Perl basics: a concise guide

Version 8
October 6, 2006

The latest version of this guide is available for download at http://hoffmancommapaul.com/perl/guide/.

Note

Notations such as this indicate a reference to one or more sections of the standard Perl documentation:

- ☛ **perldata, perlsyn**

To see the documentation for a section, use its name (e.g., “perldata”) as an argument to the **perldoc** command at the Unix or DOS prompt (e.g., ”perldoc perldata”) or look it up by name in the online Perl documentation at http://perldoc.perl.org/.

Scalar values and variables

☛ **perlintro, perldata, perlsyn, perlop**

Declare a scalar variable (a string or a number)

```
my $name;
my $age;
my ($allowance, $favorite_vegetable);
```

Assign a value to a scalar variable

```
$name = 'Yolanda';
$age = 3;
($allowance, $favorite_vegetable)
    = ('$1/week', 'rutabagas');
```

Declare and assign at the same time

```
my $name = 'Yolanda';
my $age = 3;
my ($allowance, $favorite_vegetable)
    = ('$1/week', 'rutabagas');
```

Non-scalar values and variables

Arrays

```
# An array is a list of values
my @pets = ('Lucky', 'Fido', 'Genghis');
my @lucky = (7, 23, -9041.618);
# Members of arrays are always scalars
my $first_lucky_number = $lucky[0];
$lucky[1] = $lucky[0] + 1;
```

Hashes

```
# A hash is a list of (key => value) pairs
my %email = (
    'Yolanda' => 'yoyo@mama.com',
    'Dr. Smith' => 'ima@doctor.com',
    'Paul' => 'paul@hoffmancommapaul.com'
);
# Members of hashes are always scalars
my $from_address = $email{'Paul'};
$email{'Ulysses'} = 'hotbody8273@aol.com';
delete $email{'Dr. Smith'};
```

Quotes

Variables are interpolated in double quotes

```
my $name = 'Paul';
my $greeting = "Hi, my name is $name.";
# Same result:
my $greeting = "Hi, my name is Paul."
```

This includes members of arrays

```
my @friends = ('Sam', 'Xerxes');
print "My best friend is $friends[0].\n";
print "My 2nd best friend is $friends[1].\n";
```

And members of hashes

```
my %age = ('Sam' => 2, 'Xerxes' => 117);
print "Sam is $age{'Sam'} years old.\n";
```

Variables are **not** *interpolated in single quotes*

```
my $variable_name = '$name';
```

Variables are interpolated **only once**

```
print "$variable_name\n";
# Same result:
print "'$name'\n";
# Also the same:
print '$name', "\n";
```

Character escapes inside double quotes ("...")

`\n` *newline*
`\t` *tab*
`\"` *double quote*
`\$` *dollar sign*
`\@` *at sign*
`\\` *backslash*

Character escapes inside single quotes ('...')

`\'` *single quote*
`\\` *backslash*

Other ways of quoting

`qq{Hi, $name\n}` *same as* `"Hi, $name\n"`
`q($3 per dozen)` *same as* `'$3 per dozen'`
`qw/1 2 three/` *same as* `('1', '2', 'three')`
(Other delimiters besides `{}`, `()`, *and / / may be used.)*

References

☛ **perlref, perlreftut**

Reference to a named array

```
my $array_ref = \@array;
foreach my $x (@$array_ref) { ... }
```

Reference to an anonymous array

```
my $array_ref = [ @array ];
foreach my $x (@$array_ref) { ... }
```

Reference to a named hash

```
my $hash_ref = \%hash;
while (my ($k, $v) = each %$hash_ref)) { ... }
```

Reference to an anonymous hash

```
my $hash_ref = { %hash };
while (my ($k, $v) = each %$hash_ref)) { ... }
```

Simple math

☛ **perlop**

Math operators

`+, -, *, /` *addition, subtraction, multiplication, division*

`%` *remainder (e.g., 7 % 2 == 1)*

`**` *exponentiation (raise a number to the power of another number)*

Math functions

`abs(-10)` *absolute value (here, 10)*

`int(3.14159)` *integer part of a number (e.g., 3)*

`rand()` *random number ≥ 0 and < 1 (e.g., 0.884252106628)*

`int(rand(10))` *random integer ≥ 0 and < 10 (e.g., 2)*

Fancy math operators

`$x += 12` *add 12 to $x*

`$x -= $y` *subtract $y from $x*

`$x *= 10` *multiply $x by 10*

etc.

