SHELL/BASH SCRIPTING

* "Shell" is a program, which facilitates the interaction between the user and operating system (kernel). There are many shells available, like sh, bash, korn, csh, zsh...etc.
* "Shell" scripting is a way of automating things, in the form of collection of commands. The control of execution is steered by the predefined control statements.
* "Bash shell" is a flavor of shell. You can say, it’s a subset of "shell" scripting.
* "sh" is the original (Bourne) shell, having its root from the old Unix. "bash", is "Bourne Again SHell", which is rewritten "sh".
* Bash is the most widely used shell. It comes with Linux by default, having backward compatibility with sh (though sh is also there).
* For more information, do:  
  $~ man bash

**Change the Hostname:**

$ vi /etc/hostname

$ hostname linuxserver

**Execute Firstscript.sh and websetup.sh from the Repo**

**VARIABLES**

Temporary stores of information in memory

How do they work?

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 or using the value for a variable

To assign a variable we use = sign

VariableName=Value

To read/access the value of variable we use $VariableName

e.g.: $ NAME=” Syed”

echo $NAME

e.g.: $ PACKAGE=” apache2 wget unzip”

$ sudo apt install $PACKAGE -y

**Command Line Arguments**

A command line argument is a parameter that we can supply to our Bash script at execution. They allow a user to dynamically affect the actions your script will perform or the output it will generate. Like we use arguments with cp command. $ cp source/ /destination. Here source and destination are arguments for our cp command.

E.g.: ./script.sh argument1 argument2

To do this we use the variable $0 to $9.

$0 by default it will take the script name. We can use from $1 to $9.

Let’s see practical. We have to refer script called 5\_arg.sh

* Make the script executable using command: $ chmod +x 5\_arg.sh
* Now we can supply the arguments like: $ ./5\_arg.sh Linux Windows MacOS

Let’s refer next script 🡪 6\_arg\_website.sh

In this script we can supply URL and artifact name from command line.

**Some System Variables**

There are a few other variables that the system sets for you to use as well.

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

Let’s see some examples from above system variables

1. $?
   1. free -m
      1. It will display the memory utilization & command is correct
      2. Now if you run $? Command then it will display 0, which means that the last command was success.
   2. free -mjkloj
      1. It will fail, because there is no such command in Linux.
      2. Now if you run $? Command then it will display some random number (non 0). Which means that the last command was not success.
2. $USER
   1. It will display the current user who is executing the command
3. $HOSTNAME
   1. It will display the hostname of the machine
4. $RANDOM
   1. It will display random number each time when you run the command with this variable

**QUOTES**

We have two types of Quotes in Linux. i.e. single quote ( ‘) and double quote ( “ ).

Example 1:

$ SKILL=”DevOps”

$ echo $SKILL

This will display the value of the variable SKILL i.e. DevOps.

Even we can use single quote, it will work in the same way.

$ echo “I am learning $SKILL skill.”

Now you can try with single quote for the same sentence.

$ echo ‘I am learning $SKILL skill.’

It will lose the meaning of special character if you use single quote.

Example 2:

$ DELIVERY=”success”

$ echo “ Due to $DELIVERY delivery of the project our company got profit of $8 Million. ”

$ echo “ Due to $DELIVERY delivery of the project our company got profit of \$8 Million. ”

Question 1:

Single quote and double quote are same, just design differences.

1. True
2. False

Question 2:

Command substitution is method of storing OUTPUT of a command into variable.

1. True
2. False

**COMMAND SUBTITUTION**

Example 1:

$ uptime

$ UP=”uptime”

$ echo $UP

$ UP=`uptime`

$ LOGGED\_USERS=$(who)

$ echo $LOGGED\_USER

Example 2:

$ :wq

$ FREE\_RAM=$( free -m | grep Mem | awk ‘{print $4}’)

$ FREE\_RAM= `free -m | grep Mem | awk ‘{print $4}’`

$ echo “ Free RAM is $FREE\_RAM mb ”

**EXPORTING VARIABLES**

We know how to store a string/text into a variable but sometime we want to store output of a command to a variable. Like you may need to store ls command output to a variable. For this we use Command Substitution. There are two syntax for doing this

Example 1:

$ var1=foo

$ echo $var1

foo

$ export var1

Here export we will use to store the variable for all the child processes. Lets understand with below script. If we pass the variable name inside any script file, it will create a child process whenever you execute it. When you execute it, it will not call the variable value for the child process. In order to call for all the child processes then we have to export it. Using export command.

Refer 8\_testexport.sh shell script from the repo.

Now if you want to make it permanent, even though you don’t want to loose the value stored in the variable. Then you have to add the export command in the .bashrc file

$ ls -a

$ vi .bashrc

export SEASON=”Mansoon”

save and exit

Now you need to log out from the user and login again

$ echo $SEASON

This will change for only the user where you have modified the .bashrc file.

If you want to change it for all then you have to follow below steps

$ vim /etc/profile

Now you can place your export command at the end of the file and save and quit.

Now you need to logout from the user and login again to check.

**USER INPUT**

**Ask the user for the input.**

**Refer 9\_userinput.sh script from the Repo.**

**DECISION MAKING**

**If Statement**

**Script making decision**

**Basic If Statements**

If you use bash for scripting, you will undoubtedly have to use conditions a lot. Based on a condition you decide if you should execute some commands on the system or not. A basic 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. Follow the below format.

**if [ < some test > ]**

**then**

**< command >**

**fi**

**Refer 10\_if1.sh, 11\_if2.sh**

**Lets take an example:**

**$ ip addr show**

**$ ip addr show | grep -v LOOPBACK | grep -ic mtu**

*It will display how many time mtu occurred in the command output.*

**DECISION MAKING**

You can refer the script – 13\_monit.sh from the Repo.

Once you write the script we can create Cron Job, which basically means that you can schedule it.

$ crontab -e

# MM HH DOM mm DOW COMMAND

# 30 20 \*\* 1-5 COMMAND

**30 20 \* \* 1-5 ./12\_monit.sh &>> /var/log/monit\_apache2.log**

**LOOPS**

Execute it again and again and again…

Loops allows 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.

We are going to learn two types of loops

1. **FOR LOOP**
2. **WHILE LOOP**