Linux Shell scripting

- you can run multiple command by separating them by ';'.
  eg :- ls ; pwd

- Types of command in linux shell :
  - ↳ Alias
  - ↳ Function
  - ↳ Shell built in
  - ↳ Keyword
  - ↳ File

- Linux will check for executables in 'PATH' environment.
  eg :- export PATH = $PATH: .

Hello world (hello.sh)

```
#! /bin/bash              } → first line known as shebang
echo "Hello World"        } → echo : STDOUT
exit 0                    } → exit : leave or exit script
```

- executing script :- bash hello.sh or ./hello.sh

- Arguments

$0 → name of script	$* → Refer to all argument
$1 → first argument	
${10} → if 2 or more digit are needed	
$# → argument count	

eg :-
```
#!/bin/bash
echo " Hello $1"
exit 0
```

• difference between " " and ' '.

```
echo "Hello $1"                          echo 'Hello $1'
 ./script.sh d                            ./script.sh d
→ Hello d                                → Hello $1
```

Variables

↳ User defined

```
#!/bin/bash                              array
name="Divyank"
age=22                                   #!/bin/bash
total=16.5                               myarr=(one two three four)
echo $name    (Prints Divyank)           echo ${myarr[0]}
echo $age     (Prints 22)                echo ${myarr[*]}
echo $total   (Prints 16.5)
```

```
#!/bin/bash
myarr=(one two three four)
unset myarr[1]    # This will remove the second element
unset myarr       # This will remove all the element
```

• Comment → #

Environment variable

```
$BASH_VERSION
$HOME
$PATH
$USER
```

Command Substitution

```
#!/bin/bash
cur_dir = 'pwd'
echo $cur_dir
```

```
#!/bin/bash
cur_dir = $(pwd)
echo $cur-dir
```

Script with read → read -p <prompt> <variable name>

```
#!/bin/bash
echo -n "May I ask your name :"
read
echo "Hello $REPLY"
exit 0
```

```
#!/bin/bash
read -p "May I ask your name:"n
echo "Hello $n"
exit 0
```

Limiting the number of entered characters

```
#!/bin/bash
read -p "May I ask your name :" name
echo "Hello $name"
read -n1 -p "press any key to exit"
echo      └→ no. of character
exit 0
```

Control the visibility of entered text

```
#!/bin/bash
read -sn1 -p "Enter a Character" n
echo "$n"  └→ no. of character to input
exit 0
```

list of commonly used options

- -a → list all items
- -c → get a count of all items
- -d → output directory
- -e → Expand items
- -f → Specify a file
- -h → show the help page
- -i → Ignore the character case
- -l → list a text
- -o → send output to a file
- -q → Keep silent; don't ask the user
- -r → process something recursively
- -s → use stealth mode
- -v → use verbose mode
- -x → specify an executable
- -y → accept without prompting me

Connecting to a server

Ping

```
#!/bin/bash
read -p "server:" server
ping -c3 $server 2>1 >/dev/null || echo "server dead"
```

SSH

```
#!/bin/bash
read -p "server :" server
read -p" Username :" user
ssh ${user}@ server
```

MySQL/MariaDB

```
#!/bin/bash
read -p "user :" user
read -sp "password :" password
echo
read -p " command : " cmd
read -p "Database : " db
mysql -u "user" -p $password $db -Be "$cmd"
```

Reading files

```
#!/bin/bash
while read line
do
echo $line
done < yourfile.txt
```

- Command line lists are two ~~different~~ or more statement joined using
  - > && : AND
  - > || : OR
- To read the exit variable '$?' → echo $?

Test

↳ It return true or false value
↳ It checks expression and variables

test Expression

$\underbrace{\text{test } {-a}}_{\text{and}}\ \overset{\text{or}}{-o}\ \underset{\text{not}}{!}$ Expression

test $USER = root
test ! $USER = root
-n : test if string has value
-z : Zero string

Testing integer

	Testing if
number 1 -eq number 2 :	number 1 is equal to number 2
number 1 -ge number2 :	number 1 is greater or equal to number 2
number 1 -gt number2 :	number 1 is greater than number 2
number1 -le number2 :	number 1 is smaller than or equal to number 2
number 1 -lt number 2 :	number 1 is smaller than number 2
number1 -ne number 2 :	number 1 is not equal to number 2

eg :- test 1 -ne 2

if and if-else

```
if condition; then
      statement
fi
```

```
#!/bin/bash
if [$# -lt 1]; then
    echo "Yes"
    exit1
fi
echo "No"
exit 0
```

```
if condition; then
      Statement 1
else
      Statement 2
fi
```

```
#!/bin/bash
if [$# - lt 1] ; then
     echo "1"
else
     echo "2"
fi
exit 0
```

Checking string

- S1 = S2 → equal
- S1 != S2 → not equal
- S1 \< S2 or S1 ~~$~~\> S2 → greater or less

- to check directory : -d eg if [-d mydir]
- to combine use && (AND) , || (OR)

elif

```
if condition; then
    Statement 1
elif condition; then
    Statement 2
else
    Statement 3
fi
exit 0
```

Switch case

```
case expression in
case 1)
    Statement 1
    Statement 2
;;
case 2)
    Statement 1
    Statement 2
;;
*)
    Statement 1
;;
esac
```

Loops

I) **For** (you can write it as in ())

```
#!/bin/bash
for var in one two three four; do
echo "Value : $var"
done
```

→ list of variable values

[illegible] - [illegible]

- **break** = [-d "$f"] && break

- **Continue** = [-d "$f"] || continue

II) while loop

```
COUNT = 10
while ((COUNT >= 0)); do
echo -e "$COUNT \c"
((COUNT--))
done ; echo
```

[reading file → read command]

Useful commands

> grep
> sed
> awk
> gawk

Functions

```
function-name () {

    <code to execute>
}
```

```
function <function-name> {

  <code to execute>
}
```

Any variable defined under a function is a global variable

To declare a local variable

```
myfunc () {

    local myvar=10

}
```

returning a value

```
return $var
```

Function can use recursion.