`$x++, $x--` *add or subtract 1 from $x*

Input and output

☛ **perlintro, perlfunc**

Read a line from standard input (typically, the keyboard)

```
my $line = <STDIN>;
```

Write a line to standard output (typically, the display)

```
print STDOUT "Hello, human!\n";
```

Write to the default handle (normally `STDOUT`*)*

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

Open a file for reading

```
my $file = 'people.txt';
my $handle;
open $handle, '<', $file
    or die "Couldn't open $file: $!"
```

Read a line from an open file handle

```
my $line = <$handle>;
```

Read a line and remove the trailing newline character

```
my $line = <$handle>;
chomp($line);
```

Print the contents of a file

```
while (defined(my $line = <$handle>)) {
    print $line;
}
```

Shorter version, using implicit variable $_

```
while (<$handle>) {
    print;
}
```

Copy lines from files listed on the command line **or** *std. input*

```
while (<>) {
    print;
}
```

Open a file for writing

```
my $file = 'people.txt';
my $handle;
open $handle, '>', $file
    or die "Couldn't open $file: $!"
```

Write to an open file handle

```
print $handle "Hello, file!\n";
```

Comparisons and logical operations

☛ **perlop**

Compare numbers

`$x == 1` *and its opposite* `$x != 1`
`$x < 21` *and its opposite* `$x >= 21`
`$x > $y` *and its opposite* `$x <= $y`
`$a <=> $b` *-1, 0, or 1 if $a < $b, $a == $b, or $a > $b*

Compare strings

`$name eq 'Bob'` *and its opposite* `$name ne 'Bob'`
`$name lt 'M'` *and its opposite* `$name ge 'M'`
`$name gt 'H'` *and its opposite* `$name le 'H'`
`$a cmp $b` *–1, 0, or 1 if $a lt $b, $a eq $b, or $a gt $b*

Check to see if a value is defined

```
if (defined($x)) { ... }
```

Boolean operators

`&&` *and*
`||` *or*
`!` *not*

Alternate Boolean operators (lower precendence)

`and` *and*
`or` *or*
`not` *not*

Loops and conditionals

☛ **perlintro, perlsyn**

Do something if a condition is true

```
if ($age < 21) {
    print "No beer for you!\n";
}
```

Shorter version

```
print "No beer for you!\n"
    if $age < 21;
```

Keep doing something while a condition is true

```
my $answer = <STDIN>;
chomp $answer;
while ($answer eq 'no') {
    print "I won't take no for an answer.\n";
    $answer = <STDIN>;
}
```

A more complicated example

```
my $num_rabbits = 2;
my $phi = (sqrt(5) + 1) / 2;
while ($num_rabbits < 1000) {
    print "There are $num_rabbits rabbits.\n";
    $num_rabbits =
        int($num_rabbits * $phi + 0.5);
}
```

Loop over the members of an array

```
my @numbers = (1..10);  # 1, 2, 3, etc.
foreach my $n (@numbers) {
    print "$n potato\n";
}
```

Loop over the members of a hash

```
my %ital = (1 => 'uno', 2 => 'due');
foreach my $n (keys %ital) {
    print "The word for $n is $ital{$n}.\n";
}
```

Another way to do the same thing

```
while (my ($n, $word) = each %ital) {
    print "The word for $n is $word.\n";
}
```

Loop over all matches in a string

```
while ($rec =~ /(\d{4}-\d{4}[\dXx])/g) {
    print "Found an ISSN: $1\n";
}
```

Infinite loop

```
while (1) {
    print "I will not loop infinitely.\n";
}
```

Stop before the loop condition is met

```
while (1) {
    print "Should I stop now? ";
    last if <STDIN> =~ /^y|yes$/i;
}
foreach my $n (1..999) {
    last if int(rand(10)) == 7;
    print "$n\n";
}
```

Regexes (patterns)

☛ **perlre, perlrequick, perlretut**

Match a pattern anywhere in a string

```
if ($string =~ m/xxx/) {
    print "There's an xxx in there!\n";
}
```

The `m` *may be omitted (and usually is)*

```
$string =~ /xxx/
```

Case-insensitive match

```
$string =~ /xxx/i
```

Replace the first occurrence of `xxx` *in a string with* `yyy`

```
if ($string =~ s/xxx/yyy/) {
    print "I changed xxx to yyy.\n";
}
```

