**What is Bash script?**

**Introduction**

*Bash(Bourne Again Shell)*

*Bash is the improved version of Sh ( bourne shell )*

*In the context of Bash scripts we are telling the Bash shell what it should do.*

*A Bash script is a plain text file which contains a series of commands also calls Operating System Service to do something*

*FOR EX: ls command list the files and folders in a directory.*

**what are they exactly?**

**Anything you can run normally on the command line can be put into a script and it will do exactly the same thing. Similarly, anything you can put into a script can also be run normally on the command line and it will do exactly the same thing.**

*It’s just that instead of typing them at the command line now we are entering them to a plain text file.*

*It is convention to give files that are Bash scripts an extension of .****sh***

**How they work?**

*there could be a several processes representing the same program running in memory at the same time.*

*When we are at the terminal we have a Bash process running in order to give us the Bash shell. If we start a script running it doesn’t actually run in that process but instead starts a new process to run inside*

**How do we run them?**

*Before we can execute a script it must have the execute permission set. This permission is generally not set by default, if you forget to grant this permission before running the script you‘ll just get an error message telling you as such and no harm will be done.*

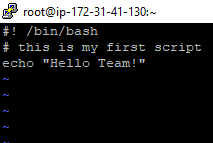
***Note : shorthand 755 is often used for scripts as it allows you the owner to write or modify the script and for everyone to execute the script.***

***In bellow picture***

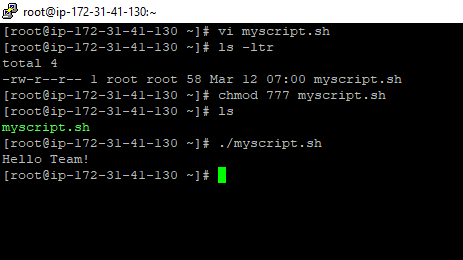
***Line 1****:is what’s referred to as the shebang(absolute path to the Bash interreter..ensures that Bash will be used to inerpret script, even if it is excuted under another shell.*

***Line 2:*** *this is comment. Anything after # is not excuted. for our refference only.*

***Line 3:*** *is the command echo which will print message to the screen. You can type this command yourself on the command line and it will behave exactly the same.*

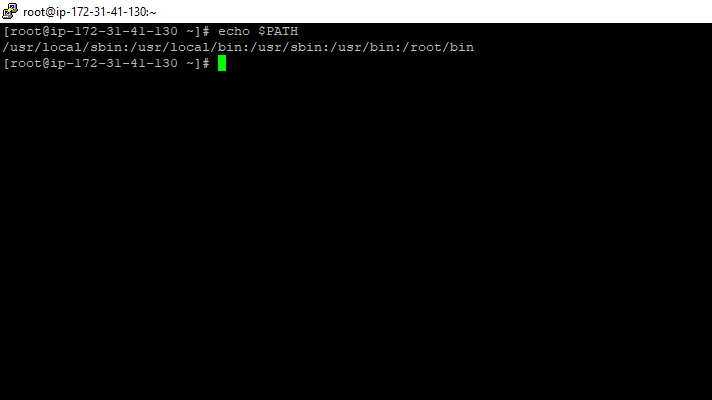
******

***Output:***

****Why the ./**

*When we run a normal command such a* ***ls*** *just type its name but when running the script above I put a* ***./*** *in front of it. When you just type a name on the command line Bash tries to find in a series* *of directories stored in a variable called* ***$PATH.*** *we can see the current value of this variable using the command* ***echo(.is reference to our current directory)***

*the directories are separated by* ***“:”***

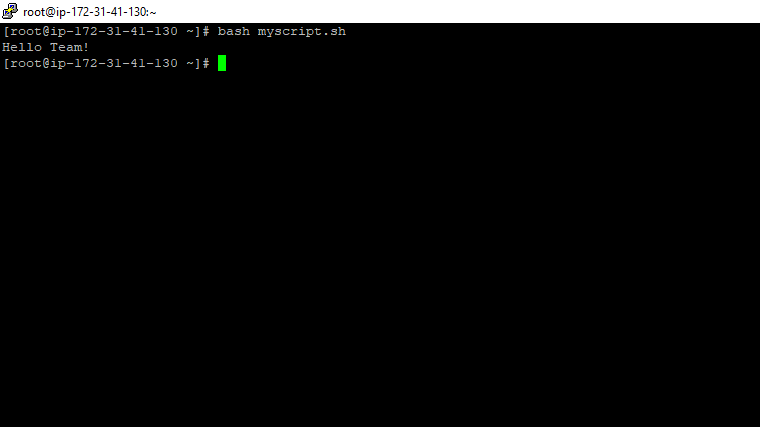
******

*it allows us to have several different version of program installed. We can control which one gets executed based on where it sits in our* ***$PATH.***

