EX NO: 2

18/08/2021

**CASE STUDY ON SHELL SCRIPTING**

**AIM:**

To do a case study on shell scripting and know the usage of commands.

**COMMAND:**

**1) Basics of shell scripting**

**a) echo**

**Code:**

echo "Hello world"

**Output:**



**b) read**

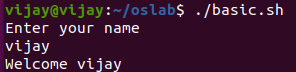
**Code:**

echo "Enter your name"

read name

echo "Welcome $name"

**Output:**



**Code:**

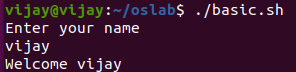
echo "Enter your name"

read name

lit="Welcome"

echo $lit $name

**Output:**



**c) Variable:**

  A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data. A variable is nothing more than a pointer to the actual data. The shell enables you to create, assign, and delete variables. The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (\_).

**Code:**

name="Vijay"

age="19"

echo "My name is: $name"

echo "My age is: $age"

**Output:**



**d) Array:**

Shell supports a different type of variable called an **array variable**. This can hold multiple values at the same time. Arrays provide a method of grouping a set of variables. Instead of creating a new name for each variable that is required, you can use a single array variable that stores all the other variables. All the naming rules discussed for Shell Variables would be applicable while naming arrays.

**Code:**

ARR[0]="bash"

ARR[1]="Vijay"

ARR[2]="OS"

echo "${ARR[1]}"

**Output:**



**2) Operators with example:**

There are five basic operators in bash/shell scripting:

* Arithmetic operators
* Relational operators
* Boolean operators
* String operators
* File Test operators

**a) Arithmetic operators:**

**+ (Addition)** - Adds value on either side of the operator.

- **(Subtraction)** - Subtracts right hand operand from left hand operand.

**\* (Multiplication)** - Multiplies values on either side of the operator.

**/ (Division)** - Divides left hand operand by right hand operand.

**% (Modulus)** - Divides left hand operand by right hand operand and returns remainder.

**= (Assignment) -** Assigns right operand in left operand.

**= = (Equality) -** Compares two numbers, if both are same then returns true.

**! = (Inequality) -** Compares two numbers, if both are different then returns true

**Code:**

echo "Enter the value of a"

read a

echo "Enter the value of b"

read b

echo "Addition of a and b"

echo `expr $a + $b`

echo "Subtraction of a and b"

echo `expr $a - $b`

echo "Multiplication of a and b"

echo `expr $a \\* $b`

echo "Division of a and b"

echo `expr $a / $b`

echo "Modulus of a and b"

echo `expr $a % $b`

c=$a

echo "The value assigned to c is:$c"

if [ $a == $b ]

then

echo "a is equal to b"

fi

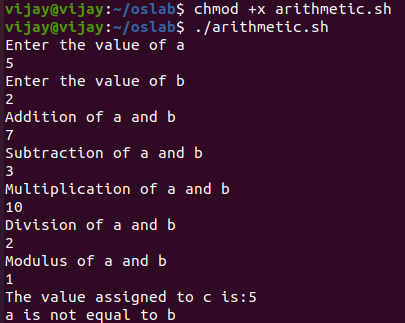
if [ $a != $b ]

then

echo "a is not equal to b"

fi

**Output:**

****

**b) Relational operators:**

**-eq** checks if the values of the two operands are equal or not; if yes, then the condition becomes true.

**-ne** checks if the values of the two operands are equal or not; if the values are not equal, then the condition becomes true.

**-lt** checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.

**-gt** checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.

**-le** checks if the value of left operand is less than or equal to value of right operand; if yes, then the condition becomes true.

**-ge** checks if the value of left operand is greater than or equal to value of right operand; if yes, then the condition becomes true.

**Code:**

echo "Enter the value of a"

read a

echo "Enter the value of b"

read b

if [ $a -eq $b ]

then

echo "a is equal to b"

else

echo "a is not equal to b"

fi

if [ $a -ne $b ]

then

echo "a is not equal to b"

else

echo "a is equal to b"

fi

if [ $a -lt $b ]

then

echo "a is less than b"

else

echo "a is not less than b"

fi

if [ $a -gt $b ]

then

echo "a is greater than b"

else

echo "a is not greater than b"

fi

if [ $a -le $b ]

then

echo "a is less than or equal to b"

else

echo "a is greater than b"

fi

if [ $a -ge $b ]

then

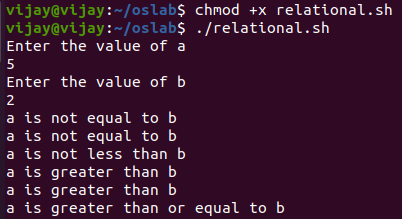
echo "a is greater than or equal to b"

else

echo "a is less than b"

fi

**Output:**



**c) Boolean operators:**

**!** -- This is logical negation. This inverts a true condition to false and vice versa.

**-o** -- This is logical OR. If one of the operands is true, then the condition becomes true.

**-a** -- This is logical AND. If both the operands are true, then the condition becomes true otherwise false.

**Code:**

echo "Enter the value of a"