Replace all occurrences of `xxx` *in a string with* `yyy`

```
$string =~ s/xxx/yyy/g;
```

Match in `$_` *(the implicit variable)*

```
/xxx/
```

Make substitutions in `$_` *(the implicit variable)*

```
s/xxx/yyy/
s/xxx/yyy/g
s/xxx/yyy/i
s/xxx/yyy/ig
```

Special characters

`\n` *matches a newline character*

`.` *matches any character* **except** *a character that matches* `\n`

`\d` *matches a digit (0 to 9)*

`\D` *matches any character* **except** *a character that matches* `\d`

`\s` *matches a space or tab character (or other whitespace)*

`\S` *matches any character* **except** *a character that matches* `\s`

`\w` *matches a 'word' character (A-Z, a-z, 0-9, or _)*

`\W` *matches any character* **except** *a character that matches* `\w`

Anchor points (match at character boundaries)

`^` *matches before the first character in a string*

`$` *matches before* `\n` *or after the last character of a string*

`\b` *matches at a word boundary*

Other regex syntax

`X+` *matches one or more Xs in a row*

`X*` *matches zero or more Xs in a row*

`X?` *matches zero or one Xs (i.e., an optional X)*

`X{4}` *matches exactly 4 Xs*

`X{2,5}` *matches 2 to 5 Xs*

`X{3,}` *matches 3 or more Xs*

Character classes

`[abc]` *matches the character a, b, or c*

`[a-z]` *matches any lowercase letter*

`[A-Za-z]` *matches any letter*

`[^A-Za-z]` *matches any* **non-***letter*

`[\d\s]` *matches any digit or whitespace character*

`[^\d\s]` *matches anything* **except** *a digit or whitespace character*

`[abc]*` *matches any number of a's, b's, and c's in a row*

etc.

Parens capture what was matched and put it in `$1`, `$2`, *etc.*

```
if ($name =~ m/^([XYZ])/) {
    print "Cool! A name beginning with $1!\n";
}
```

Capture matches one at a time in a loop

```
my $total = 0;
while ($string =~ m/(\d+)/g) {
    $total += $1;
}
print "Total: $total\n";
```

Capture matches all at once using the `g` *modifier*

```
my @numbers = ($string =~ /(\d+)/g);
```

Variables are interpolated when replacing

```
# Make sure the ISSN has a hyphen
$issn =~ /^(\d\d\d\d)-?(\d\d\d[\dXx])/$1-$2/;
```

Look for a pattern at the beginning of a string

```
if ($greeting =~ m/^Hello|Hi|Howdy/) {
    print "Hi there!\n";
}
```

Look for a pattern at the end of a string

```
if ($greeting =~ m/!!+$/) {
    # Two or more exclamation points
    print "Please don't shout.\n";
}
```

Match an entire line of text against a set of alternatives

```
if ($command =~ m/^quit|exit|done$/) {
    print "Thanks for playing.\n";
}
```

When case doesn't matter

```
if ($command =~ m/^please/i) {
    print "OK.\n"
}
```

Negative matching

```
if ($command !~ m/^please/i) {
    print "You didn't say the magic word.\n";
}
```

Summary of regex modifiers

- `g` *match (or replace) all occurrences, not just the first one*
- `i` *don't care about case (upper or lower)*
- `x` *comments and whitespace in regex are ignored*
- `e` *evaluate substitution string as a Perl expression*

Subroutines and blocks

☛ **perlsub**

Define a subroutine with parameters

```
sub greet {
    my ($name) = @_;
    print "Hello, $name.\n";
}
```

Call a subroutine with arguments

```
greet('Zainab');
```

Define a subroutine that returns a value

```
sub times_three {
    my ($number) = @_;
    return $number * 3;
}
print "6 times 3 = ", times_three(6), "\n";
```

A variable is invisible to anything outside its block

```
my $x = 123;
{
    my $y = 456;
}
print "$y\n";  # ERROR! $y not declared
```

A variable is **not** *invisible to blocks within its block*

```
my $x = 123;
{
    print "$x\n";  # OK
}
```

The three principle uses for hashes

Collect related attributes into a single variable

```
my %friend = (
    'name' => 'Ulysses K. Fishwick',
    'age'  => 93,
    'favorite_color' => blue,
    # More attributes here...
);
```

