

# Perl®

## Notes for Professionals

### Chapter 7: Control Statements

#### Section 7.1: Conditionals

Perl supports many kinds of conditional statements (statements that are based on boolean results). Common conditional statements are `if`-`else`, `unless`, and `ternary` statements. Given statements switch-like constructs from C-derived languages, and are available in versions Perl 5.10 and above.

#### if-else Statements

The basic structure of an `if`-statement is like this:

```
if (EXPR) BLOCK
else BLOCK
if (EXPR) BLOCK elsif (EXPR) BLOCK ... else BLOCK
if (EXPR) BLOCK elsif (EXPR) BLOCK ...
```

For simple `if`-statements, the `if` can precede or succeed the code to be executed.

```
$number = 7;
if ($number > 4) { print "Number is greater than four!"; }
```

```
# Can also be written this way
print "Number is greater than four!" if $number > 4;
```

#### Section 7.2: Loops

Perl supports many kinds of loop constructs: `foreach`, `while`, `do-while`, and `until`.

```
$numbers = 1..10;
for (my $i=0; $i < $numbers; $i++) {
    print "$numbers[$i]\n";
}
```

Also used to iterate an array with `my $num = $numbers`;

```
print "$num\n";
}
```

The `while` loop evaluates the conditional before executing the associated block. For example, the following code would never be executed if the file is empty, or it was already exhausted before the conditional.

```
while (my $line = readline $fh) {
    my $line;
}
```

The `do-while` and `do-until` loops, on the other hand, evaluate the conditional after the block. So, a `do-while` or a `do-until` loop is always executed at least once.

```
my $greeting_count = 0;
do {
    say "hello!";
    $greeting_count++;
} until { $greeting_count > 10 }
```

Perl Notes for Professionals

### Chapter 10: Lists

#### Section 10.1: Array as list

The array is one of Perl's basic variable types. It contains a list, which is an ordered sequence of zero or more scalars. The array is the variable holding (and providing access to) the list data, as is documented in `perlarray`.

You can assign a list to an array:

```
my @foo = (4, 5, 6);
```

You can use an array wherever a list is expected:

```
join '-', (4, 5, 6);
```

```
join '-', @foo;
```

Some operators only work with arrays since they mutate the list an array contains:

```
shift @array;
```

```
unshift @array, (1, 2, 3);
```

```
pop @array;
```

```
push @array, (7, 8, 9);
```

#### Section 10.2: Assigning a list to a hash

Lists can also be assigned to hash variables. When creating a list that will be assigned to a hash variable, it is recommended to use the `fat comma` between keys and values to show their relationship:

```
my %hash = ( foo => 42, bar => 42, baz => 44 );
```

The `=>` is really only a special comma that automatically quotes the operand to its left. So, you could use normal commas, but the relationship is not as clear:

```
my %hash = ( 'foo', 42, 'bar', 42, 'baz', 44 );
```

You can also use quoted strings for the left-hand operand of the fat comma `=>`, which is especially useful for keys containing spaces.

```
my %hash = ( 'foo bar' => 42, 'baz qux' => 44 );
```

For details see `Comma operator` at `perlfaq`: `perlfaq`

#### Section 10.3: Lists can be passed into subroutines

As to pass list into a subroutine, you specify the subroutine's name and then supply the list to it:

```
test_subroutine( 'steve', 'lterd' );
```

```
test_subroutine( 'steve', 'lterd' ); # new
```

Internally Perl makes aliases to those arguments and put them into the array `$_`, which is available within subroutines:

```
$_ = 'steve', 'lterd'; # Done internally by perl
```

Perl Notes for Professionals

### Chapter 13: Reading a file's content into a variable

#### Section 13.1: Path::Tiny

Using the idiom from *The Manual Way* several lines in a script soon gets tedious so you might want to try a module.

```
use Path::Tiny;
```

```
my $content = path($filename)->slurp;
```

You can pass a `binmode` option if you need control over file encodings, line endings etc.: see `perlbin` module.

#### Section 13.2: The manual way

```
open my $fh, "<:", $filename
```

```
or die "Could not open '$filename' for reading: $!";
```

```
my $content = do { local $?; <$fh> };
```

After opening the file `read` can be used to read specific file encodings instead of raw bytes, the trick is the `binmode` call. The `binmode` call handles in a diamond operator, returns a single record from the file. The "input record separator" variable `$/` specifies what a "record" is—by default it is set to a newline character so "a record" means "a single line". As `$/` is a global variable, `local` does two things: it creates a temporary local copy of `$/` that will vanish at the end of the block, and gives it the `binmode` value. `local` also gives to uninitialized variables.

When the input record separator has that `binmode` value, the diamond operator will return the entire file (it considers the entire file to be a single line).

Using `do` you can even get around manually opening a file. For repeated reading of files.

```
sub readFile { do { local $ARGV; $? = <$_> } }
```

```
my $content = readFile($filename);
```

Can be used. Here, another global variable `ARGV` is localized to simulate the same process used when passing a diamond operator with parameters. `$?` is still used, since the array in front of a `<` gets all incoming arguments. Next, the diamond operator `<` again delivers one record defined by `$/` (the whole file) and returns from the `do` block, which in turn returns from the sub.