read a

echo "Enter the value of b"

read b

if [ $a != $b ]

then

echo "a is not equal to b"

else

echo "a is equal to b"

fi

if [ $a != $b -o $a -lt $b ]

then

echo "True"

else

echo "Both are False"

fi

if [ $a != $b -o $a -gt $b ]

then

echo "True"

else

echo "Both are False"

fi

if [ $a != $b -a $a -gt $b ]

then

echo "Both are True"

else

echo "False"

fi

if [ $a != $b -a $a -gt $b ]

then

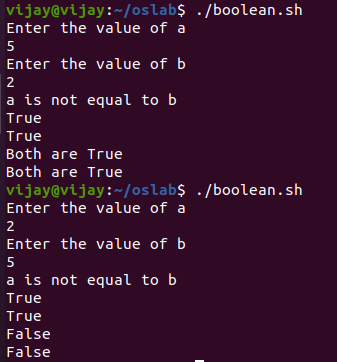
echo "Both are True"

else

echo "False"

fi

**Output:**



**d) String operators:**

= checks if the values of two operands are equal or not; if yes, then the condition becomes true.

**!=** checks if the values of two operands are equal or not; if the values are not equal, then the condition becomes true.

**-z** checks if the given string operand size is zero; if it is of length zero, then it returns true.

**-n** checks if the given string operand size is non-zero; if it is of non-zero length, then it returns true.

**Code:**

echo "Enter the first string"

read a

echo "Enter the second string"

read b

if [ "$a" = "$b" ]

then

echo "Strings are equal"

else

echo "Strings are not equal"

fi

if [ "$a" != "$b" ]

then

echo "Strings are not equal"

else

echo "Strings are equal"

fi

if [ -z "$a" ]

then

echo "String size of a is equal to zero"

else

echo "String size of a is not equal to zero"

fi

if [ -n "$b" ]

then

echo "String size of b is not equal to zero"

else

echo "String size of b is equal to zero"

fi

if [ $a ]

then

echo "a is not empty"

else

echo "a is empty"

fi

if [ $b ]

then

echo "b is not empty"

else

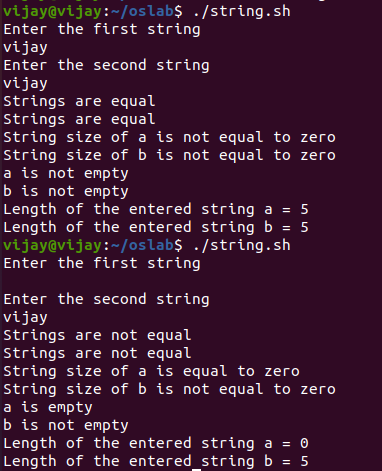
echo "b is empty"

fi

echo "Length of the entered string a = ${#a}"

echo "Length of the entered string b = ${#b}"

**Output:**



**e) File Test operators:**

**-b file** Checks if file is a block special file; if yes, then the condition becomes true.

**-c file** Checks if file is a character special file; if yes, then the condition becomes true.

**-d file** Checks if file is a directory; if yes, then the condition becomes true.

**-f file** Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.

**-g file** Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true.

**-k file** Checks if file has its sticky bit set; if yes, then the condition becomes true.

**-p file** Checks if file is a named pipe; if yes, then the condition becomes true.

**-t file** Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true.

**-u file** Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true.

**-r file** Checks if file is readable; if yes, then the condition becomes true.

**-w file** Checks if file is writable; if yes, then the condition becomes true.

**-x file** Checks if file is executable; if yes, then the condition becomes true.

**-s file** Checks if file has size greater than 0; if yes, then condition becomes true.

**-e file** Checks if file exists; is true even if file is a directory but exists.

**Code:**

file="/home/vijay/oslab/string.sh"

if [ -r $file ]

then

echo "File has read access"

else

echo "File does not have read access"

fi

if [ -w $file ]

then

echo "File has write permission"

else

echo "File does not have write permission"

fi

if [ -x $file ]

then

echo "File has execute permission"

else

echo "File does not have execute permission"

fi

if [ -d $file ]

then

echo "File is a directory"

else

echo "It is not a directory"

fi

if [ -f $file ]

then

echo "File is an ordinary file"

else

echo "It is a special file"

fi

if [ -e $file ]

then

echo "File exists"

else

echo "File does not exist"

fi

if [ -s $file ]

then

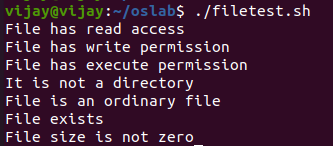
echo "File size is not zero"

else

echo "File size is zero"

fi

**Output:**



**3) Decision making statements with example**

If else statements are useful decision-making statements which can be used to select an option from a given set of options.

