Whirlwind of Perl Symbols

A Brief Introduction to Perl Special Symbols

Frank Mock*

June 4, 2016

Introduction

Perl has many special symbols that make programming tasks simple, compared to the amount of code other programming languages require to perform the same task. However, this power comes with a small price - learning what all the seemingly cryptic symbols mean. People new to Perl may find the learning curve for this language a bit steep relative to other languages due to it's cryptic syntax. This document will cover commonly used Perl special symbols and does not claim to include all of the Perl special symbols. With the introduction of each special symbol will be an example to demonstrate its use. This is important because some of the symbols have different meanings when used in a different context. However, keeping with the theme of this paper, my semantic explanations will be concise and to the point. This document assumes the reader is an experienced programmer, and does not go into the meaning of the example code beyond what and how the special symbol relates to it. I hope this document will help programmers new to Perl get up to speed quickly and make their initial experience with the language more enjoyable.

*http://www.frankmock.com

Comments

To make comments in perl code use the hash tag symbol, #

```
# This is a comment
```

Scalar and Array Related

A scalar holds a single value. When defining a scalar variable the **$** symbol is used. The scalar variable may be assigned numeric or string values.

```
$name = "Frank";
$number = 3;
```

An array variable holds multiple values. Use the **@** symbol when defining an array.

```
@names = ("George", "Fred", "Sue", "Amber");
```

The **qw** construct can be used to quote words in a list. The following names array is assigned the same values as above. Simply separate each name with a space and qw will quote each name and add a comma. The parenthesis are delimiters and can be changed to any other character. The fruit array below uses a forward slash as a delimiter.

```
@names = qw(George Fred Sue Amber);
@fruit = qw/pear, apple, bananas, grape/;
```

To print the elements of an array simply use the **print** command with the array name in double quotes. In Perl the default standard output is named **STDOUT** and is the terminal screen by default. The print command implicitly uses STDOUT to display the array elements to the user.

```
print"@names";
```

George Fred Sue Amber

To print each array element on a new line just add the newline escape sequence `\n` to the print statement. Those familar with other C based languages will recognize this character.

```
print"@names\n";
```

George
Fred
Sue
Amber

To assign a range of numbers or letters to an array use the range operator **..**

```
@numbers = (0..9);
print"@numbers";
```

0 1 2 3 4 5 6 7 8 9

```
@alphabet = ("a".."z");
print"@alphabet";
```

a b c d e f g h i j k l m n o p q r s t u v w x y z

To loop through the elements of an array a **while** loop can be used with the diamond operator <>. The diamond operator is a special kind of line-input operator that gets input from the users choice. In the example below, the array variable name is passed to the diamond operator as input.

```
while(<@alphabet>)
{
   print;
}
```

abcdefghijklmnopqrstuvwxyz

In the example above it does not appear that print command is acting on a variable, but it actually is. Behind the scenes print is implicitly using another special variable called the default scalar variable `$_`

`$_` does a lot in Perl - it holds the value of the current line when looping through a file or array, is the default pattern space for searches, and is the default variable for functions that require a scalar argument. The above loop can be re-written to explicitly use `$_`.

```
while(<@alphabet>)
{
   print $_;
}
```

abcdefghijklmnopqrstuvwxyz

Another way to loop through the elements of an array is to use a **foreach** loop. In this way you do not need to use the diamond operator.

```
foreach(@alphabet)
{
   print $_;
}
```

abcdefghijklmnopqrstuvwxyz

Notice that the while loop and the foreach loop do not add a space between elements. You can achieve this by explicitly doing so by placing double quotes around the default scalar variable and adding a space after it.

```
foreach(@alphabet)
{
   print "$_ ";
}
```

a b c d e f g h i j k l m n o p q r s t u v w x y z

You can print each element of an array on its own line by adding a newline character.

```
@colors = qw(red green blue yellow);
foreach(@colors)
{
   print "$_\n";
}
```

red
green
blue
yellow

Hash Related

A hash is a data structure like a dictionary or map. It maps keys to values. The % is used to declare a hash along with the => symbol that separates key and value pairs.

```
%eyeColor = ("Frank" => "Gray", "Debbie" => "Hazel", "Sydney" => "Green");
foreach(%eyeColor)
{
   print;
}
```

SydneyGreenFrankGrayDebbieHazel

Notice that a hash does not store its elements in any particular order so the elements were not printed as entered into the hash. The **keys** function will print the keys of a hash.

```
foreach(keys(%eyeColor)){ print"$_ "; }
```

Sydney Frank Debbie

The **values** function will print the values of a hash in no particular order.

```
foreach(values(%eyeColor))
{
   print"$_ ";
}
```

Gray Green Hazel

Regular Expressions

Regular expressions are used to search for patterns in text are a very powerful feature in Perl. For example, they allow programmers to count occurrences or filter out specific textual patterns. The search pattern is enclosed in a pair of forward slashes.