The sub has no explicit error handling, which is bad practice! If an error occurs while reading the file, you will receive `undef` as return value, as opposed to an empty string from an empty file.

Another disadvantage of the last code is the fact that you cannot use `PerlIO` for different file encodings—you always get raw bytes.

#### Section 13.3: File::Slurp

Don't use it. Although it has been around for a long time and is still the module most programmers will suggest, it is outdated and not likely to be fixed.

Perl Notes for Professionals

90+ pages  
of professional hints and tricks

# Contents

<b>About</b>	1
<b>Chapter 1: Getting started with Perl Language</b>	2
Section 1.1: Getting started with Perl	2
<b>Chapter 2: Comments</b>	4
Section 2.1: Single-line comments	4
Section 2.2: Multi-line comments	4
<b>Chapter 3: Variables</b>	5
Section 3.1: Scalars	5
Section 3.2: Array References	5
Section 3.3: Scalar References	6
Section 3.4: Arrays	7
Section 3.5: Typeglobs, typeglob refs, filehandles and constants	8
Section 3.6: Sigils	9
Section 3.7: Hash References	11
Section 3.8: Hashes	12
<b>Chapter 4: Interpolation in Perl</b>	15
Section 4.1: What is interpolated	15
Section 4.2: Basic interpolation	16
<b>Chapter 5: True and false</b>	18
Section 5.1: List of true and false values	18
<b>Chapter 6: Dates and Time</b>	19
Section 6.1: Date formatting	19
Section 6.2: Create new DateTime	19
Section 6.3: Working with elements of datetime	19
Section 6.4: Calculate code execution time	20
<b>Chapter 7: Control Statements</b>	21
Section 7.1: Conditionals	21
Section 7.2: Loops	21
<b>Chapter 8: Subroutines</b>	23
Section 8.1: Creating subroutines	23
Section 8.2: Subroutines	24
Section 8.3: Subroutine arguments are passed by reference (except those in signatures)	25
<b>Chapter 9: Debug Output</b>	27
Section 9.1: Dumping with Style	27
Section 9.2: Dumping data-structures	28
Section 9.3: Data::Show	28
Section 9.4: Dumping array list	29
<b>Chapter 10: Lists</b>	31
Section 10.1: Array as list	31
Section 10.2: Assigning a list to a hash	31
Section 10.3: Lists can be passed into subroutines	31
Section 10.4: Return list from subroutine	32
Section 10.5: Hash as list	33
Section 10.6: Using arrayref to pass array to sub	33
<b>Chapter 11: Sorting</b>	34
Section 11.1: Basic Lexical Sort	34

Section 11.2: The Schwartzian Transform .....	34
Section 11.3: Case Insensitive Sort .....	35
Section 11.4: Numeric Sort .....	35
Section 11.5: Reverse Sort .....	35
<b>Chapter 12: File I/O (reading and writing files) .....</b>	<b>36</b>
Section 12.1: Opening A FileHandle for Reading .....	36
Section 12.2: Reading from a file .....	36
Section 12.3: Write to a file .....	37
Section 12.4: "use autodie" and you won't need to check file open/close failures .....	37
Section 12.5: Rewind a filehandle .....	38
Section 12.6: Reading and Writing gzip compressed files .....	38
Section 12.7: Setting the default Encoding for IO .....	39
<b>Chapter 13: Reading a file's content into a variable .....</b>	<b>40</b>
Section 13.1: Path::Tiny .....	40
Section 13.2: The manual way .....	40
Section 13.3: File::Slurp .....	40
Section 13.4: File::Slurper .....	41
Section 13.5: Slurping a file into an array variable .....	41
Section 13.6: Slurp file in one-liner .....	41
<b>Chapter 14: Strings and quoting methods .....</b>	<b>42</b>
Section 14.1: String Literal Quoting .....	42
Section 14.2: Double-quoting .....	42
Section 14.3: Heredocs .....	43
Section 14.4: Removing trailing newlines .....	44
<b>Chapter 15: Split a string on unquoted separators .....</b>	<b>46</b>
Section 15.1: parse_line() .....	46
Section 15.2: Text::CSV or Text::CSV_XS .....	46
<b>Chapter 16: Object-oriented Perl .....</b>	<b>47</b>
Section 16.1: Defining classes in modern Perl .....	47
Section 16.2: Creating Objects .....	47
Section 16.3: Defining Classes .....	48
Section 16.4: Inheritance and methods resolution .....	49
Section 16.5: Class and Object Methods .....	51
Section 16.6: Roles .....	52
<b>Chapter 17: Exception handling .....</b>	<b>54</b>
Section 17.1: eval and die .....	54
<b>Chapter 18: Regular Expressions .....</b>	<b>55</b>
Section 18.1: Replace a string using regular expressions .....	55
Section 18.2: Matching strings .....	55
Section 18.3: Parsing a string with a regex .....	55
Section 18.4: Usage of \Q and \E in pattern matching .....	56
<b>Chapter 19: XML Parsing .....</b>	<b>57</b>
Section 19.1: Parsing with XML::Twig .....	57
Section 19.2: Consuming XML with XML::Rabbit .....	58
Section 19.3: Parsing with XML::LibXML .....	60
<b>Chapter 20: Unicode .....</b>	<b>62</b>
Section 20.1: The utf8 pragma: using Unicode in your sources .....	62
Section 20.2: Handling invalid UTF-8 .....	62
Section 20.3: Command line switches for one-liners .....	63

