**Linux – Day 7**

=> The most commonly used shell in real-time is known as BASH SHELL.

=> It is the default shell in any OS.

1. How to see the list of shells that are available?

$ cat /etc/shells

2. How to know the default shell in linux?

$ echo $SHELL

3. Writing a simple .sh file

$ vi demo.sh

Goto insert mode and write some commands

4. How to execute a shell script?

$ ./<ScriptFileName>

This will ask the execute permission

$ sh <ScriptFileName>

This will directly execute the file

YAML ---> Yet Another Markup Language (Ansible) - Ansible Playbooks

Script1

---

```
#!/bin/bash
echo "My name is Kastro"
echo "I'm a DevOps Engineer"
```

Script 2

---

```
#!/bin/bash
a=10
b=20
c=$((a + b))
echo $c
```

Script 3 - Getting input from users (add)

---

```
#!/bin/bash
echo "Enter First Number"
read a
echo "Enter Second Number"
read b
c=$((a + b))
echo "Sum of $a and $b = $c"
```

**Practice Scripts**

Connect to Instance using MobaXTerm -------->

vi vardemo.sh

```
#! /bin/bash
name=kastro
age=31
gender=male

echo $name $age $gender
```

----> sh vardemo.sh ----> You can see the response ----> I will give one more 'name' as shown below ----> vi vardemo.sh ---->

```
#! /bin/bash

name=kastro

age=31

gender=male


name=kiran


echo $name $age $gender
```

----> sh vardemo.sh ----> You can see the response ----> Now only 'kiran' is same for 'name'. It printing the second value of name, not the first value of name. This means, i'm able to modify the 'name' value. Now I dont want anyone to modify that. To do this, we will add 'readonly' infront of the 'name' ----> vi vardemo.sh ---->

```
#! /bin/bash

readonly name=kastro

age=31

gender=male


name=kiran


echo $name $age $gender
```



----> sh vardemo.sh ----> You can see the response as 'readonly variable'. It is not allowing me to modify the variable value after declaring the same. ---->

In the above script, I'm hardcoding the values. I dont want to hardcode the values, I want to pass the variables to shell script dynamically in the runtime. This kind is known as 'Command Line Arguements'

---

Command Line Arguments (cmd args)

---

=> The arguements which we will pass to the script file at the time of execution are known as 'Command Line Arguements'

=> Command Line Arguements are used to supply the values dynamically to the script file.

=> The data which we will pass to script file in the run time that data is known as command line arguments data.

=> We can send multiple command line arguments.

=> We can access the command line arguments in the script file as specified below;

# ------> it means how many cmd arguments we are sending

0 ------> it means what is the script file name

1 ------> it means we can access 'First Command Argument'

2 ------> it means we can access 'Second Command Argument'

3 ------> it means we can access 'Third Command Argument'

* ------> it means we can access 'All Command Arguments'



Lets work with cmd args ----> vi cmdargsdemo.sh ---->

```
#! /bin/bash

echo $#

echo $0

echo $1

echo $2

echo $*
```

----> sh cmdargsdemo.sh kastro trainer (Here 'kastro' and 'trainer' are inputs) ----> You can see the response as "2" "cmdargsdemo.sh" "kastro" "trainer" "kastro trainer" ----> Why we are seeing these responses? ----> cat cmdargsdemo.sh ----> 'echo $#' represents how many cmd arguments we are sending, hence it is giving response as 2 in the first line. 'echo $0' represents script file name, hence it is giving 'cmdargsdemo.sh' in the second line. Similarly the remaining lines.

----> Another example ----> sh cmdargsdemo.sh kastro trainer aws (Here 'kastro' 'trainer' and 'aws' are inputs) ----> You can see the response as "3" "cmdargsdemo.sh" "kastro" "trainer" "kastro trainer aws"

Within the script file, if I dont want to print specific arguments, how can we do that? If you give the # at the beginning of line, then it will consider as a 'comment' and hence it will not print/execute that command.

