

Perl Scalar and String Functions

(How to Tutorial)

(this is a tutorial sample only)

by Mihael Avram

<http://www.misc-perl-info.com/perl-str-func-howto.html>

Copyright © 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 scalar and 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 |

| | |
|--------|--|
| 3.2. | Perl chop function |
| 3.2.1. | The syntax forms |
| 3.2.2. | How to chop the first character of a string |
| 3.2.3. | How to use chop with a string variable |
| 3.2.4. | How to use chop with arrays |
| 3.2.5. | How to use chop with hashes |
| 3.2.6. | How to use chop with \$_ and foreach |
| 3.3. | Perl chr function |
| 3.3.1. | The syntax forms |
| 3.3.2. | A simple example |
| 3.3.3. | Convert an ASCII code string into a character string |
| 3.3.4. | Convert a hexadecimal string into a character string |
| 3.4. | Perl crypt function |
| 3.4.1. | The syntax forms |
| 3.4.2. | A simple example |
| 3.4.3. | How to use random characters for the salt |
| 3.4.4. | How to check a user password in Linux |
| 3.5. | Perl hex function |
| 3.5.1. | The syntax forms |
| 3.5.2. | How to convert a hexadecimal number into a decimal one |
| 3.5.3. | How to convert a decimal value into another numeration base |
| 3.5.4. | How to convert a scalar string into a hexadecimal format |
| 3.6. | Perl index function |
| 3.6.1. | The syntax forms |
| 3.6.2. | How to check if a string is a substring of another |
| 3.6.3. | How to find the leftmost occurrence of a substring in a string |
| 3.6.4. | How to find all the occurrences of a substring in a string |
| 3.6.5. | How to count all the matches of a substring in a string |
| 3.6.6. | How to remove the trailing spaces using the index function |
| 3.7. | Perl lc function |
| 3.7.1. | The syntax forms |
| 3.7.2. | How to convert a string in lowercase |
| 3.7.3. | How to compare two strings case insensitive |
| 3.7.4. | How to convert all the array elements in lowercase |
| 3.8. | Perl lcfirst function |
| 3.8.1. | The syntax forms |
| 3.8.2. | How to convert the first character of a string in lowercase |
| 3.9. | Perl length function |

| | |
|----------|--|
| 3.9.1. | The syntax forms |
| 3.9.2. | How to find the number of characters of a string variable |
| 3.9.3. | How to find the length of a string in bytes |
| 3.9.4. | Sort the words of a string in a descending length order |
| 3.9.5. | How to process a string one character at a time |
| 3.9.6. | Get the string length of the shortest/longest string in an array |
| 3.9.7. | How to use the length function in a foreach loop |
| 3.9.8. | Sort the elements of an array by string length |
| 3.9.9. | Sort the elements of a hash by the length of the values |
| 3.9.10. | Sort the elements of a hash by the length of the keys |
| 3.9.11. | How to find the length of an AOA (array of arrays) |
| 3.9.12. | How to find the length of an HOH (hash of hashes) |
| 3.10. | Perl oct function |
| 3.10.1. | The syntax forms |
| 3.10.2. | Simple examples |
| 3.10.3. | Octal File Permissions and the Perl oct function |
| 3.11. | Perl ord function |
| 3.11.1. | The syntax forms |
| 3.11.2. | A simple example |
| 3.11.3. | Convert a character string into an ASCII code array |
| 3.11.4. | Convert a character string into a hexadecimal string |
| 3.11.5. | How to use ord and pack to convert from binary to decimal |
| 3.12. | Perl pack function |
| 3.12.1. | The syntax form |
| 3.12.2. | A string with binary data will be null padded [a] |
| 3.12.3. | A text (ASCII) string will be space padded [A] |
| 3.12.4. | A bit string (ascending bit order inside each byte) [b] |
| 3.12.5. | A bit string (descending bit order inside each byte) [B] |
| 3.12.6. | A signed char (8-bit) value [c] |
| 3.12.7. | A unsigned char (octet) value [C] |
| 3.12.8. | A double-precision float in the native format [d] |
| 3.12.9. | A single-precision float in the native format [f] |
| 3.12.10. | A hex string (low nibble first) format [h] |
| 3.12.11. | A hex string (high nibble first) format [H] |
| 3.12.12. | A signed integer value [i] |
| 3.12.13. | An unsigned integer value [I] |
| 3.12.14. | A signed long (32-bit) value [l] |
| 3.12.15. | An unsigned long value [L] |

