Quick Guide To Perl

Learn Perl Programming

Through Real Examples

Asim Jalis

Step-by-Step Guide

Learn Perl Programming

Through Real Examples

Asim Jalis

This book is for sale at http://leanpub.com/learnperl

This version was published on 2013-02-18

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Asim Jalis by spreading the word about this book on Twitter!

The suggested hashtag for this book is #learnperl.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#learnperl

To my parents.

Contents

Preface

Who should read this book?

This book targets people who want to learn Perl.

At the end of this book you will be able to:

- Write Perl programs
- Take existing Perl programs and modify them to fit your needs
- Find useful Perl modules on CPAN[1] and use them in your code
- Add Perl to your resumÃ©, and apply for Perl jobs

Why was this book written?

This book evolved out of a course on Perl programming[2] that I teach in the Bay Area. The engagement level in the class peaks when we write real code.

This book draws on the same energy: We are going to learn by writing code.

This book is dedicated to people who know the thrill of automating tedious tasks, who like making the computer do their work for them, who like getting more done by working smarter.

How is this book different from other books?

This book is written in a *pull* style. Each new idea is motivated by a question or a problem. The information then answers the question and solves the problem. This makes the book more like a conversation or a murder mystery than a lecture. This book will interact with you and make you think.

Who is the author?

Asim Jalis has worked as a software developer at Microsoft, Hewlett-Packard, and Salesforce. He is a software consultant and instructor in San Francisco, and a CPAN author[3].

[1]http://search.cpan.org
[2]http://metaprose.com
[3]http://search.cpan.org/~asimjalis

How should this book be used?

This book is highly hands-on and exercise-based. To get the most out of it I recommend firing up your favorite editor and typing out and running all the examples. Learning happens when a person hand-writes the code, makes mistakes, gets stuck, and then experiences the sweet joy of success. Cutting-and-pasting shortcircuits that process.

I recommend typing out each example and then looking at the book for what to pay attention to and for concepts.

Final words

Perl is an indispensable tool for people in IT. Any time you are faced with a mindless repetitive task chances are you can use Perl to automate it. This book will help make Perl an indispensable part of your tool set.

Asim Jalis
February 17, 2013

1 Introduction

1.1 What is Perl

What is Perl?

- Perl is a scripting language.
- Fast edit-compile-run cycle.
- Fast, interactive, programming experience.
- Fast!!—`ack` in Perl is faster than `grep` in C. (You can download `ack` from http://betterthangrep.com.)
- Great for writing quick scripts, that over time evolve into big programs.
- Useful for automating sysadmin tasks, builds, web sites.
- Ships with Mac and Unix machines by default. Easy to install. Everywhere.

1.2 Perl Resources

What are some useful Perl resources?

- Perl CPAN
  http://search.cpan.org
- Tutorials
  http://learn.perl.org/tutorials
- Video Tutorials
  http://showmedo.com/videotutorials/series?name=perlDevijverPerlIntroSeries

1.3 Installing Perl

Exercise: Install Perl.

Solution:

- On Windows install Perl from ActiveState.
  http://www.activestate.com/activeperl/downloads
- On Mac Perl comes preinstalled.
- Verify that you have `perl`.

```
perl -v
```

1.4 Perl Scripts

Exercise: Write a script that prints `Hello, world`.

Solution:

- Save this in a text file called `hello.pl` using an editor like NotePad++, or Sublime Text 2, or TextPad.

```
#!/usr/bin/env perl
use strict;

print "Hello, world.\n";
```

- Type `perl hello.pl` to run it.
- Or type `chmod 755 hello.pl` and then type `./hello.pl` to run it.

Notes:

- Perl is free-form. Whitespace is insignificant.
- Every statement ends with a semicolon.
- Parentheses are optional.
- `print("Hello, world");` also works.

1.5 Perl BAT Files

Exercise: Create a Windows BAT file that prints hello world using Perl.