`.` matches any character except a newline

```
my $line = "George caused traffic to screech to a halt with a banana";
my @text = split(" ", $line);
foreach(@text)
{
    $_ = "pear" if /b./;
   print"$_ ";
}
```

George caused traffic to screech to a halt with a pear

In the code example above, /b./ matches any word that contains a lowercase b and is followed by any character except white space. The word banana is replaced with the word pear. Note that the pattern is case sensitive. By appending an i after the closing forward slash you can make the search case in-sensitive. For example: /b./i

Below are some more regular expression symbols and their meaning. This list is by no means exhaustive but is just the tip of the iceberg.

\d or [**0-9**] matches any single digit.

\D or [^**0-9**] matches any non-digit

\w or [**a-z**] matches an alphabetic character

\W or [^**a-z**] matches any non- alphabetic character

\s matches any whitespace character

\S matches any non-whitespace character

^ matches at the beginning (begins with)

$ matches at the end (ends with)

? match zero or more occurrences

***** match zero or one occurrences

+ match one or more occurrences

{NUMBER} NUMBER signifies the number of times to match.

Append a number between curly braces to signify the number of times to match the symbol. The following code example demonstrates this:

```
my $line = "Awesome Aerospace Company: 925-123-4567";
my @tokens = split(" ", $line);
foreach(@tokens)
{
    # print phone number if match to pattern is found
    print "$_ " if /\d{3}-\d{3}-\d{4}/;
}
```

925-123-4567

Notice that the caret symbol has a different meaning depending on it's placement in the regular expression. Used outside of square brackets means to find occurrences of the pattern at the beginning of a sequence of characters. And used inside of square brackets means to find all occurrences **not** matching the following pattern.

```
my $line = "Awesome Aerospace Company: 925-123-4567";
my @tokens = split(" ", $line);
foreach(@tokens)
{
    # print if a word begining with AE is found
    print "$_ " if /^Ae/;
}
```

Aerospace

Here is another example that searches an array of strings and finds words that begin with the letter a and ends with the letter c. The matches are printed. Only one match is found.

```
my $line = "arbitrary atomic arch character portrait particular";
my @tokens = split(" ", $line);
foreach(@tokens)
{
    # print word that begins with a and ends with c
    print "$_ " if /^a\w+c$/;
}
```

atomic

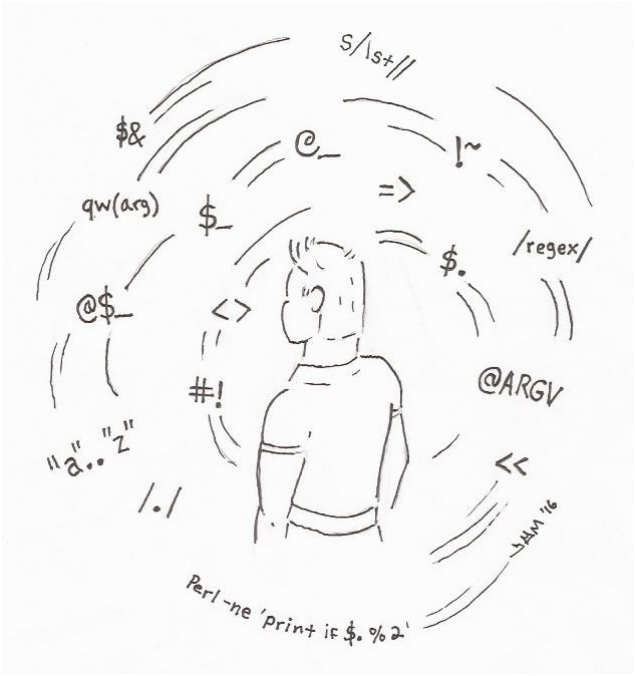
For those unfamiliar with regular expressions the pattern `/^a\w+c$/` can look quite cryptic. However, with time and practice even more cryptic patterns will become easy to read. Regular expressions are powerful and definitely a skill worth adding to your coding toolbox.

This has been a brief summary of many of the most common symbols used in Perl. I hope you found this useful and that your venture into the world of Perl is rewarding. The next page shows a list of common Perl symbols, some of which I did not cover. There are many more symbols than shown in the table. I suggest reading a good book on Perl if you want to learn more. Perl by Example, by Ellie Quigley is a good read. Also, check out the Perl website: https://www.perl.org/

Some Common Perl Special Symbols

Table 1: Perl Special Symbols

Symbol	Description
#	comment
..	range operator
$	Identifies a scalar variable
`@_`	array that holds parameters passed to function
@	Identifies an array variable
%	Identifies a hash variable
`$_`	Default scalar variable, holds current line when reading file
`$.`	Holds current line number when reading files
`$0`	Holds the name of the Perl script being run
`$!`	Holds the system error number and string
=>	Separates key and value pairs when defining a map
<>	Input operator defined by user
STDIN	Standard input filehandle
$< STDIN >$	STDIN assigned to <> reads input from keyboard
STDOUT	Standard output filehandle (the screen by default)



Whirlwind of Perl Symbols ©FrankMock 2016