***#!bin/bash***

*If possible to leave out line with these shebang and still run he script but is unwise. if you are at a terminal and running the bash shell and you execute a script without a shebang then Bash will assume it is a Bash script. So this will only work assuming the user running the script is running it in a Bash shell.*

*You can also run Bash, passing the script as an argument*

******

## **Variables!**

* Temporary stores of information

*Variable is nothing but a container “it can store some data inside them”*

*These variables can be very useful for allowing us to manage and control the actions of our Bash Script. A variable is a temporary store for a piece of information. There are two actions we may perform for variables:*

* **Setting a value for a variable.**
* **Reading the value for a variable**.

*-> When referring to or reading a variable we place a $ sign before the variable name.*

*-> When setting a variable we leave out the $ sign.*

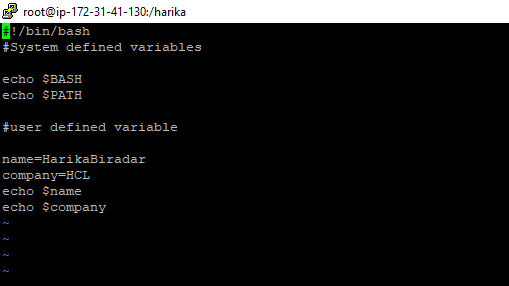
***Commonly variables are two types***

*1.System defined variables:*

*It’s created,defined&maintained by system(os)*

*2.user defined variables:*

*It’s created, defined, & maintained by us(users)*

**

## **Command line arguments:**

*You can pass command line arguments to bash shell script. These are helpful to make a script more dynamic.*

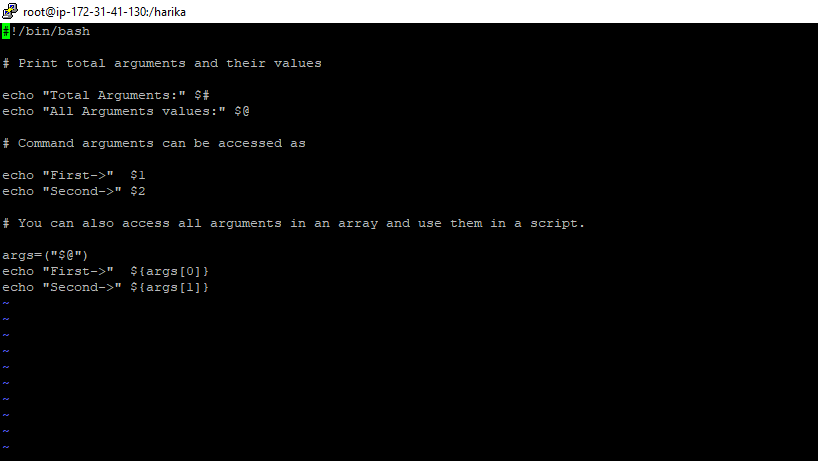
*we use the variables****$1****to represent the first command line argument,****$2****to represent the second command line argument and so on. These are automatically set by the system when we run our script so all we need to do is refer to them.*

**Syntax:**

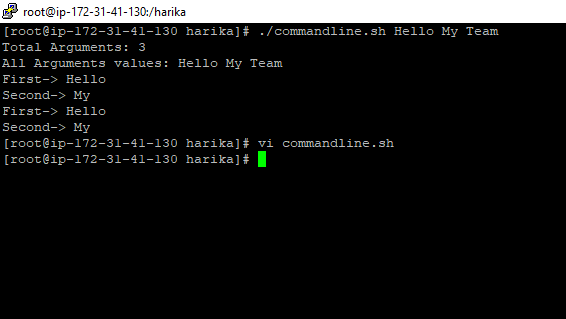
**$ ./myscript.sh ARG1 ARG2**

*Example:*

*Command line arguments can be passed just after script file name with space separated. If any argument have space, put them under single or double quote. Read below simple script.*

