl -le 'print unpack("N", 127.0.0.1)'
```

This one-liner uses a *v-string*, which is a version literal. V-strings offer a way to compose a string with the specified ordinals. The IP address `127.0.0.1` is treated as a v-string, meaning the numbers 127, 0, 0, 1 are concatenated together into a string of four characters, where the first character has ordinal value 127, the second and third characters have ordinal values 0, and the last character has ordinal value 1. Next, `unpack` unpacks them to a single decimal number in “network” (big-endian) order. (See one-liner 4.27 on page 45 for more.)

What about calculations? Let's find the sum of the numbers in the first column in a table:

```
perl -lane '$sum += $F[0]; END { print $sum }'
```

The lines are automatically split into fields with the `-a` argument, which can be accessed through the `@F` array. The first element of the array, `$F[0]`, is the first column, so you simply sum all the columns with `$sum += $F[0]`. When the Perl program finishes, it executes any code in the `END` block, which, in this case, prints the total sum. Easy!

Now let's find out how many packets have passed through iptables rules:

```
iptables -L -nvx | perl -lane '$pkts += $F[0]; END { print $pkts }'
```

The iptables program outputs the packets in the first column. All you have to do to find out how many packets have passed through the firewall rules is sum the numbers in the first column. Although iptables will output table headers as well, you can safely ignore these because Perl converts them to zero for the `+=` operation.

How about getting a list of all users on the system?

```
perl -a -F: -lne 'print $F[4]' /etc/passwd
```

Combining `-a` with the `-F` argument lets you specify the character where lines should be split, which, by default, is whitespace. Here, you split lines on the colon character, the record separator of `/etc/passwd`. Next, you print the fifth field, `$F[4]`, which contains the user's real name.

If you ever get lost with command-line arguments, remember that Perl comes with a fantastic documentation system called *perldoc*. Type **perldoc perlrun** at the command line. This will display the documentation about how to run Perl and all the command-line arguments. It's very useful when you suddenly forget which command-line argument does what and need to look it up quickly. You may also want to read *perldoc perlvar*, which explains variables; *perldoc perlop*, which explains operators; and *perldoc perlfunc*, which explains functions.

Perl one-liners let you accomplish many tasks quickly. You'll find over 130 one-liners in this book. Read them, try them, and soon enough you'll be the local shell wizard. (Just don't tell your friends—unless you want competition.)

Enjoy!



sample content of Perl One-Liners: 130 Programs That Get Things Done

- [download Mrs. Paine's Garage: and the Murder of John F. Kennedy for free](#)
- [The Gourmet Prescription: High Flavor Recipes for Lower Carbohydrate Diets for free](#)
- [click **Critical Mass \(V. I. Warshawski Series, Book 16\)**](#)
- [download online Caligula: A Biography online](#)
- [The Good Vibrations Guide to Sex book](#)
- [read online Perfect Grilled Vegetables \(Storey's Country Wisdom Bulletin A-152\) here](#)

- <http://www.experienceolvera.co.uk/library/America-in-Vietnam.pdf>
- <http://cavalldecartro.highlandagency.es/library/The-First-Year--Celiac-Disease-and-Living-Gluten-Free--An-Essential-Guide-for-the-Newly-Diagnosed.pdf>
- <http://paulczajak.com/?library/Hegel--Literature--and-the-Problem-of-Agency.pdf>
- <http://test.markblaustein.com/library/Caligula--A-Biography.pdf>
- <http://fortune-touko.com/library/The-Good-Vibrations-Guide-to-Sex.pdf>
- <http://damianfoster.com/books/Tea--A-Global-History--Edible-Series-.pdf>