PERL Scripting

Practical Extraction and Report Language:
Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

Perl is an interpreted language, which means that your code can be run as is, without a compilation stage that creates a non portable executable program.

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 (;).

Command to check version of PERL installed:

nagveni@nagveni-H55M-S2V:~$ **perl -v**

This is perl 5, version 18, subversion 2 (v5.18.2) built for i686-linux-gnu-thread-multi-64int
(with 44 registered patches, see perl -V for more detail)

Copyright 1987-2013, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl". If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.

Simple command to execute perl command at command prompt

nagveni@nagveni-H55M-S2V:~$ **perl -e 'print "Hello World\n"'**
Hello World

Write a perl script to print Hello, world on screen

```
#!/bin/perl
# This will print "Hello, World"
print "Hello, world\n";
```

Save the above file as **hello.pl**

Execute the script:

nagveni@nagveni-H55M-S2V:~/Desktop$ ./hello.pl
bash: ./hello.pl: Permission denied

nagveni@nagveni-H55M-S2V:~/Desktop$ **chmod 777 hello.pl**
nagveni@nagveni-H55M-S2V:~/Desktop$ **./hello.pl**
Hello, world

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

```
#!/bin/perl

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

Run the program:
root@MUM084:~/Desktop# **perl hello.pl**
Hello, world
Hello, world\n

Only double quotes **interpolate** variables and special characters such as newlines \n, whereas single quote does not interpolate any variable or special character.

Perl has three basic data types:

1. scalars,
2. arrays of scalars, and
3. hashes of scalars, also known as associative arrays

Scalar

Scalars are simple variables. They are preceded by a dollar sign ($).

A scalar is either a number, a string, or a reference.

A reference is actually an address of a variable, which we will see in the upcoming chapters.

Arrays

Arrays are ordered lists of scalars that you access with a numeric index, which starts with 0.

They are preceded by an "at" sign (@).

Hashes

Hashes are unordered sets of key/value pairs that you access using the keys as subscripts.

They are preceded by a percent sign (%).

To refer to a single element of a hash, you will use the hash variable name followed by the "key" associated with the value in curly brackets.

Sample program to assign values to variables

```
#!/bin/perl

$age = 25;            # An integer assignment

$name = "John Paul";   # A string

$salary = 1445.50;     # A floating point

@ages = (25, 30, 40);

@names = ("John Paul", "Lisa", "Kumar");

print "Age = $age\n";

print "Name = $name\n";
```

```
print "Salary = $salary\n";

print "\$ages[0] = $ages[0]\n";

print "\$ages[1] = $ages[1]\n";

print "\$ages[2] = $ages[2]\n";

print "\$ages[-1] = $ages[-1]\n";

print "\$ages[-2] = $ages[-2]\n";

print "\$ages[-3] = $ages[-3]\n";

print "\$names[0] = $names[0]\n";

print "\$names[1] = $names[1]\n";

print "\$names[2] = $names[2]\n";

%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);

print "\$data{'John Paul'} = $data{'John Paul'}\n";

print "\$data{'Lisa'} = $data{'Lisa'}\n";

print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

Run the program:

```
root@MUM084:~/Desktop# perl var.pl
Age = 25
Name = John Paul
Salary = 1445.5
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
$ages[-1] = 40
$ages[-2] = 30
$ages[-3] = 25
$names[0] = John Paul
$names[1] = Lisa
$names[2] = Kumar
$data{'John Paul'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

Variable Context

Perl treats same variable differently based on Context, i.e., situation where a variable is being used.

Example:

```
#!/bin/perl

@names = ('John Paul', 'Lisa', 'Kumar');
```

```
@copy = @names;
$size = @names;

print "Given names are : @copy\n";
print "Number of names are : $size\n";
```

Output:

```
root@MUM084:~/Desktop# perl context.pl
Given names are : John Paul Lisa Kumar
Number of names are : 3
```

Program for simple operations on variables:

```
#!/bin/perl

$str = "hello" . "  world";       # Concatenates strings.
$num = 5+10;                # adds two numbers.
$mul = 4*5;                 # multiplies two numbers.
$mix = $str . $num;           # concatenates string and number.

print "str = $str\n";
print "num = $num\n";
print "mul = $mul\n";
print "mix = $mix\n";
```

Output:

```
root@MUM084:~/Desktop# perl op.pl
str = hello  world
num = 15
mul = 20
mix = hello  world15
```

Write a perl script to compute power of a given number

```
#!/bin/perl

print " Enter base number: ";
$base=<>;

print " Enter power to be calculated: ";
$power=<>;

$result=$base**$power;

print " Result is : " . $result;
```

Output:

```
root@MUM084:~/Desktop# perl read.pl
 Enter base number: 3
 Enter power to be calculated: 2
 Result is : 9
```

Write a perl script to check whether a number is prime or not.

```
#!/bin/perl

print " Enter  number to check: ";
$num=<STDIN>;
chop($num);
$flag=0;
for($i=2; $i<$num; $i++)
{
        if ($num%$i ==0)
        {
                $flag=1;

                break;
        }

}
if ($flag==1)
{
        print " $num   is not prime\n ";
}
else
{
        print "$num   is  prime\n";
}
```

Output:

```
root@MUM084:~/Desktop# perl prime.pl
 Enter  number to check: 5
5   is  prime
root@MUM084:~/Desktop# perl prime.pl
 Enter  number to check: 4
 4   is not prime
```