Shell Scripting

A real programming language, complete with variables, control structures, and so forth

1.alert the system that a shell script is being started with and write any commands
#! /bin/bash
pwd
ls

2.Save it as a file: 2.sh
Change Permission to make it executable
root@ubuntu:~/Desktop/shell_scripts# chmod a+x 2.sh
3. Run
root@ubuntu:~/Desktop/shell_scripts# ./2.sh
/root/Desktop/shell_scripts
1.script 2.sh

4. Run another way
root@ubuntu:~/Desktop/shell_scripts# sh 2.sh
/root/Desktop/shell_scripts
1.script 2.sh

////////////////////////////////Variables/////////////////////////////////////
Create a file and save it with a name 2.sh

```
#! /bin/bash

var1="Hello"    #string or character array

var2="World"   #string or character array

echo $var1    #printing
echo $var2
echo $var1+$var2    #Print and conatenate with +

echo $var1,$var2  # Prints Comma separated output

var3="1"  #This is a string not number

var4=21;  #This is a number

echo $var3 +$var4   # + will concatenate and produce string

PERSON="Fred"
```

```
echo "Hello, $PERSON"
```

To Execute:

```
root@ubuntu:~/Desktop/shell_scripts# chmod a+x 2.sh
root@ubuntu:~/Desktop/shell_scripts# sh 2.sh
Hello
World
Hello+World
Hello,World
1 +21
Hello, Fred
```

//////////Print Statements//////////////////////

Code : save file as script_print.sh

```
#!/bin/bash
var1="Hello";
var2=200
var3="Coding is fun"

printf "%s\n" $var1
printf "%d\n" $var2
printf "%s\t" $var3
printf "\n"
printf "%s" $var3
printf "\n"
printf "%s %s %s \n" $var3
printf "%s\n" "Hello, World!" # all are strings with %s
printf "%d\n" "100" "200"   # all are integers with %d
printf "%f\n" "0.09" "67.00"  # all are floating points with %f
```

To Execute

```
root@ubuntu:~/Desktop/shell_scripts# chmod a+x script_print.sh
root@ubuntu:~/Desktop/shell_scripts# sh script_print.sh
Hello
200
Coding          is        fun
Codingisfun
Coding is fun
Hello, World!
100
200
0.090000
67.000000
```

////////////////////////////Take Input From User//////////////////////

```
#!/bin/bash
echo -n "Enter number1: "   #Note that "-n" causes it to keep the cursor on the same line
read num1   #Taking input1 from user

echo -n "Enter number2: "
read num2   #Taking input2 from user

#sum=$num1+$num2 #This will give concatenation of two strings
#printf "sum=%d\n" $num1+$num2; #This will give concatenation of two strings
```

#When we surround an arithmetic expression with the double parentheses, the shell will perform arithmetic expansion. $((Expression))

```
sum=$(($num1+$num2))
echo $sum  #printf "%d\n $sum";
```

Code2

```
#! /bin/bash
echo -n "Enter A: "
read A
echo -n "Enter B: "
read B

echo "first number + second number = $((A + B))"
echo "first number - second number = $((A - B))"
echo "first number * second number = $((A * B))"
echo "first number / second number = $((A / B))"
echo "first number % second number = $((A % B))"
echo "first to power second number= $((A ** B))"
```

```
root@ubuntu:~/Desktop/shell_scripts# ./script5.sh
Enter A: 4
Enter B: 2
first number + second number = 6
first number - second number = 2
first number * second number = 8
first number / second number = 2
first number % second number = 0
first to power second number= 16
```

//////Short Hand Arithmetic Expression////
Expr supports integer and string only

```
root@ubuntu:~/Desktop/shell_scripts# expr 1+2
1+2
root@ubuntu:~/Desktop/shell_scripts# expr 1 + 2
3
root@ubuntu:~/Desktop/shell_scripts# echo $(( 1 + 2 ))
3
root@ubuntu:~/Desktop/shell_scripts# echo $[ 1 + 2 ]
3

root@ubuntu:~/Desktop/shell_scripts# echo ' 1 + 2'
 1 + 2
root@ubuntu:~/Desktop/shell_scripts# echo 1 + 2
1 + 2
root@ubuntu:~/Desktop/shell_scripts# echo 1 + 2 | bc -l
3
root@ubuntu:~/Desktop/shell_scripts#

root@ubuntu:~/Desktop/shell_scripts# echo 4 / 3 | bc -l
1.33333333333333333333

`echo $A / $B | bc -l`
```

Code 4:

```
#! /bin/bash
#Using `expr
echo -n "Enter A: "
read A
echo -n "Enter B: "
read B

echo  "first number + second number = " `expr $A + $B`
echo "first number - second number = " `expr $A - $B`
echo "first number * second number = " `expr $A \* $B`
echo "first number / second number = " `expr $A / $B`
printf "%f \n" `echo  $A / $B  | bc -l`
echo "first number % second number = " `expr $A % $B`

#There is no power expression
#Length of String
x="Hello"
len=`expr length $x`
echo "Length of Expression= $len"
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# ./script8.sh
Enter A: 2
Enter B: 3
first number + second number =  5
first number - second number =  -1
first number * second number =  6
first number / second number =  0
0.666667
first number % second number =  2
Length of Expression= 5
```

////////////////Command Substitution///////

```
#! /bin/bash
#Command Substitution: Back Ticks and Brace Expansion

#It's often quite important to capture the results of some command in a variable for use by your shell
#script- Back ticks are really useful for this sort of thing because they provide you with an inline method
#for executing a command and retrieving the results before the rest of your script executes

path=`pwd`
echo $path

#use double quotation marks if you do want variables to be substituted, and commands to be executed.
#brace expansion uses the format: $(command), where command is any valid Unix command

path= "$(pwd)"
echo $path

file_count=`ls | wc -l`
echo $file_count


file_count=$(ls | wc -l)
echo $file_count

file_count="$(ls | wc -l)"
echo $file_count

#echo $(cat Chap*) ##Prints string of contents
files= echo $(cat Chap*) #Save the string in files
echo $files
```

Output:
root@ubuntu:~/Desktop/shell_scripts# ./script17.sh
/root/Desktop/shell_scripts
./script17.sh: line 14: /root/Desktop/shell_scripts: Is a directory
/root/Desktop/shell_scripts
25
25
25
Hello 1 Hello 2 Hello 3 Hello 4 Hello 5

root@ubuntu:~/Desktop/shell_scripts#

//Special Characters for Shell
Some Special Characters for Shell

$ character represents the *process ID number*, or PID,

? The previous command's exit status. **echo $? Output 0**
exit status of 0 if successful
1 if they were unsuccessful.
$ The PID of the current shell process. **echo $$ Output 3618**
- Options invoked at start-up of the current shell. **echo $- Output 0**

! The PID of the last command that was run in the background. **echo $!**
0 The filename of the current script. **echo $0 bash**
1-9 The first through ninth command-line arguments given when the current script was invoked: $1 is the value of the first command-line argument, $2 the value of the second, and so forth.
root@ubuntu:~/Desktop/shell_scripts# echo $1

root@ubuntu:~/Desktop/shell_scripts# echo $1, $2, $3
, ,
_ The last argument given to the most recently invoked command before this one
echo $- Output 0

//////////////////////Command Line Argments //////////////////////
Code 1
```
#! /bin/bash
#Command Line Arguments
echo "You Have Entered $1 $2"
```

Output:
root@ubuntu:~/Desktop/shell_scripts# chmod a+x script11.sh

root@ubuntu:~/Desktop/shell_scripts# ./script11.sh
You Have Entered

```
root@ubuntu:~/Desktop/shell_scripts# ./script11.sh 23 25
You Have Entered 23 25
root@ubuntu:~/Desktop/shell_scripts# ./script11.sh Another Day
You Have Entered Another Day
root@ubuntu:~/Desktop/shell_scripts# ./script11.sh "Tom" "Jerry"
You Have Entered Tom Jerry
```

//////////////////////////Flow Control ////////////////////

```
if [some_condition]
then
//Statements
fi
```

Code1

```
#! /bin/bash

#Mind the Gap. You have to give spaces after if [

num1="Red"
num2=200
num3=900
if [ $num1=="Red" ]
then
echo $num1
fi

if [ $num2==200 ]
then
echo $num2
fi
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# ./if_one.sh
Red
200
```

Decision Control and Relational Operators

Code1:

```
#! /bin/bash

echo -n "Enter an A: "
read A

echo -n "Enter an B: "
read B
```

```
if [ $A -gt $B ]
then
        printf "A %d is Greater than B %d\n" $A $B
else
        printf "B %d is Greater than A %d\n" $B $A

fi

##TO REMEMBER###
#$x -eq $y      $x is equal to $y
#$x -ne $y      $x is not equal to $y
#$x -lt $y      $x < $y
#$x -gt $y      $x > $y
#$x -le $y      $x <= $y
#$x -ge $y      $x >= $y
```

OUTPUT:

```
root@ubuntu:~/Desktop/shell_scripts# ./script4.sh
Enter an A: 2
Enter an B: 5
B 5 is Greater than A 2
```

Code2:

```
#! /bin/bash
echo -n "Enter A: "
read A

echo -n "Enter B: "
read B


if [ $A -eq $B ]
then
        echo "A is equal to B"

elif [ $A -ne $B ]
then
        echo "A is not equal to B"

elif [ $A -lt  $B ]
then
        echo "A is less than B"
```

```
elif [ $A -gt $B ]
then
        echo "A is greater than B"

elif [ $A -le $B ]
then
        echo "A is less than equal B"

elif [ $A -le $B ]
then
        echo "A is less than equal B"


elif [ $A -ge $B ]
then
        echo "A is greater than equal B"

else
        echo "Cannot Determine"

fi

#$x -eq $y      $x is equal to $y
#$x -ne $y      $x is not equal to $y
#$x -lt $y      $x < $y
#$x -gt $y      $x > $y
#$x -le $y      $x <= $y
#$x -ge $y      $x >= $y
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# ./script6.sh
Enter A: 100
Enter B: 100
A is equal to B
```

Code 3:

```
#! /bin/bash
#Comparing ASCII Values of Strings
echo -n "Enter an A: "
read A

echo -n "Enter an B: "
```

```
read B

if [ $A != $B ]  #Example of Not Equal
then
        printf "A %s is not equal B %s\n " $A $B
else
  echo "We got a Match!"

fi

#= string a = string b
#!= string a != string b
# > string a > string b
# < string a < string b
```

Output

root@ubuntu:~/Desktop/shell_scripts# ./script7.sh
Enter an A: Andy
Enter an B: Andy
We got a Match!

Code 4: Nested If

```
#! /bin/bash
today="Sunday"
rain="no"

if ( test $today == "Sunday" )   #test keyword has same purpose as [ bracket bt used with ( first bracket
then
        if (test $rain == "no" )
        then
          echo "We play football"
        else
          echo "We wait"
        fi
fi
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# chmod 111 script9.sh
root@ubuntu:~/Desktop/shell_scripts# ./script9.sh
We play football
```

//////////////////////Switch Case//////////////////

```
case expression/variable in
    pattern1)
        do-something-here
        ;; #break statement
    pattern2)
        do-something-here
        ;;
esac
```

Code1:

```
#! /bin/bash

case 1 in

1)
echo "1"
;;
2)
echo "1"
;;
*)
echo "Default"
esac
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# chmod a+x script12.sh
root@ubuntu:~/Desktop/shell_scripts# ./script12.sh
1
root@ubuntu:~/Desktop/shell_scripts#
```

Code2:

```
#! /bin/bash

echo -n "Enter number1 "
read num1
echo -n "Enter number2 "
read num2

echo -n "Enter operator add sub mult div "
read oper

case $oper in
        "add" )
                echo $[ $num1 + $num2 ]
        ;;
```

```
        "sub" )
                echo $[ $num1 - $num2 ]
        ;;

        "mult" )
                echo $[ $num1 * $num2 ]
        ;;

        "div" )
                echo $[ $num1 / $num2 ]
        ;;

        * )
                echo "Default"
esac
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# ./script13.sh
Enter number1 2
Enter number2 3
Enter operator add sub multiply div add
5
root@ubuntu:~/Desktop/shell_scripts# ./script13.sh
Enter number1 2
Enter number2 1
Enter operator add sub multiply div sub
1
root@ubuntu:~/Desktop/shell_scripts# ./script13.sh
Enter number1 3
Enter number2 4
Enter operator add sub multiply div multiply
Default
root@ubuntu:~/Desktop/shell_scripts# ./script13.sh
Enter number1 2
Enter number2 3
Enter operator add sub mult div mult
6
root@ubuntu:~/Desktop/shell_scripts# ./script13.sh
Enter number1 4
Enter number2 2
Enter operator add sub mult div div
2
```

////////////While Loop ////////////

Code 1

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

```
x=5
while [ $x -ge 0 ]
 do
            echo $x
            let x--
 done


##Second While Loop

i=0
while [ $i -le 2 ]
do
  echo "i= $i"
  let i++   #$i=((i+1)), ((i=i+1)) ((i++))
done

##Third While loop

#infinite loop
while :      # while true
do
printf "Hello\n"
done
```

//////////////For Loop //////////////////

```
#! /bin/bash

#For Loop 1
for x in {1,2,3,4,5}
do
 echo "Loop 1: $x "
done

#For Loop 2
for x in {1..5}
do
 echo "Loop 2: $x "
done
#For Loop 3
for ((i=0; i<3; i++))
 do
  echo "Loop 3: $i"
 done

# Create files Chap1 to Chap5 using for loop
for x in {1..5}
```

```
do
 echo "Hello $x" > Chap$x
done

#Read from these files

for x in {1..5}
do
 cat Chap$x | while read line
  do
   echo $line
  done
done
```

Output:

```
root@ubuntu:~/Desktop/shell_scripts# ./script15.sh
Loop 1: 1
Loop 1: 2
Loop 1: 3
Loop 1: 4
Loop 1: 5
Loop 2: 1
Loop 2: 2
Loop 2: 3
Loop 2: 4
Loop 2: 5
Loop 3: 0
Loop 3: 1
Loop 3: 2
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
root@ubuntu:~/Desktop/shell_scripts#
```

////////////////Array////////////////

```
#! /bin/bash
#Arrays
files=(100 200 300 400 500)

echo "First Element: " ${files[0]}
echo "All Elements :  ${files[*]} "
echo "All Elements :  ${files[@]} "
echo "All Elements : " ${files[@]:0}
```

```
echo "All Elements : " ${files[*]:0}
echo "Elements : " ${files[*]:1}
echo "Elements : " ${files[*]:2}
echo "Elements : " ${files[*]:3}
echo "Elements : " ${files[*]:4}
#Length
echo Length:${#files[*]}

#Looping the elements
for((x=0;x<5;x++))
 do
  echo ${files[x]}
done

#Array Chap1,2,... fed to cat to see the contents
cat $(ls Chap*)
```

```
root@ubuntu:~/Desktop/shell_scripts# ./script16.sh
First Element:  100
All Elements :  100 200 300 400 500
All Elements :  100 200 300 400 500
All Elements :  100 200 300 400 500
All Elements :  100 200 300 400 500
Elements :  200 300 400 500
Elements :  300 400 500
Elements :  400 500
Elements :  500
Length:5
100
200
300
400
500
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
root@ubuntu:~/Desktop/shell_scripts#
```

```
//////////////////Function Call///////
#! /bin/bash

#Function Defination
myFunction(){
cal
}

myFunction  #Function Call

##Function Taking Arguments
echo -n "Enter Variable1: "
read var1
echo -n "Enter Variable2: "
read var2

myFunction2(){

expr $var1 + $var2


}

##Function Call
myFunction2


##Function3
myFunction3(){
echo "Inside Func3"
#var3=` echo $var1 + $var2 | bc -l `
var3=$(($var1 + $var2))
return $var3
}

myFunction3
echo $?  #Capture value returnd by last command

root@ubuntu:~/Desktop/shell_scripts# ./script10.sh
```

```
    August 2022
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

Enter Variable1: 2
Enter Variable2: 3
5
Inside Func3
5
```