**BASH SCRIPTING**

Writing the code into bash to automate some commands is called bash scripting like installing pkgs, running services, sleeping , cp, rm etc

**Pre-requisites**

1. Create an VM for bash scripting for practicing
2. run ‘**vagrant up scriptbox**’ in vagrant directory this will only start scriptbox VM via vagrantfile (geerlingguy/centos7) miniature
3. run -> ‘**vagrant scriptbox ssh**’ -> iteract as root user using -> ‘**sudo -i’**
4. change hostname ->open hostname file ‘**vi /etc/hostname’** -> change **localhost.localdomain to scriptbox** -->run **‘hostname scriptbox’** --> restart VM or logout and login again.

**FirstScript**

all scripts will be writen in directory /opt/scripts

run -> **‘mkdir /opt/scripts’**

install vim editor

run -> **‘yum install vim -y’**

create first file in /opt/scripts named firstscript.sh and open it in vim

run -> **‘cd /opt/scripts’**

run -> **‘vim firstscript.sh’** -> press ‘i’ for insert mode

**#!** => a shebang is the character sequence consisting of the characters number sign and exclamation mark at the beginning of a script. It is also called sharp-exclamation, sha-bang, hashbang, pound-bang, or hash-pling,

After that you can define your interpretor it can be

**#!/bin/bash** bash interpretor path

**#!/bin/python3**  python interpretor path

#!/bin/bash

### This Script is for checking System INFO ###

# Checking System Uptime

echo "---------------------------------------------"

echo "System Uptime is "

uptime

# Checking Memory Utilization

echo "---------------------------------------------"

echo "Memory Utilization "

free -m

# Checking Disk Utilization

echo "---------------------------------------------"

echo "Disk Utilization"

df -h

**Sample Script**

This Script is used deploying site using nginx loadbalancer via httpd service in it.

1. we will create the script file in /opt/scripts/websitedeployement.sh
2. we will use /tmp/website for storing zip file of template in it
3. Deploy by coping the files from /tmp/website/template\_folder to /var/www/html
4. Restart the httpd service
5. Now change mod of script in /opt/scripts -> chmod +x websitedeployment

#!/bin/bash

### A Script for Deploying an template on localhost at :80 using nginx loadbalancer (via httpd service)###

echo "Installing Dependencies"

echo "-------------------------------------------------------"

sudo yum install wget unzip httpd -y > /dev/null

echo "Service httpd start and enable"

echo "-------------------------------------------------------"

sudo systemctl start httpd

sudo systemctl enable httpd

### Creating directory in nested route so used -p in case the directory not present will create the directory itself

mkdir -p /tmp/website

cd /tmp/website

echo "Downloading Website Template"

echo "-------------------------------------------------------"

wget https://templatemo.com/tm-zip-files-2020/templatemo\_504\_page\_one.zip

unzip templatemo\_504\_page\_one.zip > /dev/null

### using -r from nested repositories and -f for force overwrite while copying ##

cp -rf ./templatemo\_504\_page\_one/\* /var/www/html/

echo "Restaring httpd Service"

echo "-------------------------------------------------------"

sudo systemctl restart httpd

sudo systemctl status httpd

**Variables**

We can use variable to reduce redundancy in scripting. Variables will be allocated temporarily in the memory and can be used easily with $ sign, in this example we will define variable with the scope of child bash so these variable will be destroyed with the script execution terminates.

#!/bin/bash

PACKAGES="wget unzip httpd"

URL="https://templatemo.com/tm-zip-files-2020/templatemo\_504\_page\_one.zip"

SVC="httpd"

ARTIFACT\_NAME="templatemo\_504\_page\_one"

TEMP\_DIR="/tmp/website"

### A Script for Deploying an template on localhost at :80 using nginx loadbalancer (via httpd service)###

echo "Installing Dependencies"

echo "-------------------------------------------------------"

sudo yum install $PACKAGES -y > /dev/null

echo "Service httpd start and enable"

echo "-------------------------------------------------------"

sudo systemctl start $SVC

sudo systemctl enable $SVC

### Creating directory in nested route so used -p in case the directory not present will create the directory itself

mkdir -p $TEMP\_DIR

cd $TEMP\_DIR

echo "Downloading Website Template"

echo "-------------------------------------------------------"

wget $URL

unzip $ARTIFACT\_NAME.zip > /dev/null

### using -r from nested repositories and -f for force overwrite while copying ##

cp -rf ./$ARTIFACT\_NAME/\* /var/www/html/

echo "Restaring httpd Service"

echo "-------------------------------------------------------"

sudo systemctl restart httpd

sudo systemctl status httpd

**Command Line Arguments**

Bash scripts receive command line arguments from the terminal. Several command line arguments are built-in, which take the user input to perform a specific function. The pre-defined command line arguments in Linux are:

|  |  |
| --- | --- |
| $0 | It shows the script’s name. |
| $1 | It shows the first command line argument. |
| $2 | It takes the second argument from the terminal. |
| $# | A script’s total number of arguments is displayed with this option. |
| $@ $\* | Both options provide the list of arguments given to the bash script. |
| $? | It displays the exit code for the execution of the script. |
| $$ | It shows the process ID (**PID**) for the script. |

