Shell Session 2

1. Math Expressions

In a Shell script, variables are by default treated as strings but not as numbers. This creates challenges for performing the math operation in shell scripts. However, there are few good commands which help us perform arithmetic operations in Shell.

+ add, - subtract, * multiply, / divide, ++ increment by 1, -- decrement by 1, % modulus

1.1 let – it helps us perform arithmetic operation through command line

`letmath.sh` contains all its variations of let command

```
#!/bin/bash
#Basic Arithmetic using Let command
echo "Basic Arithmetic using Let command"
let a=15+20
echo "a= 15+20 =" $a #35
let "b=29*20"
echo "b= 29*20 =" $a #580
let a++
echo "a++ =" $a #36
let "x=$1+30"
echo "x= $1+30 =" $x  #30 + first command line argument
let u=16/4
echo "u= 16/4 =" $u #4
let y=4/16
echo "y =4/16 =" $y #ceil the value to smallest number
```

1.2 Expr – it is like *let* command. However, instead of saving the result to a variable it prints the answer. Unlike *let* command you don't need to enclose the expression in quotes. You must provide spaces between the items of the expression. It is also common to use *expr* within command substitution to save the output into a variable.

`exprmath.sh` contains all its variations of expr command

```
#!/bin/bash

# expr with space does give the value as output
expr 23 + 29  #52

# expr with no spaces just prints the string
expr  23+29 #23+29

# expr with double quotes just prints the string
expr  "23+29" #23+29
```

```
# expr with escape character (backslash) will give us multiply
operator. '*' directly does work here.
expr 5 \* $1 # prints based on the argument passed during execution

# we get syntax error if we give * directly
expr 5 * $1  #expr: syntax error

# modulus operation - remainder when the first item is divided by
second
expr 21 % 2 #1

#expr with a substitute in order to display it in a variable
a=$( expr 43 - 5 )
echo $a #38
```

1.3 Double Parenthesis (( )) – It is a way to handle arithmetic operations by passing the expression within the parenthesis. `doublemath.sh` contains all the variations below:

```
#!/bin/bash

a=$(( 21 + 26 ))
echo $a #47

c=$(( 49-3))
echo $c #46

b=$(( a * 12))
echo $b #564

x=$(( $a  / 2 ))
echo $x #23

(( c++ ))
echo $c #47

#adding 3 values to c
(( c += 3 ))
echo $c #50
```

1.4 bc (Basic Calculator) – This is a plugin in BASH to run basic calculator application. Use the command apt-get install `bc` to install the basic calculator. By just running `bc` command, the basic calculator will be opened in interactive mode. We can run the `bc` command in non-interactive mode using `|` symbol. `bc` helps us to handle the floating-point value-based calculations.

Non-Interactive mode - `echo "579*2.5" | bc`

Below is an example script for using `bc` command within a script. Floating points are handled without any issues using `bc` command.

```
#!/bin/bash
clear
echo
read -p "Enter hours worked:" hoursworked
read -p "Enter hourly wage:" hourlywage
echo
grosspay= `echo "$hoursworked*$hourlywage" | bc
echo "Gross pay is: \$" $grosspay
```

1.5 Conversion Numbers between bases – We sometime try to convert the numbers from base-2 to base10 or base-16 or binary based on our need while working with numeric data. Using BASH, below file `binary.sh` can be used to convert our input base-2 IP Address value to Binary values
i/p: 192.168.1.1
o/p: 11000000 10101000 1 1

```
#!/bin/bash
clear
echo
read -p "Enter IPv4 address:" ip_address
for ((i=1; i<5;i++))
do
     octet[$i] = `echo $ip_address | cut -f$i -d "."`
     #array of i value to fetch the values from IPv4 address
     #Your Input Example IP 192.168.1.1
     #cut command to cut IP address into parts using $i counter
     # d is a delimiter for cut. "." is our delimiter here.
done

for ((i=1;i<5;i++))
do
     current_bin_val=`bc <<< "obase=2; ${octate[$i]}"`
     #Logic to convert current octet to binary
     #bc command to convert octate arrange $i value
     full_bin_val=$full_bin_val" "$current_bin_val
     #add binary values in loop
done
echo
echo "Binary Value of IP address:" $full_bin_val
#binary value output for your input IP 11000000 10101000 1 1
```

1.6 test Command - This command checks file type and compare values on command line and returns the success flag based on true or false condition.

`test 27 -gt 2 && echo "Yes" –` Returns Yes, as 27 is greater than 2

`test 27 -lt 2 && echo "Yes" –` Returns nothing as we have not defined false condition
`test 27 -lt 2 && echo "Yes" || echo "No" –` Returns No, as 27 is not less than 2

`test 25 -eq 25 && echo "Yes" || echo "No"` – Returns Yes, as 25 is equal to 25

`test 15 -ne 10 && echo Yes || echo No –` Returns Yes, as 15 is not equal to 10

`test -f /etc/resolv.conf && echo "File /etc/resolv.conf found." || echo "File /etc/resolv.conf not found." –` Returns relevant echo based on the presence of respective file.

2. Working with Arrays

The Array is a type of a variable which is the collection of multiple items of same type. The counter in the array starts with zero – 0. Declared as `depts = ("cse" "cnd" "ece")` as `depts` being the name of the array. To fetch the output - `echo $depts`. Below are the variations on working with arrays.

