

Perl Statements How To Tutorial

(this is a tutorial sample only)

by Mihael Avram

www.misc-perl-info.com/perl-statements-howto.html

Table of Contents

1. Copyright
2. Introduction
 - 2.1. How to run the examples included in this eBook
 - 2.1.1. How to run the script in Windows
 - 2.1.2. How to run the script in Linux
3. Perl statements
 - 3.1. Perl simple statements
 - 3.2. Perl blocks
 - 3.3. Perl Compound Statements
 - 3.4. Perl Control Statements
 - 3.4.1. Perl Conditional Statements
 - 3.4.1.1. **if**
 - 3.4.1.1.1. The syntax forms
 - 3.4.1.1.2. How to use a simple if statement
 - 3.4.1.1.3. How to use if with the else clause
 - 3.4.1.1.4. How to use if with a { do nothing } block
 - 3.4.1.1.5. How to use if with the elsif and else clauses
 - 3.4.1.1.6. How to use if as an expression modifier
 - 3.4.1.1.7. How to use if with the exists function
 - 3.4.1.1.8. How to use if with the next and last loop controls
 - 3.4.1.1.9. How to use if with defined
 - 3.4.1.1.10. How to use the operators with the if statement
 - 3.4.1.1.11. How to use if with the pattern binding operator =~
 - 3.4.1.1.12. How to use if with the AND (&&) short-circuit operator
 - 3.4.1.1.13. How to use if with the OR (||) short-circuit operator
 - 3.4.1.1.14. How to use if statement to find if a variable is reference
 - 3.4.1.1.15. How to use a function with the if statement
 - 3.4.1.1.16. How to use the if statement with regular expressions
 - 3.4.1.2. **unless**
 - 3.4.1.2.1. The syntax forms
 - 3.4.1.2.2. How to use a simple unless statement
 - 3.4.1.2.3. How to use unless with the else clause
 - 3.4.1.2.4. How to use unless with the elsif and else clauses
 - 3.4.1.2.5. How to use unless as an expression modifier
 - 3.4.1.2.6. How to use unless with the exists function

- 3.4.1.2.7. How to use unless with the next loop control
- 3.4.1.2.8. How to use unless with defined
- 3.4.1.3. **switch**
 - 3.4.1.3.1. How to emulate switch using a bare block and && operator
 - 3.4.1.3.2. How to emulate switch using an if modifier inside a while loop
 - 3.4.1.3.3. How to emulate a switch statement using a hash structure
 - 3.4.1.3.4. How to use the Perl Switch module
 - 3.4.1.3.5. How to use switch starting with the 5.10 Perl version
- 3.4.2. Perl Loop Statements
 - 3.4.2.1. **for**
 - 3.4.2.1.1. The syntax forms
 - 3.4.2.1.2. A simple for statement example
 - 3.4.2.1.3. A for infinite loop
 - 3.4.2.1.4. How to use the for statement to print the keys of a hash
 - 3.4.2.1.5. How to print a two dimensional array using a nested for
 - 3.4.2.2. **foreach**
 - 3.4.2.2.1. The syntax forms
 - 3.4.2.2.2. How to use foreach with the iterator variable
 - 3.4.2.2.3. How to use foreach as a modifier of a statement
 - 3.4.2.2.4. How to use foreach with the special variable \$_
 - 3.4.2.2.5. How to use foreach and \$#
 - 3.4.2.2.6. How to use foreach with the .. range operator
 - 3.4.2.2.7. How to use foreach and @_
 - 3.4.2.2.8. How to use foreach with my or local
 - 3.4.2.2.9. How to use the foreach loop statement with STDIN
 - 3.4.2.2.10. How to use the foreach loop with the next loop control
 - 3.4.2.2.11. How to use the foreach loop with the last loop control
 - 3.4.2.2.12. How to use the foreach loop with the redo loop control
 - 3.4.2.2.13. How to use the foreach loop with the continue clause
 - 3.4.2.2.14. How to quit a foreach loop
 - 3.4.2.2.15. How to use nested foreach loops
 - 3.4.2.2.16. How to use the foreach loop with a list
 - 3.4.2.2.17. How to use the foreach loop with an array
 - 3.4.2.2.18. How to use foreach to traverse an array in a reversed order
 - 3.4.2.2.19. How to use foreach and array references
 - 3.4.2.2.20. How to create an array inside a foreach loop
 - 3.4.2.2.21. How to use foreach to modify the elements of an array
 - 3.4.2.2.22. How to use foreach to print all array elements