"Attach" information to names, IDs, etc.

```
my %ages = (
    'Xerxes' => 108,
    'Yolanda' => 3,
    'Zainab' => 57
);
```

Keep sets of things

```
my %fruit = (
    'apple' => 1,
    'orange' => 1,
    'banana' => 1,
    'kiwi' => 1,
    # etc.
);
my @things = ('egg', 'apple', 'shovel');
# Which of these is a fruit?
foreach my $thing (@things) {
    print "$thing\n" if $fruit{$thing};
}
```

The good, the bad, and the ugly

BAD: *Regexes that look like monkeys typed them*

```
$x =~ s/^(\d(-?\d+){2}(-?[\dXx]))$/norm($1)/e;
```

GOOD: *Regexes that use the x modifier, whitespace, & comments*

```
$x =~
s/^
    (
      \d           # Country code
      (-?\d+){2}   # Publisher-specific
      (-?[\dXx])   # Check digit
    )
$/
    norm($1);      # Normalize it
/ex;
```

BAD: *Repetitive, duplicative code*

```
print "Name: ";
my $name = <STDIN>; chomp $name;
die "No name" unless defined $name;
print "Age: ";
my $age = <STDIN>; chomp $age;
die "No name" unless defined $name;
```

GOOD: *Isolate specific, well-defined functionality in subroutines*

```
sub ask {
    my ($attribute) = @_;
    print ucfirst($attribute), ': ';
    my $val = <STDIN>;
    die "No $attribute" if !defined($val);
    chomp $value;
    return $value;
}
my $name = ask("name");
my $age = ask("age");
```

Random wisdom

Always let the Perl interpreter help with simple mistakes

```
use warnings;
use strict;
```

Declare variables when you use them (or perhaps sooner)

```
print "Name: ";
my $name = <STDIN>;
print "Hello, $name.\n";
print "Age: ";
my $age = <STDIN>;
```

Add comments as you write your code

```
# Collect ISBNs from the input
my @isbns;
while (<>) {
    # Find ISBNs without hyphens
    while (/(\d{9}[\dXx])/g) {
        # Normalize x to upper-case
        my $isbn = uc($1);
        push @isbns, $isbn;
    }
}
# Print ISBNs in ascending order
foreach my $isbn (sort @isbns) {
    print "$isbn\n";
}
```

Important variables

☛ **perldoc perlvar**

`$_` *the implicit variable: e.g.,* `foreach (@foo)` **or** `while (<>)`

`@_` *arguments passed to a subroutine*

`$1` *(etc.) captured subpatterns*

`@ARGV` *parameters provided on the command line*

`$!` *error string (set by* `open`, `close`*, etc.)*

Where to get help

Learn Perl
`http://learn.perl.org/`

perldoc
`http://perldoc.perl.org/`

Perl For Libraries (mailing list)
`http://perl4lib.perl.org/`

PerlMonks
`http://www.perlmonks.org/`

The Comprehensive Perl Archive Network (CPAN)
`http://search.cpan.org/`

Perl modules for handling MARC records and files
`http://search.cpan.org/dist/MARC-Record/`
`http://search.cpan.org/dist/MARC-Lint/`
`http://search.cpan.org/dist/MARC-XML-0.83/`

Perl modules for dealing with XML (just a selected few!)
`http://search.cpan.org/dist/XML-Parser/`
`http://search.cpan.org/dist/XML-Twig/`
`http://search.cpan.org/dist/XML-SAX/`

Perl news and articles
`http://www.perl.com/`

Books

Learning Perl, 4th ed. (ISBN 0-596-10105-8)
Programming Perl, 3rd ed. (ISBN 0-596-00027-8)
Perl Pocket Reference, 4th ed. (ISBN 0-596-00374-9)
Object Oriented Perl (ISBN 1884777791)
Programming the Perl DBI (ISBN 1-56592-699-4)
CGI Programming with Perl, 2nd ed. (ISBN 1-56592-419-3)

Miscellaneous

What is Perl?

Practical Extraction and Reporting Language
or Pathologically Eclectic Rubbish Lister

What is perl?

The program ("the Perl interpreter") that runs programs written in Perl.

What is PERL?

A misspelling you use if you want to be descended upon by a horde of angry Perl programmers.

Why all the crazy punctuation?

Because Larry Wall has a background in linguistics? Or maybe he's just crazy. (There may be a correlation here…)