Solution:

- On Windows save this to `file.bat` and then you can run it from the command line.

```
@echo off
perl.exe -w -x %~f0 %*
goto :EOF
#! -*- perl script start -*-
#line 5

print "Hello, world.\n";
```

- Or you can save Perl in `file.pl` and then run it directly through Perl. This works on all systems.

```
perl file.pl
```

1.6 Perl Documentation

Exercise: Find the documentation for the `print` function.

Solution:

```
perldoc -f print
```

Exercise: Find the documentation for `$_` variable.

Solution:

```
perldoc -v \$_
```

Exercise: Find the FAQ's related to `date`

Solution:

```
perldoc -q date
```

Exercise: Find the list of all Perl man pages.

Solution:

```
perldoc perl
```

Exercise: Find the Perl tutorial for object-oriented programming.

Solution:

```
perldoc perl
perldoc perlboot
```

Who is Larry Wall?

- Larry Wall[1] is the inventor and author of Perl.

What are the three principal virtues of a programmer?

- Laziness
- Impatience
- Hubris

What does PERL stand for?

- *Practical Extraction and Reporting Language.*
- Also, *Pathologically Eclectic Rubbish Lister.*

What does TMTOWTDI stand for?

- *There's More Than One Way To Do It.*

[1]http://upload.wikimedia.org/wikipedia/commons/b/b3/Larry_Wall_YAPC_2007.jpg

2 Perl Data Types

2.1 Scalars

Perl has string and numeric scalars.

```perl
#!/usr/bin/env perl
use strict;

print 1.234, "\n";
print "Hello, world", "\n";
```

Scalars can be assigned to variables.

```perl
#!/usr/bin/env perl
use strict;

my $x = 1234;
my $name = "John Smith";
```

2.2 Numeric Scalars

Exercise: Print one million.

Solution:

```perl
#!/usr/bin/env perl
use strict;

print 1000000, "\n";
print 1000000., "\n";
print 1000000.0, "\n";
print 1e6, "\n";
print 1.0e6, "\n";
print 1_000_000, "\n";
```

Exercise: Check if Perl distinguishes between floating-point numbers and integers.

Solution:

```
#!/usr/bin/env perl
use strict;

print 11 / 3, "\n"
```

2.3 String Scalars

Strings are marked by double-quotes or single-quotes.

```
#!/usr/bin/env perl
use strict;

print "Hello, world.\n";
print 'Hello, world.\n';
```

What's the difference between single-quotes and double-quotes?

- Escape sequences like `\n` only work in double-quotes, not in single-quotes.

What other escape sequences are there?

Sequence	Value
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\b`	Backspace
`\a`	Bell
`\u`	Uppercase next character
`\U`	Uppercase next word
`\l`	Lowercase next character
`\L`	Lowercase next word
`\'`	Single-quote
`\"`	Double-quote
`\\`	Backslash

How can I print `Jim's Garage` in a single-quoted string?

```
#!/usr/bin/env perl
use strict;

print 'Jim\'s Garage', "\n";
print q{Jim's Garage}, "\n";
print q[Jim's Garage], "\n";
print q(Jim's Garage), "\n";
print q|Jim's Garage|, "\n";
```

How can I print `Larry "Tim Toady" Wall` using a double-quoted string?

```
#!/usr/bin/env perl
use strict;

print "Larry \"Tim Toady\" Wall\n";
print qq{Larry "Tim Toady" Wall\n};
print qq[Larry "Tim Toady" Wall\n];
print qq(Larry "Tim Toady" Wall\n);
print qq|Larry "Tim Toady" Wall\n|;
```

2.4 Perl One-Liners

Exercise: Print `Hello, world.` on the command line.

Solution:

- On Unix:

  ```
  perl -e 'print "Hello, world.\n"'
  ```

- On Windows:

  ```
  perl -e "print qq|Hello, world\n|"
  ```

2.5 Scalar Variables

Exercise: Write a program that greets John Smith.

