

# Perl chomp Function Tutorial

by Mihael Avram  
[www.misc-perl-info.com](http://www.misc-perl-info.com)

# 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 string functions
  - 3.1. Perl chomp Function
    - 3.1.1. The syntax forms
    - 3.1.2. How to use the return value of the chomp function
    - 3.1.3. How to chomp a string variable
    - 3.1.4. How to chomp an array
    - 3.1.5. Read file as a chomped array
    - 3.1.6. How to chomp the array elements one by one
    - 3.1.7. How to chomp a hash
    - 3.1.8. How to use chomp and the special variable \$\_
    - 3.1.9. How to use chomp with \$/ and set \$/ to different strings
    - 3.1.10. How to chomp a reference variable (scalar, array or hash)
    - 3.1.11. How to use chomp when reading from STDIN
    - 3.1.12. How to use chomp in a single line along with STDIN
    - 3.1.13. How to use chomp to remove all the trailing newlines from a string
    - 3.1.14. How to use chomp with foreach statement
    - 3.1.15. How to use chomp with while statement and STDIN
    - 3.1.16. How to use chomp along with defined and STDIN
    - 3.1.17. How to use chomp with last, next and redo loop controls
    - 3.1.18. A chomp, index, lc, uc, unshift and push example
    - 3.1.19. How to chomp large data
    - 3.1.20. How to use chomp and push
    - 3.1.21. How to use chomp and split
    - 3.1.22. How to use chomp, lc and if on STDIN
    - 3.1.23. How to use chomp and hex
    - 3.1.24. How to chomp complex data structures using a recursive subroutine
    - 3.1.25. How to use chomp when reading multiline strings
    - 3.1.26. How to remove LF or CR at the end of a line read from a text file
    - 3.1.27. How to use chomp with a binary file
    - 3.1.28. Which is the difference between chop and chomp

```
@LIST = map BLOCK @ARRAY
```

In our example, the `map` function waits for an array as its second argument, so Perl will create an array when reading from `<DATA>`. Next, the `map` function will run the block `{chomp; $_ => 0}` on each element of the array read from `<DATA>`:

- each element of the array will be assigned to `$_`
- `chomp` will remove the trailing newline from `$_`
- the pair element `($_,0)` will be added to `%fruits` hash

Finally, I print the key of `%fruits` hash using the `foreach` statement and the `keys` function. The `keys` function creates an array with the keys of a hash and only afterwards `foreach` statement will loop through the keys array. Please note that this approach to `print` a hash using `foreach` and `keys` is not very suitable for hashes with large amount of data.

The output is as follows:

```
cherry plum apricot
```

### 3.1.8. How to use `chomp` and the special variable `$_`

The special variable `$_` is a scalar variable, so in this case we use `chomp` in a scalar context. If you recall the third syntax form of `chomp` function, if you use `chomp` without argument that means that the content of `$_` variable will be chomped. See the next short example for this:

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

use strict;
use warnings;

$_ = "123\n";
my $nr = chomp;      # or my $nr = chomp $_;
print "Characters removed: $nr\n";
# it expects: Characters removed: 1
```

You can use this short form of `chomp` (where the argument by default is `$_`) within a block of a statement or with a function that uses `$_` (like `foreach`, `map`, etc). The next example chomps the elements of an array one by one, using the `foreach` statement:

# Perl Glossary

by Mihael Avram  
[www.misc-perl-info.com](http://www.misc-perl-info.com)

<p><b>\$a, \$b</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 &lt;=&gt; \$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><b>&lt;=&gt;</b></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 &lt;=&gt; \$v2; print "\$v\n"; # displays -1;</pre>
<p><b>=~</b></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><b>/e</b></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>