- 3.4.2.2.23. How to use foreach beginning from a given index of an array
- 3.4.2.2.24. Find a pattern running foreach across an array
- 3.4.2.2.25. How to use foreach and delete on an array
- 3.4.2.2.26. How to use foreach with two arrays of same size
- 3.4.2.2.27. How to use foreach and the arrays concatenation
- 3.4.2.2.28. Using foreach to convert an array to lowercase/uppercase
- 3.4.2.2.29. Using foreach to find all the odd, even elements in a list
- 3.4.2.2.30. How to find max and min using a foreach loop
- 3.4.2.2.31. How to use foreach with a scalar
- 3.4.2.2.32. Find the union, intersection and difference of two arrays
- 3.4.2.2.33. How to traverse a hash using foreach
- 3.4.2.2.34. How to create a hash inside a foreach loop
- 3.4.2.2.35. How to use foreach and keys function to print a hash
- 3.4.2.2.36. How to sort a hash by keys in a foreach loop
- 3.4.2.2.37. How to sort a hash by values in a foreach loop
- 3.4.2.2.38. How to use foreach and delete on a hash
- 3.4.2.2.39. How to use hash references inside a foreach loop
- 3.4.2.2.40. How to use foreach and chomp
- 3.4.2.2.41. How to check defined in a foreach loop
- 3.4.2.2.42. How to use foreach and substr
- 3.4.2.2.43. How to use foreach and length
- 3.4.2.2.44. How to use foreach and split
- 3.4.2.2.45. foreach and pop, push, shift, unshift example
- 3.4.2.2.46. How to use foreach and push
- 3.4.2.2.47. How to use nested foreach loops to print a 3d-array
- 3.4.2.2.48. How to use nested foreach to print a 3d-hash
- 3.4.2.2.49. How to use foreach with an AOA (array of arrays)
- 3.4.2.2.50. How to use foreach with an AOH (array of hashes)
- 3.4.2.2.51. How to use foreach with a HOA (hash of arrays)
- 3.4.2.2.52. How to use foreach with a HOH (hash of hashes)
- 3.4.2.2.53. Which is the difference between foreach and for
- 3.4.2.2.54. foreach versus grep
- 3.4.2.2.55. foreach versus map
- 3.4.2.2.56. foreach versus while
- 3.4.2.2.57. How to emulate a switch statement in a foreach loop
- 3.4.2.2.58. How to use the goto statement inside a foreach block
- 3.4.2.3. **while**
 - 3.4.2.3.1. The syntax forms

- 3.4.2.3.2. A simple while statement example
- 3.4.2.3.3. How to use while when condition expression is false initially
- 3.4.2.3.4. The while statement and an infinite loop
- 3.4.2.3.5. How to use while with my in the conditional expression
- 3.4.2.3.6. How to simulate a for statement using a while loop
- 3.4.2.3.7. How to use the while statement with and \$_
- 3.4.2.3.8. How to use while with the diamond operator <>
- 3.4.2.3.9. How to use the while statement with @ARGV and <>
- 3.4.2.3.10. How to use while with chomp, lc and STDIN
- 3.4.2.3.11. How to use while with defined and STDIN
- 3.4.2.3.12. How to use while as a modifier
- 3.4.2.3.13. How to split a file record on columns in a while loop
- 3.4.2.3.14. How to skip blank lines in a while loop
- 3.4.2.3.15. How to quit a while loop using the last loop control
- 3.4.2.3.16. How to use the next loop control inside a while loop
- 3.4.2.3.17. How to use the redo loop control inside a while loop
- 3.4.2.3.18. How to use the while loop with the continue clause
- 3.4.2.3.19. How to control the while loop flow with next, last and redo
- 3.4.2.3.20. How to use nested while loops with labels
- 3.4.2.3.21. How to use while with push and shift
- 3.4.2.3.22. How to print a hash using each and while
- 3.4.2.3.23. How to find the while odd and even iterations
- 3.4.2.3.24. How to find a substring in a string using a while loop
- 3.4.2.3.25. How to use while with pattern matching
- 3.4.2.3.26. How to use while and defined with an array
- 3.4.2.3.27. How to use while to generate an AOA (array of arrays)
- 3.4.2.3.28. How to use while to generate an AOH (array of hashes)
- 3.4.2.4. **do-while**
 - 3.4.2.4.1. The syntax form
 - 3.4.2.4.2. A simple do-while statement example
 - 3.4.2.4.3. How to use the do-while statement with STDIN
 - 3.4.2.4.4. How to use do-while with the loop controls
- 3.4.2.5. **until**
 - 3.4.2.5.1. The syntax forms
 - 3.4.2.5.2. A simple until statement example
 - 3.4.2.5.3. The until statement and an infinite loop
 - 3.4.2.5.4. How to use until and STDIN
 - 3.4.2.5.5. How to use until with next, last, redo and continue

- 3.4.2.5.6. How to use the until statement with arrays
- 3.4.2.5.7. How to use the until statement with hashes
- 3.4.2.5.8. How to use until as a modifier
- 3.4.2.6. **do-until**
 - 3.4.2.6.1. The syntax form
 - 3.4.2.6.2. A simple do-until statement example
 - 3.4.2.6.3. How to use do-until in conjunction with arrays and hashes
 - 3.4.2.6.4. How to use do-until construct with the loop controls
- 3.4.3. Perl Loop Controls
- 3.4.4. Perl Jump Statements
- 3.4.5. Perl Modifiers

In the following example the `foreach` loop is used to transform an array into a hash.