Solution:

```
#!/usr/bin/env perl
use strict;

my $first_name = "John";
my $last_name = "Smith";
print "Hello, $first_name $last_name\n"
```

Notes:

- Variable names are `$` followed by a letter or underscore, followed by letters, underscores, or numbers.
- Perl convention is to use `snake_case` rather than `camelCase` or `PascalCase` for variable names.
- Variables can be *interpolated* into strings as long as you use double-quotes (or `qq`).
- `my` declares the variable. Variable declarations are only checked when `use strict;` is enabled. Leaving out `use strict;` is not recommended.
- The last line does not need a semicolon.

2.6 Arithmetic Operators

Exercise: If Jim earns $10/hour and he worked 35 hours this week, how much is he owed?

Solution:

```
#!/usr/bin/env perl
use strict;

my $hourly_rate = 10.0;
my $hours_worked = 35;
my $amount_owed = $hourly_rate * $hours_worked;
print "Jim is owed $amount_owed.\n";
```

Numeric scalars can be combined using standard arithmetic operations.

Expression	Operation
`$a + $b`	Adding `$a` and `$b`
`$a - $b`	Subtracting `$a` from `$b`
`$a * $b`	Multiplying `$a` and `$b`
`$a / $b`	Dividing `$a` by `$b`
`$a ** $b`	Raising `$a` to the power `$b`
`$a % $b`	Remainder of dividing `$a` by `$b`

2.7 Assignment Operators

Exercise: Suppose Jim is already owed $400 from last week. How much should he be paid?

Solution:

```perl
#!/usr/bin/env perl
use strict;

my $amount_owed = 400;
my $hourly_rate = 10.0;
my $hours_worked = 35;
my $amount_owed += $hourly_rate * $hours_worked;
print "Jim is owed $amount_owed.\n";
```

The operator = assigns the value of the right to the left.

The operator += adds the value of the right to the left.

Assignment	Meaning
`$a = $b`	Save `$b` to `$a`
`$a += $b`	Add `$a` to `$b` and save to `$a`
`$a -= $b`	Subtract `$b` from `$a` and save to `$a`
`$a *= $b`	Multiply `$a` and `$b` and save to `$a`
`$a /= $b`	Divide `$a` by `$b` and save to `$a`
`$a **= $b`	Raise `$a` to the power of `$b` and save to `$a`
`$a %= $b`	Find the remainder of dividing `$a` by `$b` and save to `$a`

2.8 Increment/Decrement Operators

Exercise: Print the numbers 1 through 5.

Solution:

```perl
#!/usr/bin/env perl
use strict;

my $i = 1;
print "i = ", ++$i, "\n";
print "i = ", ++$i, "\n";
print "i = ", ++$i, "\n";
print "i = ", ++$i, "\n";
print "i = ", ++$i, "\n";
```

Exercise: Print the numbers 0 through 4.

Solution:

```perl
#!/usr/bin/env perl
use strict;

my $i = 1;
print "i = ", $i++, "\n";
print "i = ", $i++, "\n";
print "i = ", $i++, "\n";
print "i = ", $i++, "\n";
print "i = ", $i++, "\n";
```

What are the different ways of incrementing the value of `$i`?

```perl
$i = $i + 1;
$i += 1;
$i++;
++$i;
```

What will be the output of these two tests?

```
#!/usr/bin/env perl
use strict;

my $i;
print "Test 1\n";
$i = 0; print $i++, "\n" ; print $i, "\n" ;
print "Test 2\n";
$i = 0; print ++$i, "\n" ; print $i, "\n" ;
```

Notes:

- `$i++` increments `$i` after returning its value.
- `++$i` increments `$i` and then returns its value.

Exercise: Determine the output of these two tests.