----> vi cmdargsdemo.sh ---->

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

echo $#

echo $0

echo $1

echo $2

#echo $*

----> sh cmdargsdemo.sh kastro trainer aws (Here 'kastro' 'trainer' and 'aws' are inputs) ----> It will only print $#, $0, $1, $2. You can see the response as "3" "cmdargsdemo.sh" "kastro" "trainer"

It has not printed last line i.e $* because we have commented that command using #

Note: To comment any single line we can use # at the beginning of the line

We can comment multiple lines also in the the script as shown below;

<<COMMENT

.....

COMMENT



Lets say I want to wait for some time to execute a specific command. To do this, we will use 'sleep' command followed by number of seconds, as shown below;

----> vi cmdargsdemo.sh ---->

#! /bin/bash

echo $#

echo $0

echo $1

sleep 30s

echo $2

#echo $*

----> sh cmdargsdemo.sh kastro trainer aws ----> It will wait for 30seconds to print the 2nd argument. The reason is I have stopped the script execution for 30 seconds by using 'sleep' command ----> Like this we can stop the execution for sometime.

---

CONDITIONAL STATEMENTS

---

=> In the shell scripting, If i want to execute some commands based on the conditions, we will use conditional statements.

=> 'IF ELSE' is a conditional statement.

Syntax:

```
if [condition]
then
        <statements>
else
        <statements>
```

=> If given statements are satisfied then 'if' statements will be executed, otherwise 'else' statements will be executed.

=> If you want to check multiple conditions, we will use 'if elif'



Syntax:

```
if [condition]
then
        <statements>
elif [condition]
then
        <statements>
else
         <statements>
fi
```

Note: 'fi' terminates the conditions. It is the end point for condition checks.

Demo on 'elif'

---

vi controlstatement.sh

```
#! /bin/bash
echo "Enter your favourite cricketer"
read CRICKETER

IF [ $CRICKETER == dhoni ]
then
        echo "Worldcup winning captain"
ELIF [ $CRICKETER == sachin ]
        echo "Opener in worldcup final match"
else
then
        echo "Worldcup 2011"
fi
```

(OR)

```
#!/bin/bash

echo "Enter your favourite cricketer"
read CRICKETER

if [ $CRICKETER == "dhoni" ]  # Added quotes for string comparison
then
  echo "Worldcup winning captain"
elif [ $CRICKETER == "sachin" ]  # Added quotes for string comparison
then
  echo "Opener in worldcup final match"
else
  echo "Worldcup 2011"
fi
```

----> sh controlstatement.sh ----> Type cricketers name and observer the responses.

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

FOR Loop Example

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

I want to execute the loop for 10 times.

vi forloop.sh ----->

```
#! /bin/bash

for (( i=1; i<=10; i++ ))
do
echo "$i"
done
```

-----> sh forloop.sh -----> You will see 20 times it got executed.



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

WHILE Loop Example

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

WHILE loop is a conditional based loop. Until a condition is satisfied, print the loop.

Lets say i want to print the numbers in reverse order.

vi whileloop.sh ----->

```
#! /bin/bash

i=10
while [ $i -ge 0 ]
do
echo "$i"
```

let i--;

done

-----> sh whileloop.sh -----> You will see 20 times it got executed.

Note:

In FOR loop, it is checking the range.

In WHILE loop, it is checking the condition first then it is going to execute the statement. Until the condition is getting satisfied, the loop will gets executed. Once the condition is failed, loop will terminate.

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

INFINITE Loop Example

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

If the condition is getting satisfied all the time, then it is known as infinite loop.



vi infiniteloop.sh ----->

#! /bin/bash

while true #'while true' means always the condition is getting satisfied

do

echo "This is an infinite loop"

done

-----> sh infiniteloop.sh -----> You will see continously loop is getting executed.

To stop the infinite loop, press CONTROL+C

==========

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

FUNCTIONS

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

Lets say I have a Linux based VM, in which an app is deployed.

On daily basis, I want to do activities like; stop the server, delete temp folder data, create new files, start the server, check server connectivity.

Instead of doing these activities manually, we can write a script to automate the above activities, by dividing into small activities and then we will create functions.

Function is a logical unit to perform some actions. Functions are used to divide big task into small tasks.