| | |
|----------|---|
| 3.12.16. | An unsigned short (16-bit) in a big-endian order [n] |
| 3.12.17. | An unsigned long (32-bit) in a big-endian order [N] |
| 3.12.18. | A signed short (16-bit) value [s] |
| 3.12.19. | An unsigned short value [S] |
| 3.12.20. | A Unicode character number [U] |
| 3.12.21. | An unsigned short (16-bit) in little-endian order [v] |
| 3.12.22. | An unsigned long (32-bit) in little-endian order [V] |
| 3.12.23. | A null byte [x] |
| 3.12.24. | Back up a byte [X] |
| 3.12.25. | How to reverse the bits in each character of a string |
| 3.12.26. | Short dictionary |
| 3.13. | Perl q/STRING/ |
| 3.14. | Perl qq/STRING/ |
| 3.15. | Perl reverse function |
| 3.15.1. | The syntax forms |
| 3.15.2. | Simple examples |
| 3.15.3. | Get a sentence with the words in an inverted order |
| 3.15.4. | Check up if a string is palindrome |
| 3.16. | Perl rindex function |
| 3.16.1. | The syntax forms |
| 3.16.2. | How to check if a substring is included in a string |
| 3.16.3. | How to find the rightmost occurrence of a substring in a string |
| 3.16.4. | How to get all the occurrences of a substring in a string |
| 3.17. | Perl sprintf function |
| 3.17.1. | The syntax form |
| 3.17.2. | sprintf versus printf |
| 3.17.3. | How to limit the number of decimal places in your numbers |
| 3.17.4. | How to round a certain number of decimals |
| 3.17.5. | How to convert from decimal to hexadecimal |
| 3.17.6. | How to convert from decimal to octal |
| 3.17.7. | How to convert from decimal to binary |
| 3.17.8. | How to pad a string with blanks |
| 3.17.9. | How to pad a number on the left with zeroes |
| 3.17.10. | How to expand variables in text strings |
| 3.17.11. | How to format floating-point numbers |
| 3.17.12. | How to format the local time in a scalar variable |
| 3.17.13. | How to use sprintf with regexp |
| 3.18. | Perl substr function |

| | |
|----------|--|
| 3.18.1. | The syntax forms |
| 3.18.2. | How to use substr to extract a substring from a string |
| 3.18.3. | How to use substr like a lvalue |
| 3.18.4. | Replace a substring with a different substring in a string |
| 3.18.5. | How to get the column fields from a flat file database |
| 3.18.6. | How to pad a string left and right with any character |
| 3.18.7. | How to convert between number representations |
| 3.18.8. | Split a string into fixed-length parts |
| 3.18.9. | How to extract a substring between two other substrings |
| 3.18.10. | The substr function and the =~ binding operator |
| 3.19. | Perl tr/// operator |
| 3.19.1. | The syntax form |
| 3.19.2. | How to use tr/// with the =~ binding operator |
| 3.19.3. | How to use tr/// with \$_ |
| 3.19.4. | How to use tr/// with the complement (c) modifier |
| 3.19.5. | How to use tr/// with the delete (d) modifier |
| 3.19.6. | How to use tr/// with the squash (s) modifier |
| 3.19.7. | How to use tr/// with more than one modifier |
| 3.19.8. | How to use variables in tr/// |
| 3.19.9. | How to use tr/// for counting characters |
| 3.19.10. | Which is the difference between tr/// and s/// |
| 3.20. | Perl uc function |
| 3.20.1. | The syntax forms |
| 3.20.2. | How to convert a string in uppercase |
| 3.20.3. | How to compare two strings case insensitive |
| 3.20.4. | How to convert all the array elements in uppercase |
| 3.21. | Perl ucfirst function |
| 3.21.1. | The syntax forms |
| 3.21.2. | Convert the first character of a string in uppercase |
| 3.21.3. | How to capitalize all words in a string |
| 3.22. | Perl y/// operator |

The following example shows you a simple way to print the hash elements in the descending order of the keys length, using the `sort` and the `length` functions:

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

use strict;
use warnings;

# define a hash
my %hash = (if => 1, until => 1, foreach =>1, else => 1);

# foreach loop
foreach my $key (sort { length $b <=> length $a } keys %hash) {
    print "$key: $hash{$key}\n";
}
```

The output:

```
foreach: 1
until: 1
else: 1
if: 1
```

You can note that the `length` function is used in a syntax format without parentheses to make the code more readable.

3.9.11. How to find the length of an AOA (array of arrays)

I'll examine below the Perl array length, by which I mean the length in bytes for a Perl array.

For computing the Perl array length I'll use the `length` function which returns the number of characters of an expression.

If you'll use Unicode characters in your arrays, the number of characters may be different from the number of bytes.

Let's examine the following example:

```
#!/usr/bin/perl

use warnings;
use strict;

my @AOA = ("blue", "red", "green",
          ["lightgreen", "lightblue"]);

# using the length() function
{
    use bytes;
    my $count = 0;

    foreach my $item1 (@AOA){
        foreach my $item2 (@{$item1}){
            $count += length($item2);
        }
    }
    print "Perl array length (through length() function) is: ",
          $count, "\n";
}
```

The `@AOA` is a 2-dimensional array and the code snippet shown above will get us the length in bytes of the `@AOA` array. I used the `use bytes` pragma, and I put the snippet code in a block `{ }` in order to limit the effect of this pragma to the scope in which it appears, i.e. inside the block.

To print the length of the `@AOA` array the `foreach` loop statement is used.

The output after running this code is as follows:

```
Perl array length (through length() function) is: 31
```

This is one way of finding out how big the array is in bytes, but this is not the real memory usage of the array Perl variable. If you want to find the memory usage of an array variable, you can use the `Devel::Size` module (a quick search through the CPAN modules will help you to find it). However, keep in mind that the result will be dependent of your OS.

3.9.12. How to find the length of an HOH (hash of hashes)

I'll examine below the Perl hash length, by which I mean the length in bytes of all the hash values. It's appropriate for hashes that have strings as values.

For computing the hash length I'll use the `length` function which returns the number of characters of an expression. If you'll use Unicode characters in your hash values, the number of characters may be different from the number of bytes.

Perl Glossary

(this is a glossary sample only)

by Mihael Avram

<http://www.misc-perl-info.com/perl-str-func-howto.html>

| | |
|---|---|
| <p><code>\$a</code>, <code>\$b</code></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><code><=></code></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><code>=~</code></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><code>/e</code></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> |