1. Lets have the same script and replace URL with $1 and ARTIFACT\_NAME with $2
2. Replace all variables with respected command line arguments
3. run script with arguments
4. -> sudo **05\_CommandLine\_Arguments** <https://templatemo.com/tm-zip-files-2020/templatemo_504_page_one.zip> templatemo\_504\_page\_one

$2 (Argument)

$1 (Argument)

#!/bin/bash

PACKAGES="wget unzip httpd"

### URL="https://templatemo.com/tm-zip-files-2020/templatemo\_504\_page\_one.zip"

SVC="httpd"

### ARTIFACT\_NAME="templatemo\_504\_page\_one"

TEMP\_DIR="/tmp/website"

### A Script for Deploying an template on localhost at :80 using nginx loadbalancer (via httpd service)###

echo "Installing Dependencies"

echo "-------------------------------------------------------"

sudo yum install $PACKAGES -y > /dev/null

echo "Service httpd start and enable"

echo "-------------------------------------------------------"

sudo systemctl start $SVC

sudo systemctl enable $SVC

### Creating directory in nested route so used -p in case the directory not present will create the directory itself

mkdir -p $TEMP\_DIR

cd $TEMP\_DIR

echo "Downloading Website Template"

echo "-------------------------------------------------------"

wget $1

unzip $2.zip > /dev/null

### using -r from nested repositories and -f for force overwrite while copying ##

cp -rf ./$2/\* /var/www/html/

echo "Restaring httpd Service"

echo "-------------------------------------------------------"

sudo systemctl restart httpd

sudo systemctl status httpd

System Variables

There are many System Variables some of them are very important

1. $0 - The name of the bash script
2. $1 to $ 9 - The first 9 arguments of the bash script
3. $# - How many arguments were passed in bash script
4. $@ - All the arguments supplied to the bash script
5. $$ - The process ID of the Current Script
6. $USER - UserName of the user currently running the script
7. $HOSTNAME - Current Hostname of machine script running on
8. $? - Status of last executed command (0 means successfull ,1 means not successfull, and other)
9. $SECONDS - Number of seconds the script has started
10. $RANDOM - generates random number
11. $LINENO - return current line number in bash script

**Quotos**

Double Quotes = “”

Single Quotes = “”

Double Quotes understand the special chracters like using $ for variables

Single Quotes will treat every input as text even special chracter will be treated as text

Example

let set variables

1. AMOUNT = 12000
2. CURRENCY=”RUPESS”
3. **echo “Spendings is $AMOUNT $CURRENCY”** - > Spending is 12000 RUPESS
4. **echo ‘Spendings is $AMOUNT $CURRENCY’** -> Spendings is $AMOUNT $CURRENCY

To bypass or skip Special Chracter check use backslash \

Eg

1. OBJECT=”IT”
2. **echo “In $OBJECT Sector the total budget is \$12 Millions Dollar”**

In IT Sector the total budget is $12 Millions Dollar

**Command Substitution**

This is method is used to execute commands and save it’s output to an variable.

1. using back ticks -> `command`

#!/bin/bash

###This Script will get the system info ###

echo "Welcome $USER on $HOSTNAME"

ROOTFREE=$(df -h | grep /dev/sda1 | awk '{print $4}')

LOAD=$(uptime | awk '{print $9}''{print $10}''{print $11}')

MEMORYFREE=$(free -m | grep Mem | awk '{print $4}' )

echo "##############################################"

echo "System Checkup"

echo "---------------------------------"

echo "Current Average Load $LOAD"

echo "---------------------------------"

echo "Memory Free $MEMORYFREE"

echo "---------------------------------"

echo "Free Root Partition Size $ROOTFREE "

1. using dollar and brackets -> $(command)

Eg. Command is **free -m |grep Mem| awk '{print $4}**'

it will store its result in FREE Variable

-> FREE=`**free -m |grep Mem| awk '{print $4}**'`

-> **echo “Free Memory is $FREE Mb”** -> Free Memory is 216 Mb

**Variables**

Bash variables are very handy, just like variables in other programming languages, such as Java or C++. One major difference though is that they don’t need declaration; just assigning a value to the variable name creates a variable:

Example

MYVAR=1729

export MYVAR=1729

The first definition creates a variable named MYVAR and assigns it the value 1729. **This is a shell variable**.

**The second definition with the** export **command** is another way of defining a variable. It creates a variable named MYVAR, assigns it the value 1729, and marks it for export to all child processes created from that shell. **This is an environment variable**.

**The** export **attribute marks** MYVAR **for automatic export to the environment of the child processes created by the subsequent commands**:

export MYVAR=1729

echo $MYVAR

1729

bash # Open a new child shell

echo $MYVAR

1729

Now remember this is also temporary when you **logout or restart** the machine these environment **variables are gone.**

**So there are solutions**

1. **if environment variable must be stored for specific user then go to ~/.bashrc or ~/.bash\_profile**
2. **if environment variable must be stored for all users then go to /etc/profile**

Now just go simply to the end line and add your variable and export it

eg -> export FRUIT=”Mango”

**Now remember** for **root** user the priority for .**bashrc or .bash\_profile** is higher then **/etc/profile**

for other users **/etc/profile** declared variables has higher priority then .**basharc or .bash\_profile**