**

*Output:*

**

## **Other Special Variables**

* ***$0****- The name of the Bash script.*
* ***$1 - $9****- The first 9 arguments to the Bash script. (As mentioned above.)*
* ***$#****- How many arguments were passed to the Bash script.*
* ***$@****- All the arguments supplied to the Bash script.*
* ***$?****- The exit status of the most recently run process.*
* ***$$****- The process ID of the current script.*
* ***$USER****- The username of the user running the script.*
* ***$HOSTNAME****- The hostname of the machine the script is running on.*
* ***$SECONDS****- The number of seconds since the script was started.*
* ***$RANDOM****- Returns a different random number each time is it referred to.*
* ***$LINENO****- Returns the current line number in the Bash scrip*

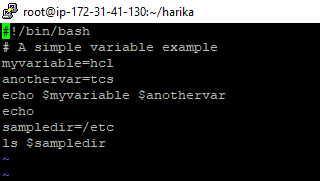
## **Setting Our Own Variables**

*As well as variables that are preset by the system, we may also set our own variables.*

**variable=value**

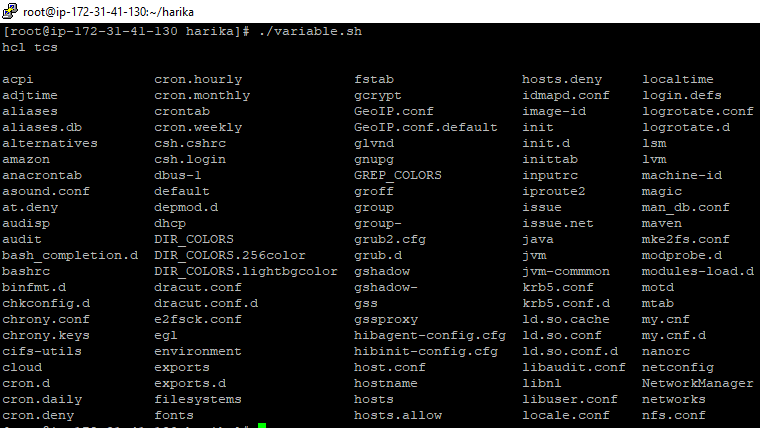
***Note*** *there is no space on either side of the equals ( = ) sign. We also leave off the $ sign from the beginning of the variable name when setting it.*

Example:



* ***Lines 3 and 4****- set the value of the two variables myvariable and anothervar.*
* ***Line 5****- run the command****echo****to check the variables have been set as intened.*
* ***Line 6****- run the command****echo****this time with no arguments. This is a good way to get a blank line on the screen to help space things out.*
* ***Line 7****- set another variable, this time as the path to a particular directory.*
* ***Line 8****- run the command****ls****substituting the value of the variable sampledir as its first command line argument.*

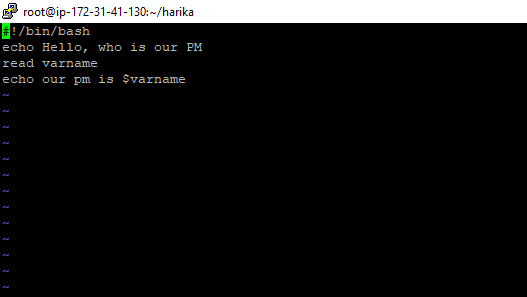
Output:



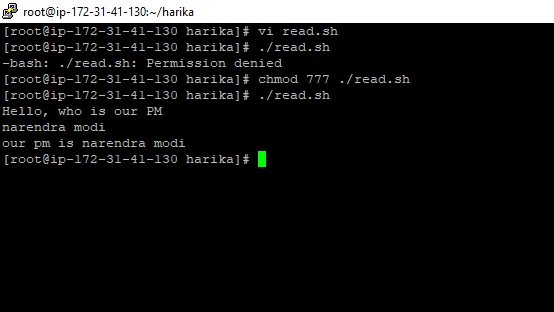
## **User Input!**

*If we would like to ask the user for input then we use a command called****read****. This command takes the input and will save it into a variable.*

