LinuxScripting.md

Introduction

Shell script or Bash script is a set of instructions to be peformed in a sequential manner or in a logical order subject to condtions.

Note: Bash is the default Shell in most Linux distributions, hence Bash scripting and Shell scripting or the same.

Syntax

Linux Shell script starts with the the below instruction informing that this program has to be executed by `bash`

```
#!/bin/bash
```

This is followed by the commands you want to be executed in order

Example

The below example prints "Hello World" on the screen

Let's create a file called `helloworld.sh` and add the below content

```
#!/bin/bash

echo "Hello World"
```

Use comments to make the shell script readable and easy to understand for debugging for further development
Any statement starting with `#` is a comment
Let's edit `helloworld.sh` add a comment in the script

```
#!/bin/bash

# The below statement prints Hello World
echo "Hello World"
```

Executing a Shell Script.

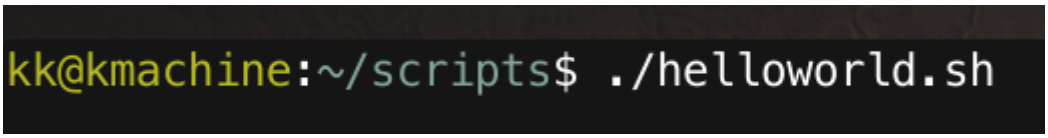
The user has to have execute rights on Shell script (the file in which the script is saved)
To make the shell script executable run one of the below commands

```
chmod +x helloworld.sh
```

Gives execute permission to every one
or

```
chmod 755 helloworld.sh
```

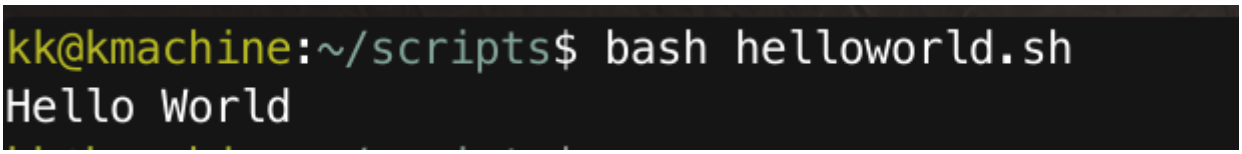
To execute the script go to the folder where the script is located and execute as below. Let's say the `helloworld.sh` in your `~/scripts` then you execute the program as below:



Make sure you give the `./` before the file name. This tells bash that you're executing a shell script located in the current folder and not some bash command.

Alternatively this can be executed as

```
bash helloworld.sh
```



Variables

Variables are a great way to create re-usable scripts.
Let's create a scirpt `dir.sh` which is going to create a directory and `cd` into that directory and create a sub-directory

```
#!/bin/bash
# Lets' create a directory named as mydirectory.

# This directory will be created in the same directory where the shell
script is being executed.

mkdir mydirectory

# Let's change into that directory

cd mydirectory
# Now Let's create a sub-directory named as `mysubdirectory`

mkdir mysubdirectory
```

Now let's use Variables to avoid repeating the name of the variable

```
#!/bin/bash
# Let's Create Variables called "maindir" & "subdir" and assign values to
it.
maindir="mydirectory"
subdir="mysubdirectory"

# Lets' create a directory named as $maindir

# This directory will be created in the same directory where the shell
script is being executed.

mkdir $maindir

# Let's change into that directory

cd $maindir
# Now Let's create a sub-directory named as $subdir

mkdir $subdir
```

Source Variables from a different file

There may be cases where the variables might have to be loaded from a different file. In such cases, we use `source` command to load variables

lets say there's a file `vars.txt` with the following content:

```
Web_VM_IP='10.0.10.25'
User_Name='ubuntu'
Private_Key='/opt/keys/id_rsa'
DB_VM_IP='10.0.20.178'
LB_IP='201.20.39.13'
```

To use the variables in a different script, we need to run `source vars.txt` and each of the variable can be substituted by using `$VARNAME`
`$Web_VM_IP` substitutes the value i.e., `10.0.10.25`
Now let's create a shell script which could use these variables:

```
#!/bin/bash
#Lets' load the variabels from vars.txt
source ./vars.txt
# Use the values in vars.txt and run commands
scp -i $Private_Key /opt/code $User_Name@$Web_VM_IP:~/
scp -i $Private_Key /opt/database $User_Name@$DB_VM_IP:~/
```

Conditionals

Just like in real world, one would have to run commands based on some conditions.
`If else` `If else else if` or common conditional statements used in bash script

The syntax for `if` statement is

```
if [[ condition ]]
then
    commands
fi
```

Every `if` block shold be closed with `fi`
Simple if else condition example to find if a number is an even number or odd number

```
#!/bin/bash
x=12

if [[ $x%2 == 0 ]]
then
    echo "$x is an even number"
else
    echo "$x is an odd number"
fi
```

Let's take a look at simple example where we install packages based on the distribution

```
#!/bin/bash

os_family=$(cat /etc/*se | grep -i ID_LIKE)

if [[ os_family == '*debian*' ]]
then
    sudo apt-get update && sudo apt-get install apache2 -y
elif [[ os_family == '*centos*' ]]
then
    sudo yum install httpd -y
else
    echo "Unknown operating system"
```

Loops

While performing some tasks you may have to iterate over a set of items such as an array(list) or rows in a file. To perform such a task we have `for` and `while` loops

For loop.

Let's take a simple example of a multiplication table, when you write a program for a multiplication table, you need to multiply a given number with a range of numbers (1 to 10) and print the output. This can be acheived through for loop.

Let's create a program to print out for a number `x` where `x` is provided by user.

```
#!/bin/bash

read -p "Enter the Number to get it's Table:   " x # This takes input from user and stores it in x
echo "Multiplication table for $x"
for i in {1..10}  # i's value changes from 1 to 10 in incremental order
do
    echo "$x X $i = $[$x*$i]" # This can also be written as $x $i = $(($x*$i))
done
```

Let's a take little comple example. We have a list of IP addreses of VMs and we need to run a certain command such as installing some packages in each of the VM.

The assumption is all of them have same username and use same public key for authentication and the computer is in the same network as the VMs and has SSH connectivity (Port 22 is open)

The IP Addresses are stored in a text file named as IPs.txt such as below

```
10.0.10.155
10.0.11.29
10.0.10.109
10.0.13.189
10.0.11.89
```

The command to execute a command on a remote machine is as below:

```
ssh -i <private_key> <username>@<ipAddress> <command_to_execute>
```

For example the command to install apache on a remote machine is as below

```
ssh -i privatekey.pem ubuntu@10.0.11.29 'sudo apt-get update && sudo apt-get install apache2'
```

Now the same command has to run in all the VMs whose IPs are listed in the file.

We shall use `for` loop and ssh command to install apache on mutliple machines

```
#!/bin/bash
user="ubuntu"
privatekey="opskey.pem"
File='.IPs.txt'

for IP in `cat IPs.txt`
do      echo "Installing apache on $IP"
    ssh -i ${privatekey} ${user}@${IP} 'sudo apt-get update && sudo apt-get
install apache2 -y'
done
```

When running ssh command on multiple machines, you may have to disable `host key checking` on the machine from which you're running this script. You can do this by creating a file name `config` in your `~.ssh/` folder with the following text

```
Host *
   StrictHostKeyChecking no
```

Conclusion

These are some of the basic examples of using Bash scripts. There may be many more uses cases and many more complex requirements but using these ideas, you could create scripts to acheive complex tasks.