Solution:

```
#!/usr/bin/env perl
use strict;

my $i;
print "Test 1\n";
$i = 0; print $i--, "\n" ; print $i, "\n" ;
print "Test 2\n";
$i = 0; print --$i, "\n" ; print $i, "\n" ;
```

Hints:

- `$i--` decrements `$i` after returning its value.
- `--$i` decrements `$i` and then returns its value.

2.9 String Operators

Exercise: Write `Happy Birthday, Dmitri!` and draw 25 candles.

Solution:

```
#!/usr/bin/env perl
use strict;

my $name = 'Dmitri';
my $age = 29;
my $candle_flames = '.' x $age;
my $candle_sticks = '|' x $age;

print "Happy Birthday, " . $name . "!\n";
print "$candle_flames\n";
print "$candle_sticks\n";
```

Notes:

- The `.` operator concatenates strings.
- The `x` operator multiplies strings.
- `'-' x 79` is useful for printing lines.
- They do no modify the values of their arguments. Only assignment modifies values.
- *Variable interpolation* replaces `${name}` inside the string with its value. The braces are optional.

Exercise: What will be the output of this program?

Solution:

```
#!/usr/bin/env perl
use strict;

my $x = 'hello';
$x .= ' world';
print $x, "\n";
```

Exercise: What will be the output of this program?

Solution:

```
#!/usr/bin/env perl
use strict;

my $x = 'a';
$x x= 10;
print $x, "\n";
```

2.10 Unary Operators

Exercise: Fix the capitalization of `"MORT"` and tell him his balance is `-25`.

Solution:

```
#!/usr/bin/env perl
use strict;

my $name = "MORT";
$name = ucfirst(lc($name));
my $balance = -25;
print "Hello ${name}.\n";
print "Your balance is ${balance}!\n";
```

Unary operators do not modify the value of their argument.

Operator	Result
`uc($a)`	Uppercase `$a`
`ucfirst($a)`	Uppercase first letter of `$a`
`lc($a)`	Lowercase `$a`
`lcfirst($a)`	Lowercase first letter of `$a`
`int($a)`	Integer part of `$a`
`length($a)`	Length of `$a` as a string
`cos($a)`	Cosine of `$a`
`rand($a)`	Random decimal number from `0` to less than `$a`

2.11 Reading Input

Exercise: Prompt the user for name, cost per gallon of gas and gallons of gas, and tell them the total amount they owe.

Solution:

```
#!/usr/bin/env perl
use strict;

print "Name: ";
my $name = <STDIN>;
chomp $name;
print "Price per gallon: ";
my $price_per_gallon = <STDIN>;
print "Gallons: ";
my $gallons = <>;
my $total_amount = $price_per_gallon * $gallons;
print "Hello ${name}!\n";
print "Total amount due: \$${total_amount}\n";
```

Notes:

- `<STDIN>` waits until the user enters a newline and returns the line.
- `<>` is an abbreviation for `<STDIN>`.
- `chomp` removes the newline from the end of the input.
- Unlike other unary operators `chomp` has an effect on its input.
- The $ character in `\$${total_amount}` is escaped.

2.12 Population Calculator

Exercise: Estimate the population of California in 2021. The population in 2012 was 38,000,000 and the growth rate was 1%. The population can be estimated as $P = P*0 (1 + R)ˆT$, where T is the number of years since 2012, $P*0$ is the population in 2012, R is the population growth rate.

Solution:

```
#!/usr/bin/env perl
use strict;

my $pop_initial = 38_000_000;
my $year_initial = 2012;
my $year_final = 2021;
my $year_count = $year_final - $year_initial;
my $pop_rate = 0.01;
my $pop_final =
    $pop_initial * ((1 + $pop_rate) ** $year_count);
print "The population of California " .
    "in $year_final will be $pop_final.\n";
```

Exercise: Are the parentheses around `(1 + $pop_rate) ** $year_count` required?