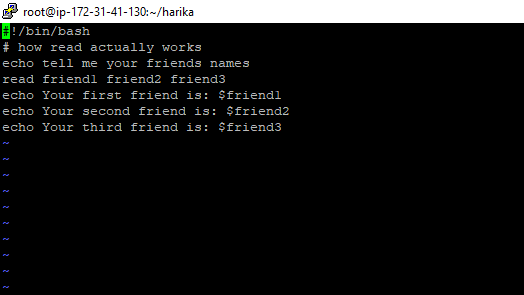
**Read var1**



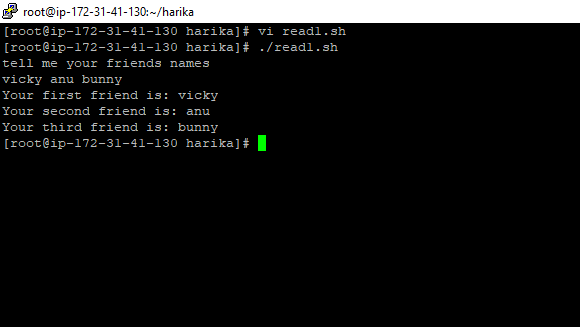
Output:



### **More variables:**



Output:



## **Arithmetic!**

It all adds up.

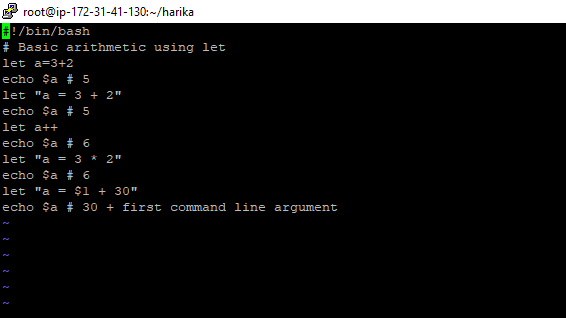
*There are several ways to go about arithmetic in Bash scripting.*

**Let:**

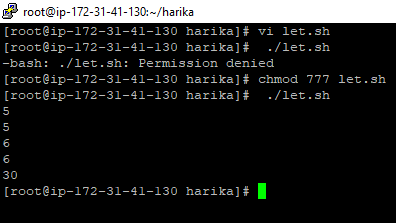
***let****is a builtin function of Bash that allows us to do simple arithmetic. It follows the basic format:*

**let <arithmetic expression>**

a simple example:

****

**Output:**

****

*Here is a table with some of the basic expressions you may perform.*

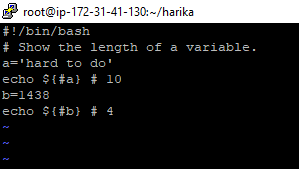
|  |  |
| --- | --- |
| **Operator** | **Operation** |
| **+, -, \\*, /** | *addition, subtraction, multiply, divide* |
| **var++** | *Increase the variable var by 1* |
| **var--** | *Decrease the variable var by 1* |
| **%** | *Modulus (Return the remainder after division*) |

**Lenth of a variable:**

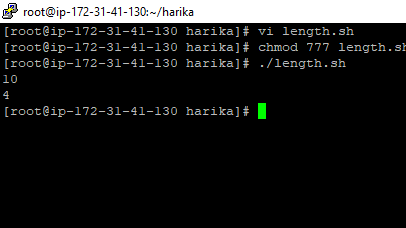
*If you want to find out the length of a variable (how many characters) you can do the following:*

**${#variable}**

example:

**

*Output:*

**

**If statements:**

Decisions, decisions.

* ***If****statements allow us to make decisions in our Bash scripts*
* *If statements, combined with*[*loops*](https://ryanstutorials.net/bash-scripting-tutorial/bash-loops.php)*allow us to make much more complex scripts which may solve larger tasks.*

*Basic if statements:*

*A basic if statement effectively says,****if****a particular test is true, then perform a given set of actions. If it is not true then don't perform those actions.*

***Syntax:***

if [<some test>]

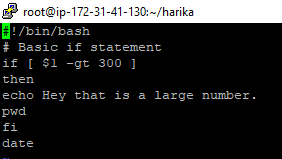
then

<commands>

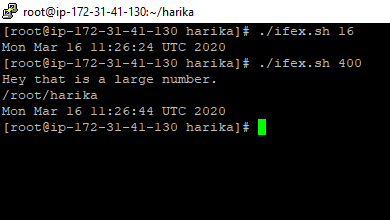
fi

*Anything between****then****and****fi****(if backwards) will be executed only if the test (between the square brackets) is true.*