See the code:

```
#!/usr/local/bin/perl

use strict;
use warnings;

# initialize an array
my @array = ('1-one', '2-two', '3-three',
            '4-four', '5-five', '6-six');

my %hash;

foreach (@array) {
    my @tmp = split /-/;
    $hash{$tmp[0]} = $tmp[1];
}

print "$_ => $hash{$_}\n" foreach (keys %hash);
```

And the output:

```
6 => six
4 => four
1 => one
3 => three
2 => two
5 => five
```

Inside the block of the first `foreach` statement:

- each element of the array is assigned in turn to the special variable `$_`
- the string from `$_` is `split` into the `@tmp` array, using the '-' separator
- the pair element is added to the hash

The second one-line `foreach` is used to print the elements of the hash, if you want them in a specific order you need to order them by using the `sort` function.

The same example is rewritten below using the `map` function:

```
#!/usr/local/bin/perl

use strict;
use warnings;

# initialize an array
my @array = ('1-one', '2-two', '3-three',
            '4-four', '5-five', '6-six');

my %hash = map split (/--/), @array;

print map { "$_ => $hash{$_}\n" } keys %hash;
```

Now I want to say a few words about the first `map`. It is used the syntax form with an expression – for each element of the array, `map` evaluates the expression and finally it will return a list that has twice as many elements as the `@array`. Next, the list returned by `map` will be turned into a hash, grouping each two elements into a pair (key, value) hash element.

The second `map` will print the elements of the hash.

Just as a note, instead of the line of code with the first `map` that uses the syntax form with an expression, you could use the `map` form with the syntax form for blocks, but this is slower. Anytime you use a block, you introduce a new scope which adds some extra options and time to execute it. So try to use a block only when necessary.

Anyway, the form of `map` with a block is as follows:

```
my %hash = map { split /--/ } @array;
```

3.4.2.2.56. *foreach versus while*

If you want to read large files in your scripts, the `while` statement is more convenient.

In a `foreach` loop, the Perl interpreter needs to know the end of the array before iterating over it. If the data is in a file handle, you first need to read the whole file into an array and only then you can begin iterating over it. This could end with some memory issues if your file is big.

The alternative is to use the `while` statement. In a `while` statement Perl will read the records of the file one at a time and will finish the loop when the EOF is reached.

Perl Glossary

(this is a glossary sample only)

by Mihael Avram

www.misc-perl-info.com/perl-statements-howto.html

<p>\$a, \$b</p>	<p>They are special package variables and you can use them in the <code>sort</code> function. For example:</p> <pre>@array = sort {\$a <=> \$b} @array;</pre> <p>will sort the <code>@array</code> numerically, and:</p> <pre>@array = sort {\$b cmp \$a} @array;</pre> <p>will sort the <code>@array</code> in a reverse alphabetical order.</p>
<p><=></p>	<p>It is the binary numeric comparison operator and returns <code>-1</code>, <code>0</code>, or <code>1</code> if the left operand is less than, equal to or greater than the right one. Example:</p> <pre>my (\$v1, \$v2, \$v) = (5, 7); \$v = \$v1 <=> \$v2; print "\$v\n"; # displays -1;</pre>
<p>=~</p>	<p>This binary operator binds a string expression to a pattern match. The string which is intended to bind is put on the left meanwhile the operator itself is put on the right. You can use the pattern binding operator when you have a string which is not stored in the <code>\$_</code> variable and you need to perform some matches or substitutions upon that string. Example:</p> <pre>my \$v = "black and white"; if(\$v =~ m/white/) { print "Yes (expected)\n"; }</pre>
<p>/e</p>	<p>It is a substitution operator modifier and tells the regex engine to evaluate the replacement as if it were a Perl statement. An example to illustrate how it works:</p> <pre>my \$str = "This is about Perl" ; \$str =~ s/Perl/uc 'php'/e; print "\$str\n"; # it displays: This is about PHP</pre> <p>The <code>/e</code> modifier will tell to the regex engine that the replacement field of the substitution operator is pure Perl code, so it will call the <code>uc</code> (uppercase) function with the <code>'php'</code> string argument, returning the value of <code>'php'</code> string in uppercases as a replacement text. Please note the use of binding (<code>=~</code>) operator here.</p>