Solution:

- Look at `perldoc perop` and look up operator precedence to verify they are not required.

It's safer to use parentheses and not guess precendence if it's not clear.

Associativity	**Precedence**
nonassoc	`++ --`
right	`**`
left	`* / % x`
left	`+ - .`
left	`<< >>`
nonassoc	`named unary operators`
nonassoc	`< > <= >= lt gt le ge`
nonassoc	`== != <=> eq ne cmp ~~`
left	`&`
left	`^`
left	`&&`
right	`?:`
right	`= += -= *= etc.`
left	`, =>`
nonassoc	`list operators (rightward)`
right	`not`
left	`and`
left	`or xor`

3 Control Flow

3.1 Using `if`, `elsif`, `else`

Exercise: Write a program that calculates a letter grade based on a student's GPA.

Grade	GPA Minimum
A	3.5
B	2.5
C	1.5
D	1.0
F	0.0

Solution:

```perl
#!/usr/bin/env perl
use strict;

print "Enter GPA: ";
my $gpa = <>;
my $grade = "";
if ($gpa >= 3.5) {
    $grade = "A";
}
elsif ($gpa >= 2.5) {
    $grade = "B";
}
elsif ($gpa >= 1.5) {
    $grade = "C";
}
elsif ($gpa >= 1.0) {
    $grade = "D";
}
else {
    $grade = "F";
}
print "Final Grade: $grade\n";
```

Notes:

- The `if/elsif/else` construct represents a fork in the road.

- The program evaluates each condition and takes the first branch whose condition is true.
- If no condition is true then it takes the `else` clause.
- The `elsif` and the `else` are optional.
- So if no condition is true, and there is no `else` clause then it ignores the whole construct.

Why did we have to declare `$grade` before the `if`?

- Each block marked by curly braces defines a new *scope*. A variable defined in a block with `my` disappears when the program exits the block.

3.2 Relational Operators

Here is a list of the numeric relational operators that can be used to compare numbers.

Numeric Relations	**Is True When**
`$a < $b`	`$a` is less than `$b`
`$a <= $b`	`$a` is less than or equal to `$b`
`$a > $b`	`$a` is great than `$b`
`$a >= $b`	`$a` is great than or equal to `$b`
`$a == $b`	`$a` is equal to `$b`
`$a != $b`	`$a` is not equal to `$b`

Here is a list of the string relational operators that can be used to compare strings.

String Relations	**Is True When**
`$a eq $b`	`$a` is equal to `$b`
`$a ne $b`	`$a` is not equal to `$b`
`$a lt $b`	`$a` is less than `$b`
`$a le $b`	`$a` is less than or equal to `$b`
`$a gt $b`	`$a` is greater than `$b`
`$a ge $b`	`$a` is greater than or equal to `$b`

With string relational operators "less than" implies earlier in dictionary order where the letters are ordered using their ASCII codes. So `a` is less than `b` and `A` is less than `a`.

3.3 Ternary Operator: `$a ? $b : $c`

`$a ? $b : $c`. If `$a` is true, this evaluates to `$b`. Otherwise it evaluates to `$c`.

Exercise: Set `$x` to `0` or `1`. Print `"True"` if it is true, and `"False"` if it is false.

Solution:

```
#!/usr/bin/env perl
use strict;

my $input = 1;
my $message = $input ? "x is True" : "x is False";
print "$message\n";
```

3.4 Falseness

Which values of scalars are false and which are true?

- `0`, `""`, `"0"` are false.
- All representations of `0` are false. For example, `0.0`, `0.0e+100`, `00.00`, `000_000.000`
- All other scalars are true including `"Hello"`, `" "`, `"00"`.
- All string representations of `0` are false except `"0"`. For example `"0.0"`, `"00"`, `"0.0e+100"`.

Exercise: What will be the output of this program?

