PERL - SYNTAX OVERVIEW

Perl borrows syntax and concepts from many languages: awk, sed, C, Bourne Shell, Smalltalk, Lisp and even English. However, there are some definite differences between the languages. This chapter is designd to quickly get you up to speed on the syntax that is expected in Perl.

A Perl program consists of a sequence of declarations and statements, which run from the top to the bottom. Loops, subroutines, and other control structures allow you to jump around within the code. Every simple statement must end with a semicolon ; .

Perl is a free-form language: you can format and indent it however you like. Whitespace serves mostly to separate tokens, unlike languages like Python where it is an important part of the syntax, or Fortran where it is immaterial.

First Perl Program

Interactive Mode Programming

You can use Perl interpreter with **-e** option at command line, which lets you execute Perl statements from the command line. Let's try something at $ prompt as follows −

```
$perl -e 'print "Hello World\n"'
```

This execution will produce the following result −

```
Hello, world
```

Script Mode Programming

Assuming you are already on $ prompt, let's open a text file hello.pl using vi or vim editor and put the following lines inside your file.

```
#!/usr/bin/perl

# This will print "Hello, World"
print "Hello, world\n";
```

Here **/usr/bin/perl** is the actual perl interpreter binary. Before you execute your script, be sure to change the mode of the script file and give execution priviledge, generally a setting of 0755 works perfectly and finally you execute the above script as follows −

```
$chmod 0755 hello.pl
$./hello.pl
```

This execution will produce the following result −

```
Hello, world
```

You can use parentheses for functions arguments or omit them according to your personal taste. They are only required occasionally to clarify the issues of precedence. Following two statements produce the same result.

```
print("Hello, world\n");
print "Hello, world\n";
```

This will produce the following result −

```
Hello, world
Hello, world
```

Perl File Extension

A Perl script can be created inside of any normal simple-text editor program. There are several programs available for every type of platform. There are many programs designd for programmers available for download on the web.

As a Perl convention, a Perl file must be saved with a .pl or .PL file extension in order to be recognized as a functioning Perl script. File names can contain numbers, symbols, and letters but must not contain a space. Use an underscore _ in places of spaces.

Comments in Perl

Comments in any programming language are friends of developers. Comments can be used to make program user friendly and they are simply skipped by the interpreter without impacting the code functionality. For example, in the above program, a line starting with hash **#**is a comment.

Simply saying comments in Perl start with a hash symbol and run to the end of the line −

```
# This is a comment in perl
```

Lines starting with = are interpreted as the start of a section of embedded documentation *pod*, and all subsequent lines until the next =cut are ignored by the compiler. Following is the example −

```
#!/usr/bin/perl

# This is a single line comment
print "Hello, world\n";

=begin comment
This is all part of multiline comment.
You can use as many lines as you like
These comments will be ignored by the
compiler until the next =cut is encountered.
=cut
```

This will produce the following result −

```
Hello, world
```

Whitespaces in Perl

A Perl program does not care about whitespaces. Following program works perfectly fine −

```
#!/usr/bin/perl

print       "Hello, world\n";
```

This will produce the following result −

```
Hello, world
```

But if spaces are inside the quoted strings, then they would be printed as is. For example −

```
#!/usr/bin/perl

# This would print with a line break in the middle
print "Hello
          world\n";
```

This will produce the following result −

```
Hello
```

```
    world
```

All types of whitespace like spaces, tabs, newlines, etc. are equivalent for the interpreter when they are used outside of the quotes. A line containing only whitespace, possibly with a comment, is known as a blank line, and Perl totally ignores it.

Single and Double Quotes in Perl

You can use double quotes or single quotes around literal strings as follows −

```
#!/usr/bin/perl

print "Hello, world\n";
print 'Hello, world\n';
```

This will produce the following result −

```
Hello, world
Hello, world\n
```

There is an important difference in single and double quotes. Only double quotes **interpolate** variables and special characters such as newlines \n, where as single quote does not interpolate any variable or special character. Check below example where we are using $a as a variable to store a value and later printing that value −

```
#!/usr/bin/perl

$a = 10;
print "Value of a = $a\n";
print 'Value of a = $a\n';
```

This will produce the following result −

```
Value of a = 10
Value of a = $a\n
```

"Here" Documents

You can store or print multiline text with a great comfort. Even you can make use of variables inside the "here" document. Below is a simple syntax, check carefully there must be no space between the << and the identifier.

An identifier may be either a bare word or some quoted text like we used EOF below. If identifier is quoted, the type of quote you use determines the treatment of the text inside the here document, just as in regular quoting. An unquoted identifier works like double quotes.

```
#!/usr/bin/perl

$a = 10;
$var = <<"EOF";
This is the syntax for here document and it will continue
until it encounters a EOF in the first line.
This is case of double quote so variable value will be
interpolated. For example value of a = $a
EOF
print "$var\n";

$var = <<'EOF';
This is case of single quote so variable value will not be
interpolated. For example value of a = $a
EOF
print "$var\n";
```

This will produce the following result −

```
This is the syntax for here document and it will continue
until it encounters a EOF in the first line.
This is case of double quote so variable value will be
interpolated. For example value of a = 10

This is case of single quote so variable value will be
interpolated. For example value of a = $a
```

Escaping Characters

Perl uses the backslash (\) character to escape any type of character that might interfere with our code. Let's take one example where we want to print double quote and $ sign −

```
#!/usr/bin/perl

$result = "This is \"number\"";
print "$result\n";
print "\$result\n";
```

This will produce the following result −

```
This is "number"
$result
```

Perl Identifiers

A Perl identifier is a name used to identify a variable, function, class, module, or other object. A Perl variable name starts with either $, @ or % followed by zero or more letters, underscores, and digits (0 to 9).

Perl does not allow punctuation characters such as @, $, and % within identifiers. Perl is a **case sensitive** programming language. Thus **$Manpower** and **$manpower** are two different identifiers in Perl.