Table of contents

Linux basic operations day 2

- Agenda for today would be understanding and performing some of the basic linux operations that will help our understanding

Local DNS

- DNS is domain name service which helps us define a name for an ip address or url
- linux has /etc/hosts which is a DNS file and it is the source of truth for the machine
- Any request originating from the machine will first go to this DNS for resolution , if it doesnt find it then it will go to it's ISP's DNS.
- You can see the externam dns in /etc/resolv.conf in "nameserver property "
- For testing /etc/hosts , we will add few entries in it to test lie below

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost6 localhost6.localdomain6
10.0.0.1    www.google.com
10.0.1.25   db1
~
~
```

- Once above entries are added to /etc/hosts , try pinging the server names

```
[root@ip-172-31-87-91 ~]# vi /etc/hosts
[root@ip-172-31-87-91 ~]# ping www.google.com
PING www.google.com (10.0.0.1) 56(84) bytes of data.
^C
--- www.google.com ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2042ms

[root@ip-172-31-87-91 ~]# ping db1
PING db1 (10.0.1.25) 56(84) bytes of data.
^C
--- db1 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2041ms

[root@ip-172-31-87-91 ~]#
```

- As seen above , even though previously we were able to ping google.com, since we added the entry in local dns , it takes first precedence
- db1 name is also resolved with the ip that we have given

Creating our first shell script

- Shell script is nothing but a set of linux shell commands written in an order to execute a task
- We will first use our known commands to create our first shell script

- Let us create

```
sudo su -
vi install.sh
```

- Once the file opens paste below scripts inside the file

```
#!/bin/bash
yum install httpd -y
echo "this is my first script " >> /var/www/html/index.html
service httpd start
```

```
#once we have pasted the content , run below commands
chmod 777 install.sh
#above command will give the fule executable permissions
./install.sh
#above command will execute the script , once executed observe the output
# You can even store the output of the execution in a file using ./install.sh >>
output.txt
```

-

Accepting input from the user

- We may feel the need from time and time again to get inputs from the user
- We can fetch the arguments which are passed while executing the script
- for ex "./test.sh sample " here sample is an argument which is passed to test.sh
- Similarly one can pass multiple arguments

```
#!/bin/bash
echo "you have entered " $1
```

- While executing above script we need to pass 1 argument
- This argument will be printed in the output
- These arguments are also called as positional arguments . if there are 2 arguments which are passed we can use them with $2 ,$3 etc

```
#!/bin/bash
echo "first number you have entered is " $1
echo "second number you have entered is " $2
```

```
sum=$(($1 + $2))
echo "addition of numbers is  $sum"
```

- Above script expects 2 positional arguments to be passed while executing a script
- We can also prompt the user to give output that we need
- this output we can read in the script and operate on the same

```
#!/bin/bash
echo "Please enter your name "
read name
echo "hello $name , please enter city you are residing in "
read city
echo "$city is a good city "
```

- Above ways of getting input can be used to get away from hard coding the details
- Even though shell scripts are not as evolved as other programming languages considering these are meant for just scripting purposes , they do support looping as well
- As of today we will just see if else loop
- These are conditional loops which one can use execute simple condition based tasks

```
#!/bin/bash
echo "enter the number "
read num
if [ $((num%2)) -eq 0 ]
then
        echo "even number "
else
        echo "not even number "
fi
```

- as you can see above if loop begins with if and ends with fi
- things to be executed under then loop has to be properly intended
- same goes for else

```
echo "enter your access key "
read accesskey
echo "enter your secret access key "
read secret
echo "enter the bucket that you need backup in "
read bucket
```

```
echo "enter the backup folder full path "
read sourcepath
export AWS_ACCESS_KEY_ID=$accesskey
export AWS_SECRET_ACCESS_KEY=$secret
export AWS_DEFAULT_REGION=us-east-1
aws s3 cp $sourcepath s3://$bucket/backup/ --recursive
```

- above scripts puts the things we have learnt so far in perspective
- It will take path from user and then based on the inputs initiate the backup

```
#!/bin/bash
echo "enter your access key "
read accesskey
echo "enter your secret access key "
read secret
echo "enter the bucket that you need backup in "
read bucket
echo "enter the backup folder full path "
read sourcepath
export AWS_ACCESS_KEY_ID=$accesskey
export AWS_SECRET_ACCESS_KEY=$secret
export AWS_DEFAULT_REGION=us-east-1
aws s3 cp $sourcepath s3://$bucket/backup/ --recursive
if [ $? -eq 0 ]
then
        echo "backup successful"
else
        echo "backup failed , check your inputs "
fi
```

- To make the script better here , we are using "$?" operator . This stores the output of the previous command .
- If the output is 0 , previous command was successful , or else it failed