`echo $depts` – will return 0th value or first value in the array i.e. *cse*

`echo ${depts[*]}` – returns all as we have * in the [ ]

`echo ${depts[2]}` – returns third value i.e. *ece*

`echo ${depts[3]}` – returns nothing as there is no value for `dept[3]`

`depts[3] = "research"` – to insert a fourth element in dept array

`echo ${#depts[*]}` – returns count of elements in the array i.e. returns *4*

`echo ${depts[*]:2:1}` – skips first two elements and only prints the next one value i.e. returns *ece*

`echo ${depts[*]/research/phd}` – replaces "research" element in the array with "phd" on runtime. However, `echo ${depts[*]}` still shows research as the change is only for runtime.

`depts2 = ` echo ${depts[*]/research/phd}`` - to store the changed `depts` array into `depts2` array. `echo ${depts2[*]}` returns values with phd in fourth position.

`echo sort <<<"${depts[*]}"` - triple redirection to sort a text in array argument but not in a file argument.

3. If and Else

If and Else helps us handle and judge the conditions on data, we manage. Below are few examples on Syntax. Highlighted are the required syntax constructs of if & else.

```
#!/bin/bash
if test $USER := "root"
#to open if. Used $USER env variable to check user
then
{
   clear
   echo "Your are not root"
}
else # to open else
{
     clear
     echo "Your are root"
}
fi #close the if
```

Running If and Else on command like as below. As soon as you close if using fi keywords, the if and else logic will get executed directly.

```
root@ubuntu1:/scripts# numdays=400
root@ubuntu1:/scripts# echo $numdays
400
root@ubuntu1:/scripts# if test "$numdays" -gt 100
>then
>{
>echo "Over Budget!"
>}
>fi
Over budget!
root@ubuntu1:/scripts#
```

4. For Loop

For loop is used to repeat a task in a specific number of times we would like to process something. Below `for.sh` is an example to demo a `FOR` loop with conduction using `IN` keyword to list of .sh files in each sub directory based on the total size of the file. The below script will print the size of all the .sh files in a list and then gives us the sum of the total size. Highlighted are the required syntax constructs for the For Loop.

```
#!/bin/bash
clear
#declaring variables so that can be used
totalsize=0
currentfilesize=0

#currentfile is a temporary variable used for counter
for currentfile in /scripts/*.sh
do
  currentsize=`ls -l $currentfile | tr -s " " | cut -f5 -d " "`
#Gets file size and cuts it with space delimiter
#tr command to squeeze the space on the file name spaces
  let totalsize=$totalsize+$currentsize
#calculate the totalsize
  echo $currentsize
done
echo
echo "Total space used by Script files is:" $totalsize
```

Below is an example for `FOR` loop using array. Array is a variable contains multiple items of the same type. For examples `cities=("Chennai","Hyderabad")` is an array with two items. `echo $cities` will print the first value in the cities array i.e. Chennai. Highlighted are the required syntax constructs for the For Loop. @ value in the array to call each element based on the increment value of i.

```
echo ${cities[0]} – return Chennai

echo $(cities[1]} – will return "Hyderabad"

root@Ubuntu:/scripts# for ((i=0;i<${#cities[@]};i++));do
>echo ${cities[$i]}
>done
Chennai
Hyderabad
```

5. While Loop

Unlike a `FOR` loop, when we use `WHILE` loop – we don't necessarily know how many times we are going through the loop. `While.sh` is an example script that prompts the user to select an option, so while the user doesn't choose to exit and live in the loop. Highlighted are the required syntax constructs of WHILE Loop.

```
#!/bin/bash
choice="0"
while (( $choice != "1"))
do
      clear
      echo
      echo "Please select and Option"
      echo
      echo "1 - Exit"
      echo
      read choice
done
```

The script waits for the input choice from the user. Until the input is given as 1 the loop is not broken. The loop will execute initially as the choice value is zero by default and waits for the input, will be in loop unless the user gives the value as 1.

6. Break and Continue in side Loop

Depending on the code, we might wish to break out of loop or continue the loop so that we go back to loop statements and run them again. Below is an example for break and continue. Unless we type ***exit*** , we will not be able to break the loop, else it still continues to run the same *while* again and again.

```
#!/bin/bash
while true
do
      clear
      echo "To leave, Type exit"
      echo
      read -p "What say you?" choice
      if test $choice = "exit"
      then
      {
            break
      }
      else
      {
            continue
      }
done
```

7. Case Statements

Case statements help us to test a value of something for multiple possible items, instead of using bunch of If-else, we can use Case statements. Below is an example `case.sh` for your reference. Highlighted are the constructs used in the case statements.

```
#!/bin/bash
clear
echo ”City Type”
echo
read -p “Enter your City:” city
case $city in
      “Hyderabad”) city_type =”Tier 2”;;
      # Pipe Symbol is a OR command
      “Bengaluru” | “Mumbai”)city_type=”Tier 1”;;
      “New Delhi”) city_type=”Capital”;;
      # here star is to consider anything else
      *)clear; echo “Invalid City $city”;exit;;
esac #case in reverse to close the case statement
clear
echo “City Type for $city has been set to $city_type”
echo
```