Intro to PERL:

``````````````

=> Its very powerful & adaptable language. Developed by Larry Wall for NASA in 1987.

=> PERL is "Practical Extraction and Report Language"

=> Perl is case-sensitive

=> In PERL every line ends with ; (semicolon).

=> Perl used in many fields like

- application development,

- test automation,

- website development,

- XML manipulation,

- configuration management,

- CGI programming [common gateway interface],

- Bioinformatics,

- system administration,

- game development,

- databases,

- GUI development,

- desktop apps and many more.

=> PERL uses the Perl DBI its standard database interface. It fully supports third-party databases integration of Oracle, Postgres, MySQL and others.

=> Perl is compatible with HTML, XML, JSON and other markup languages.

=> Perl supports procedural programming & OOPS.

=> There is no requirement of specifying data type in PERL.

=> It has three type - Scalar, Array, Hashes.

Installation:

``````````````

Manual:

```````

Download: https://www.perl.org/get.html

Automated:

``````````

# apt search perl | grep "^perl"

# apt info perl

# apt install perl -y

PERL in command mode:

``````````````````````[.pl]

=> The following arguments can be used while executing a Perl program.

# perl -v [ To check version ]

# perl script.pl

OR

#./script.pl [ chmod u+x script.pl ]

# perl -<argument> script.pl

w – argument shows a warning.

d – used for debugging.

c – compiles only do not run.

e – execute [ Basic perl functions ].

# perl -e 'print "Ulalalalala\n"'

[ -e : execute ]

# perl -E 'say "Ulalalalala"'

[ -E : Enable Adv features ]

[ say : Auto add newline \n ]

Interective console:

````````````````````

# perl

print "LOL\n";

^D

LOL

++++++++++++++++++++++++++++++++++++++++++++++++++

First line of PERL:

````````````````````````````[ .pl]

SHEBANG : #!

ENV : /usr/bin/perl

# which perl

/usr/bin/perl

# vim first.pl

#!/usr/bin/perl

print "Hello CDAC\n";

# perl first.pl

OR

# chmod u+x first.pl

# ./first.pl

Compile Only:

`````````````

# perl -c script.pl

script.pl syntax OK

Debug:

``````

# perl -d script.pl

main::(script.pl:2): print "Ulalalalalala\n";

DB<1> n [ To execute & go to next line of code ]

DB<1> h [ help ]

DB<1> q [ quit ]

+++++++++++++++++++++++++++++++++++++++++++++++++

Variables:

``````````

=> Variables are case-sensitive in PERL

=> There are three types of variables in PERL:

- Scalar

- Array

- Hashes

Scalar:

```````

=> Perl automatically identify whether is it int or string or float or a char or a paragraph or an entire web page, and both are scalar values for perl.

=> Scalar starts with $

=> To declare multiple variables at once : $var1, $var2, $var3

$a = 2

$a, $b = 1

($a, $b) = (1, 2)

=> If you want to return string without interpolation, you need to use '' [Single Quotes]

eg: print 'Data $100';

NOTE: it will print Data $100 as it is.

=> Perl has a special "undefined" value, written as "undef"

=> Declared but not assigned variables have the value undef

=> Default scalar have initial value as 0

- 0 [ZERO] is a number itself is FALSE

- An empty string '' (with single quotes) and the string '0' are false

- undef itself is false.

- ANYTHING ELSE is TRUE.

[+] https://pastebin.com/raw/diB0rgwF

#!/usr/bin/perl

$test = undef; # now $test var is undefined

$a = '';

$b = '0';

$c;

print "$test\n";

print "$a\n";

print "$b\n";

print "$c\n";

=> Float point values include the spacial values "inf" [infinity] and "NaN" [not a Number]. [Only as a single variable]\*\*\*

=> Every scalar must begin with $, like:

````````````````````````````````````````

$variable1 = "natasha"; # String

$variable2 = 99; # int

$variable3 = 1.19; # Float

----------------------

# vim script.pl

#!/usr/bin/perl

$variable1 = "natasha"; # String

$variable2 = 99; # int

$variable3 = 1.19; # Float

print "OUT1 = $variable1\n";

print "OUT2 = $variable2\n";

print "OUT3 = $variable3\n";

# perl script.pl

OUT1 = natasha

OUT2 = 99

OUT3 = 1.19

+++++++++++++++++++++++++++++++++++++++

Arithmetic ops:

````````````````

$a = 2;

$b = 3;

print $a + $b;

print $a - $b;

print $a / $b;

print $a \* $b;

print $a \*\* $b;

print $a % $b;

OR

#!/usr/bin/perl

$a = 2;

$b = 3;

print ($a + $b, "\n");

print ($a - $b, "\n");

print ($a / $b, "\n");

print ($a \* $b, "\n");

print ($a % $b, "\n");

+++++++++++++++++++++++++++++++++++++++

OS Command/System Call:

```````````````````````

#!/usr/bin/perl

$a = `pwd`;

print $a;

-----------------

OS Command output with exit status:

```````````````````````````````````

#!/usr/bin/perl

system("pwd");

OR

#!/usr/bin/perl

$a = system("pwd");

print $a;

-------------------------------------

User input in Perl:

```````````````````

- Perl use <> operator

- this operator reads a line along with the newline (\n) char.

#!/usr/bin/perl

print "Enter target: ";

$a = <>;

print "Target is : $a"

# perl script.pl

Enter target: NASA

NASA\n

NOTE: One more thing you should notice here is that <> also returned a newline character corresponding to the ENTER we pressed after giving input.

Perl chomp():

`````````````

- We can use 'chomp' function. It just removes the newline character from the end.

#!/usr/bin/perl

print "Enter target: ";

chomp($a = <>);

print "Target is $a, Lets hack now."

# perl script.pl

Enter target: NASA

Target is NASA, Lets hack now.