Perl split Function

Perl **split** function allows you to break up a string into an array or a list, by using a specific pattern.

The pattern can be either a single character or a more sophisticated regular expression.

In this last case you need to know a bit about the regexp language alphabet.

Anyway, in this short free tutorial I'll give you all the necessary information to understand the provided snippet code and I'll show you some ways about how you can use Perl **split** function in your scripts.

Please note that Perl **split** function is the opposite of Perl join function, and there are a lot of situations where they are used together.

www.profmariamichael.com

While Perl **join** function connects the elements of a list into a string, Perl **split** function breaks apart a string into a list of substrings and returns that list.

The general syntax form of the Perl **split** function is as follows:

```
LIST = split(/PATTERN/, EXPR, LIMIT);
```

where:

- **LIST** – is a list, array or hash that is returned by the **split** function; if it is omitted, the Perl **split** function will be called in a scalar context and the **EXPR** will be split in the `@_` array – in this case the function will return the number of fields in which the **EXPR** will be split (anyway, be aware if your call to **split** function is inside the body of a subroutine because it alters the value of `@_`)
- **PATTERN** – usually is a regular expression but could be a single character or a string; the **EXPR** is split on every occurrence of the **PATTERN**, if **LIMIT** is not specified; if the **PATTERN** parameter is omitted, the Perl split function will split on whitespace, skipping any leading ones.
- **EXPR** – is the string expression that will be split into an array or a list; if this parameter is omitted, the content of the special scalar variable `$_` will be split
- **LIMIT** is the maximum number of fields the **EXPR** will be split into

You can omit the parenthesis, the **LIST** that will be returned and any of the third arguments presented above – only the word split is compulsory.

Please note that if you use inside the pattern a metacharacter as:

```
^ $ ( ) \ | @ [ { ? . + *
```

you need to escape it by using the \ (backslash character) to indicate that this character is to be regarded as something to match and not some fancy character. For instance, if you want to split your string using the pattern `/&&/` you need to write it as `/\&\&/`, otherwise you certainly won't get what you expect.

I hope that the following examples will clear up some of the features described above.

Using split without parameters and in a scalar context

```
# initialize $_
$_ = "Perl array functions";
split;
print "@ \n";
# it displays: Perl array functions
```

In this example, all the parameters are omitted:

- **EXPR** is omitted, therefore the `$_` will be used
- **PATTERN** is omitted, so whitespace will be used
- **LIMIT** is omitted, so there are no limits for the number of elements returned in the list
- **LIST** is omitted, hence we are in a scalar context and the `@_` special array will be used

In order to print the elements of the special variable array `@_` delimitated by space, we use the print function with quotes.

Using Perl split function when the delimiter is a string

```
my $string = 'brown || red || yellow || white';
my @colors = split(' \|\| ', $string);
print "@colors\n";
#it displays: brown red yellow white
```

In the above example, the string separator `' || '` consists of 4 characters, from which two are metacharacters, so we must escape them.

There are situations where the separator consists of a single character, like in the following example:

```
$string = 'brown:red:yellow:white';
@colors = split(':', $string);
```

If we'll print the array like in the previous example, we'll get the same result.

Using split with different characters within the pattern

We define a set of characters and we want to split a string whenever we found a character from our set. For example, let's say our set contains the characters:

-	dash
,	comma
.	dot
	space
:	colon
;	semicolon
!	exclamation mark
?	question mark
()	round brackets
[]	square brackets
{}	curly braces

I think there are enough characters in our set. Now we want to use Perl **`split`** function to split a string using the above set as a pattern. Please have a look at the following snippet code:

```
my $string = "This ?is! a  [short]-(sentence)! Why?";
my @array = split (/[\-,. :;!?()[\]{}]+/, $string);
print $_, " " foreach @array;
print "\n";
#it displays: This is a short sentence Why
```

We enclose our characters from the set in square brackets to specify that we want to match any of these characters and we use the + metacharacter to specify that any character can be repeated more than once. If we enclose some characters in square brackets, we say that we define a character class. Inside a character class the characters `] ^ -` are special so we must precede them by a backslash. In our example we escape the `]` and – characters.

Use Perl split function to read data from a csv file

A csv file is a comma delimited text data file and has the extension ".csv". There are a lot of applications such as Excel and MS Access that allows you to save the data in a csv format. It's very easy to create a csv file yourself, by delimiting the data with commas and ending each line with a newline character. The "persons.csv" file presented below is an example:

Persons.csv
Raw structure: firstname, lastname, age, eyecolor

```
John, Silva, 25, blue
Mary, Brown, 32, brown
Paul, Williams, 52, green
```

We intend to read this file and create the bidimensional **`@records`** array with the records of

the file. Each element of the array is a reference of the `@columns` array that has as elements the items from the file raw:

```
use warnings;
use strict;

my @records;
open FILE, "persons.csv" or die $!;
#we will read the file line by line
while(<FILE>)
{
  # read each line in $_
  # chomp the trailing newline from $_
  chomp;
  my @columns = split ", ", $_;
  push @records, \@columns;
}

# print the array
foreach my $record(@records) {
  foreach my $column(@{$record}) {

    # test the last column

    if(\$column == \$$record[-1]) {
      print "$column";
    }
    else {
      print "$column:";
    }
  }
  print "\n";
}
```

The output produced after running this snippet code is as follows:

```
John:Silva:25:blue
Mary:Brown:32:brown
Paul:Williams:52:green
```

Please note the line:

```
if(\$column == \$$record[-1])
```

where I tested if I reached the last element of the array in the `foreach` loop. See the next simple example if you need to find the last element of an array in a `foreach` loop:

```
@array = qw(one two three four);
foreach $item (@array) {
  if(\$item == \$array[-1]) {
    print "Last element of the array = $item\n";
  }
}
#it displays: Last element of the array = four
```

And of course, don't forget to look at the way you can use the `split` function to split the record of the file in columns.