Solution:

```
#!/usr/bin/env perl
use strict;

my $a;

$a = 0;         print $a ? "True\n" : "False\n";
$a = 0.0;       print $a ? "True\n" : "False\n";
$a = "";        print $a ? "True\n" : "False\n";
$a = "0";       print $a ? "True\n" : "False\n";
$a = 1;         print $a ? "True\n" : "False\n";
$a = -1;        print $a ? "True\n" : "False\n";
$a = +1;        print $a ? "True\n" : "False\n";
$a = 2;         print $a ? "True\n" : "False\n";
$a = " ";       print $a ? "True\n" : "False\n";
$a = "a";       print $a ? "True\n" : "False\n";
$a = "true";    print $a ? "True\n" : "False\n";
$a = "false";   print $a ? "True\n" : "False\n";
```

3.5 Logical Operators

Boolean values of true and false can be combined using logical operators.

Operator	Alternative	Is True When
`$a && $b`	`$a and $b`	`$a` is true and `$b` is true
`$a || $b`	`$a or $b`	`$a` is true or `$b` is true
`!$a`	`not $a`	`$a` is false

These operators short-circuit—the evaluation stops as soon as the outcome is clear.

Exercise: Write a program that greets user by name and uses the default name of `"stranger"` if the user doesn't provide one.

Solution:

```
#!/usr/bin/env perl
use strict;

my $name = <>;
chomp $name;
$name ||= "stranger";
print "Hello, $name.\n";
```

The `||=` idiom is frequently used for default values. Consider `$name = $name || "stranger"`. If `$name` is not blank, it's true, so the `||` short-circuits and `$name` is set to `$name`. Otherwise it is set to `"stranger"`.

3.6 Short Form `if`

Exercise: Print "You are the millionth customer!" for the millionth customer.

Solution:

```
#!/usr/bin/env perl
use strict;

my $customer = 1_000_000;
print "You are the millionth customer!" if $customer == 1_000_000;
```

When the `if` block has only one statement you can put the `if` and the condition after the statement.

3.7 Looping with `while` and `for`

Exercise: Pick a number from 1 to 10 and then prompt the user to guess it.

Solution:

```perl
#!/usr/bin/env perl
use strict;

my $number = int(rand(10)) + 1;
my $guess = -1;
print "I picked a number.\n";
while ($number != $guess) {
    print "Guess what number I picked: ";
    $guess = <>;
    chomp $guess;
    if ($number == $guess) {
        print "Yes, you guessed right.\n";
    }
    else {
        print "You were wrong. Try again.\n";
    }
}
```

The program keeps running the code in the `while` block until the condition becomes false.

Exercise: Rewrite the program as an infinite loop. Terminate using `last`.

Solution:

```perl
#!/usr/bin/env perl
use strict;

my $number = int(rand(10)) + 1;
my $guess = -1;
print "I picked a number.\n";
while (1) {
    print "Guess what number I picked: ";
    $guess = <>;
    chomp $guess;
    if ($number == $guess) {
        print "Yes, you guessed right.\n";
        last;
    }
    else {
        print "You were wrong. Try again.\n";
        next;
    }
}
```

Use `last` to break out of a loop early.

Use `next` to start from the start of the block again.

Exercise: Print the squares of the numbers from 0 through 19.

Solution:

```perl
#!/usr/bin/env perl
use strict;

my $i = 0;
while ($i < 20) {
    my $square = $i ** 2;
    print "square of $i = $square\n";
    $i++;
}
```

Loops for going through sequences of numbers are so commonly used that there is a special construct for it: the `for` loop.

```perl
#!/usr/bin/env perl
use strict;

for (my $i = 0 ; $i < 20 ; $i++) {
    my $square = $i ** 2;
    print "square of $i = $square\n";
}
```

Exercise: Write a program that finds all the numbers from 0 to 100 that have an integer square root.

Exercise: Write a program that repeatedly picks a number and then asks a user to guess it. The program terminates when you user types `q`.