Section 20.4: Standard I/O .....	64
Section 20.5: File handles .....	64
Section 20.6: Create filenames .....	65
Section 20.7: Read filenames .....	66
<b>Chapter 21: Perl one-liners</b> .....	<b>68</b>
Section 21.1: Upload file into mojolicious .....	68
Section 21.2: Execute some Perl code from command line .....	68
Section 21.3: Using double-quoted strings in Windows one-liners .....	68
Section 21.4: Print lines matching a pattern (PCRE grep) .....	68
Section 21.5: Replace a substring with another (PCRE sed) .....	69
Section 21.6: Print only certain fields .....	69
Section 21.7: Print lines 5 to 10 .....	69
Section 21.8: Edit file in-place .....	69
Section 21.9: Reading the whole file as a string .....	69
<b>Chapter 22: Randomness</b> .....	<b>70</b>
Section 22.1: Accessing an array element at random .....	70
Section 22.2: Generate a random integer between 0 and 9 .....	70
<b>Chapter 23: Special variables</b> .....	<b>71</b>
Section 23.1: Special variables in perl: .....	71
<b>Chapter 24: Packages and modules</b> .....	<b>72</b>
Section 24.1: Using a module .....	72
Section 24.2: Using a module inside a directory .....	72
Section 24.3: Loading a module at runtime .....	73
Section 24.4: CPAN.pm .....	73
Section 24.5: List all installed modules .....	74
Section 24.6: Executing the contents of another file .....	74
<b>Chapter 25: Install Perl modules via CPAN</b> .....	<b>75</b>
Section 25.1: cpanminus, the lightweight configuration-free replacement for cpan .....	75
Section 25.2: Installing modules manually .....	75
Section 25.3: Run Perl CPAN in your terminal (Mac and Linux) or command prompt (Windows) .....	76
<b>Chapter 26: Easy way to check installed modules on Mac and Ubuntu</b> .....	<b>78</b>
Section 26.1: Use perldoc to check the Perl package install path .....	78
Section 26.2: Check installed perl modules via terminal .....	78
Section 26.3: How to check Perl corelist modules .....	78
<b>Chapter 27: Pack and unpack</b> .....	<b>79</b>
Section 27.1: Manually Converting C Structs to Pack Syntax .....	79
Section 27.2: Constructing an IPv4 header .....	80
<b>Chapter 28: Perl commands for Windows Excel with Win32::OLE module</b> .....	<b>82</b>
Section 28.1: Opening and Saving Excel/Workbooks .....	82
Section 28.2: Manipulation of Worksheets .....	82
Section 28.3: Manipulation of cells .....	83
Section 28.4: Manipulation of Rows / Columns .....	84
<b>Chapter 29: Simple interaction with database via DBI module</b> .....	<b>85</b>
Section 29.1: DBI module .....	85
<b>Chapter 30: Perl Testing</b> .....	<b>87</b>
Section 30.1: Perl Unit Testing Example .....	87
<b>Chapter 31: Dancer</b> .....	<b>89</b>
Section 31.1: Easiest example .....	89

<b>Chapter 32: Attributed Text</b> .....	90
Section 32.1: Printing colored Text .....	90
<b>Chapter 33: GUI Applications in Perl</b> .....	91
Section 33.1: GTK Application .....	91
<b>Chapter 34: Memory usage optimization</b> .....	92
Section 34.1: Reading files: foreach vs. while .....	92
Section 34.2: Processing long lists .....	92
<b>Chapter 35: Perl script debugging</b> .....	93
Section 35.1: Run script in debug mode .....	93
Section 35.2: Use a nonstandard debugger .....	93
<b>Chapter 36: Perlbrew</b> .....	94
Section 36.1: Setup perlbrew for the first time .....	94
<b>Chapter 37: Installation of Perl</b> .....	95
Section 37.1: Linux .....	95
Section 37.2: OS X .....	95
Section 37.3: Windows .....	96
<b>Chapter 38: Compile Perl cpan module sapnwrfc from source code</b> .....	97
Section 38.1: Simple example to test the RFC connection .....	97
<b>Chapter 39: Best Practices</b> .....	98
Section 39.1: Using Perl::Critic .....	98
<b>Credits</b> .....	102
<b>You may also like</b> .....	104

[Click here to download full PDF material](#)