Example:



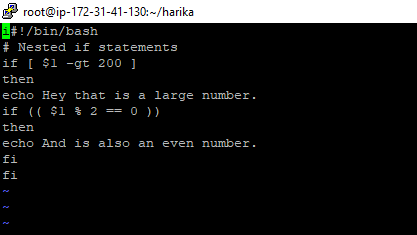
**Output:**



**Nested if statements:**

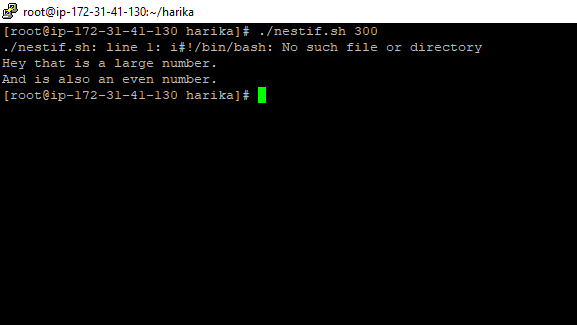
having the if statement inside the if statement is called nested if.

Ex:



* ***Line 3****- Perform the following, only if the first command line argument is greater than 100.*
* ***Line 7****- This is a light variation on the****if****statement. If we would like to check an expression then we may use the double brackets just like we did for*[*variables*](https://ryanstutorials.net/bash-scripting-tutorial/bash-variables.php)*.*
* ***Line 8****- Only gets run if both if statements are true.*

**Output:**



Ifelse:

*Sometimes we want to perform a certain set of actions if a statement is* true, and another set of actions if it is false. We can accommodate this with the **else** mechanism.

If [ <some test> ]

Then

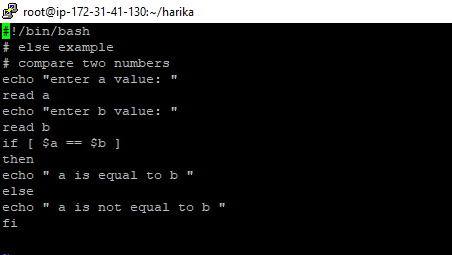
<commands>

Else

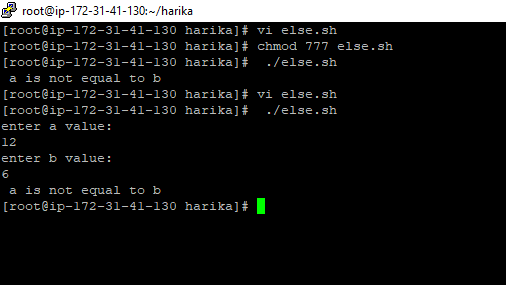
<other commands>

fi

example:



Output:



## **Loops!**

Round and round we go.

*Loops allow us to take a series of commands and keep re-running them until a particular situation is reached. They are useful for automating repetitive tasks.*

*There are 3 basic loop structures in Bash scripting.*

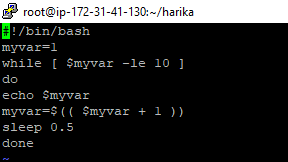
**While loops:**

*One of the easiest loops to work with is****while****loops. They say, while an expression is true, keep executing these lines of code.*

# while [ <some test> ] do <commands> done

*You'll notice that similar to*[***if****statements*](https://ryanstutorials.net/bash-scripting-tutorial/bash-if-statements.php)*the test is placed between square brackets [ ]*.

*Ex:*



Ouput:

# 

forloops:

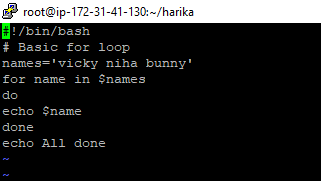
*The****for****loop is a little bit different to the previous two loops. What it does is say for each of the items in a given list, perform the given set of commands. It has the* ***syntax:***

# for var in <list> do <commands> done

*The for loop will take each item in the list (in order, one after the other), assign that item as the value of the variable****var****, execute the commands between do and done then go back to the top, grab the next item in the list and repeat over.*

*The list is defined as a series of strings, separated by spaces.*

*Ex:*

**

Ouput:

