Shell Scripting & Bash Scripting:

shell is the interface between the user and the kernal.

Shell scripting involves writing a series of commands for the command-line interpreter (shell) to automate tasks and perform various operations. Bash (Bourne Again SHell) is one of the most popular and widely used shells in Unix and Linux-like operating systems

shebang: #!/bin/bash

A shebang is a character sequence at the beginning of a script or a data file that indicates the interpreter to be used for executing the script. It consists of the number sign (#) followed by an exclamation mark (!). The shebang is also known as a "hashbang," "pound-bang," or "hash-exclam.

most common shell types:

* bash shell
* csh shell
* ksh shell
* zsh shell

Basic commands:

- cat /etc/shells --to check all available shells
- echo $SHELL --what script is used by the terminal
- cat > filename.sh -- shortcut to create a file using cat and directly start modifying it, use ctrl+d to get out of edit mode
- to comment out add a '#' at the begining of the line

types of variables:

* local variable--created by us

* environment variable-- system already have it and we can retrive it

* predefined variable--existing variables

- program to demonstrate if-else condition :

```
#!/bin/bash
echo "Program to check whether you are eligible to vote using if-else"
read -p "enter your age" age
if [ $age >= 18 ]
then
    echo "you are eligible to vote "
else
    echo "you are not eligible to vote"
fi
```

(OR)

```
#!/bin/bash

echo "Program to check whether you are eligible to vote using if-else"
read -p "enter your age" age

if [ $age >= 18 ]
then
    echo "you are eligible to vote. "

elif [ $age = 18 ]
then
    echo "please apply for voter id card."

else
    echo "you are not eligible to vote."
fi
```

Arithmetic/Relational Operations:

here:

-gt --greater than

-ge --greater than equal to

-lt --less than

-le --less than equal to

-eq --equal to

-ne --not equal to

-a --and operator

-o --or operator

- Program to demonstrate arithemetic operation:

```
#!/bin/bash

Echo "Demonstration of Arithmetic Operation"

read -p "Enter the first number :" n1

read -p "Enter the secound number :" n2

echo "Addition :" $((n1+n2))
```

```
echo "Subtraction :" $((n1-n2))

echo "Multiplication :" $((n1*n2))

echo "Division :" $((n1/n2))

echo "Modulus :" $((n1%n2))
```

- Program to demonstrate relational operation:

```
Echo "Demonstration of Relational Operation"

read -p "Enter the first number :" a1

read -p "Enter the secound number :" a2

if [ $a1 -gt $a2 ]
then
    echo "a1 > a2"
else
    echo "a1 < a2"
```

- program to check whether the number is even or odd:

```
#!/bin/bash
echo -n "Enter a number"
read n

if [`expr $n %2` == 0]
then
  echo "$n is an even number"
else
  echo "$n is an odd number"
fi
```

types of loops:

* for --itration till it meets the condition (i=1,i<=10,i++)

syntax:

```
for((initialization; condition; increment/decrement))
                    (OR)
for i in 1 2 3 4
do
echo"Itration:" $i
done
```

- program to demonstrate for loop:

```
#!/bin/bash
a=1
for(( i=a;i<=10;i++ ))
echo "for loop iteration $a"
done
```

* while --it works until the condition is true and when the condition is false it comes out of loops

syntax:

```
while [condition]
do
//code
done
```

- program to demonstrate while loop:

```
#!/bin/bash
echo "demo while loop"
 a=0
 while [ $a -le 10 ]
  do
    echo "while loop iteration $a"
    a=$((a+1))
    done
```

* until --it works until the condition is false and when the condition is true it comes out of loops

Syntax:

```
until [condition]
      do
      //code
      done
```

- program to demonstrate until loop

```
#!/bin/bash
a=20
 until [ $a -le 1]
  do
echo" until loop iteration $a"
a=$((a-1))
done
```

switch case:

- program to demonstrate switch case

```
#!/bin/bash
echo -n "Are you a student? [yes or no]: "
read response
case $response in

        "Y" | "y" | "YES" | "Yes" | "yes")
                echo -n "Yes, I am a student.";;

        "N" | "n" | "No" | "NO" | "no" | "nO")
                echo -n "No, I am not a student.";;

        *) echo -n "Invalid input"
           ;;
esac
```

Local & Global:

In programming, variables can be classified as either local or global based on their scope, which refers to the region of the code where the variable can be accessed

Local Variables:

Scope: Local variables are declared inside a specific block of code, such as a function or a loop. They are only accessible within that block or the block's nested blocks.

Global Variables:

Scope: Global variables are declared outside of any function or block, making them accessible throughout the entire program, including inside functions.

Functions & Nested Functions:

a function is a named block of code that performs a specific task or a set of tasks. Functions are used to break down a program into smaller, more manageable pieces, promote code reuse, and enhance readability

syntax: functionName(){

code

}

functionName

- program to print helloworld using functions

```
#!/bin/bash

echo "Demonstrate functions in shell"
hello(){
    echo "welcome to linux"
}

hello
```

- program to demonstrate addition using functions

```
#!/bin/bash
echo "Demonstrate addition using shell functions"
add_fn(){
        echo "calculate sum of two numbers"
        sum=$((num1+num2))
        echo "sum of $num1 and $num2 is:"$sum
}
read -p "Enter the first number:" num1
read -p "Enter the secound number:" num2
add_fn num1 num2
```

- program to demonstrate nested function

```
#!/bin/bash
demo1(){
      echo "this is message is from demo1 function"
}
demo2(){
       echo "this is demo2"
       demo
}

demo2
```

- program to demonstrate addition and subtraction using nested function

```
#!/bin/bash
sum(){
    a=num1
    b=num2
    c=$(($a+$b))

    echo "sum of $a and $b is:" $c

sub(){
     c=$(($a-$b))
    echo "sub of $a and $b is:" $c
}
sub

}

read -p "Enter the first number: " num1
read -p "Enter the secound number: " num2

sum
```

Break & continue:

break --it breaks the execution of a loop.

continue --it skips the execution on some certain condition.

- program to demonstrate break

```
#!/bin/bash

for((i=o;i<=10;i++)){

  if [ $i -eq 7 ]
  then
  break
  else
  echo $i
  fi
}
```

- program to demonstrate continue

```
#!/bin/bash
for((i=o;i<=10;i++)){
  if [ $i -eq 7 ]
  then
  continue
  else
  echo $i
  fi
}
```

- program to demonstrate array:

```
#!/bin/bash

# "Array is a data structure which is used for storing multiple values in a single variable"

nameList=(roy sam john mike brad)

      #  0   1   2    3    4

# Array indexing: ' it is used to access the elements present in an array '
```

```
echo "First Index:" ${nameList[0]}

echo "Second Index:" ${nameList[1]}

echo "Third Index:" ${nameList[2]}

echo "Fourth Index:" ${nameList[3]}

echo "Fifth Index:" ${nameList[4]}

echo "All array elements:" ${nameList[*]}

echo "All array elements:" ${nameList[@]}
```

Automation:

- program to aad multiple directories through automation

```
#!/bin/bash

read -p "Enter a directoryName to create a directory" dirName

for((i=0;i<=4;i++)){
  mkdir $dirName &i
}
```

- program to aad multiple files through automation

```
#!/bin/bash

read -p "Enter a fileName to create a directory" fileName

for((i=0;i<=4;i++)){
  touch $fileName &i
}
```

- program to add user

```
#!/bin/bash
read -p "Enter the user name: " username
read -p "Enter the password: " password

 sudo useradd -m -s /bin/bash $username
echo "$username:$password" | sudo chpasswd
```

- program to create group, users, directory,changing mod and ownership

first create a "users.txt" (labuser1,labuser2,labuser3,labuser4)
now create a 'sh' file adduser.sh

```
#!/bin/bash

echo "Demonstrate user creation using shell script"
for i in `users.txt`
do
    sudo groupadd demogroup
    sudo mkdir -p /home/$i
    sudo useradd -d /home/$i $i
    sudo usermod -aG demogroup $i
```

```
    sudo chown -R $i:$i /home/$i
done
echo "all users in /etc/passwd"
cat /etc/passwd
```