Unix shell supports following forms of if-else statement:

* if…fi statement
* if…else…fi statement
* if…elif…else…fi statement

**a) if…fi statement:**

The **if...fi** statement is the fundamental control statement that allows Shell to make decisions and execute statements conditionally.

**Syntax:**

if [ expression ]

then

Statement(s) to be executed if expression is true

fi

**Code**:

echo "Enter the value of a"

read a

echo "Enter the value of b"

read b

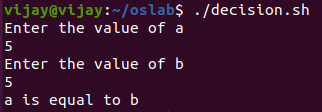
if [ $a = $b ]

then

echo "a is equal to b"

fi

**Output:**

****

**b) if…else…fi statement:**

The **if...else...fi** statement is the next form of control statement that allows Shell to execute statements in a controlled way and make the right choice.

**Syntax:**

if [ expression ]

then

Statement(s) to be executed if expression is true

else

Statement(s) to be executed if expression is not true

fi

**Code:**

echo "Enter the value of a"

read a

echo "Enter the value of b"

read b

if [ $a = $b ]

then

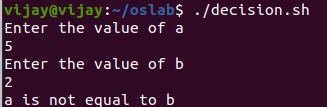
echo "a is equal to b"

else

echo "a is not equal to b"

fi

**Output:**

****

**c) if…elif…else…fi statement:**

The if...elif...else…fi statement is the one level advance form of control statement that allows Shell to make correct decision out of several conditions.

## Syntax:

if [ expression 1 ]

then

Statement(s) to be executed if expression 1 is true

elif [ expression 2 ]

then

Statement(s) to be executed if expression 2 is true

elif [ expression 3 ]

then

Statement(s) to be executed if expression 3 is true

else

Statement(s) to be executed if no expression is true

fi

**Code:**

echo "Enter the value of a"

read a

echo "Enter the value of b"

read b

if [ $a -lt $b ]

then

echo "a is less than b"

elif [ $a -gt $b ]

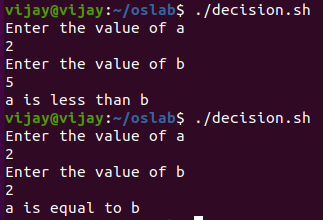
echo "a is greater than b"

else

echo "a is equal to b"

fi

**Output:**

****

**4) Case statements with example**

**case** command in Linux is the best alternative when we had to use multiple if/elif on a single variable. It is used to execute the commands based on the pattern matching.

**Syntax:**

case word in

pattern1)

statement(s) to be executed if pattern1 matches

;;

pattern2)

statement(s) to be executed if pattern2 matches

;;

pattern3)

statement(s) to be executed if pattern3 matches

;;

\*)

Default condition to be executed

;;

esac

**Code:**

echo "Enter your name"

read name

echo "Enter year of study[First/Second/Third/Final]"

read year

case $year in

"First")

echo "$name is a First year student"

;;

"Second")

echo "$name is a Second year student"

;;

"Third")

echo "$name is a Third year student"

;;

"Final")

echo "$name is a Final year student"

;;

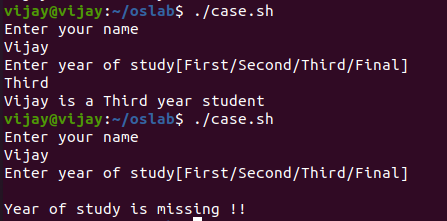
\*)

echo "Year of study is missing !!"

;;

esac

**Output:**



**5) Looping statements with example**

A loop is a powerful programming tool that enables you to execute a set of commands repeatedly. Types of loops available to shell programmers −

* [The while loop](https://www.tutorialspoint.com/unix/while-loop.htm)
* [The for loop](https://www.tutorialspoint.com/unix/for-loop.htm)
* [The until loop](https://www.tutorialspoint.com/unix/until-loop.htm)
* [The select loop](https://www.tutorialspoint.com/unix/select-loop.htm)

**a) While loop:**

The **while** loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

## Syntax:

while command

do

Statement(s) to be executed if command is true

done

**Code:**

a=0

while [ $a -lt 5 ]

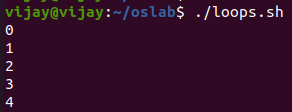
do

echo "$a"

a= `expr $a + 1`

done

**Output:**



**b) For loop:**

The **for** loop operates on lists of items. It repeats a set of commands for every item in a list.

**Syntax:**

for var in word1 word2 ... wordN

do

Statement(s) to be executed for every word.

done

**Code:**

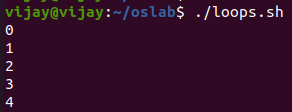
for a in 0 1 2 3 4

do

echo "$a"

done

**Output:**



**Code:**

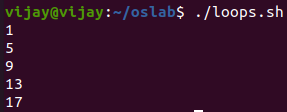
for a in $(seq 1 4 20)

do

echo "$a"

done

**Output:**



**c)** **Until loop:**

The while loop is perfect for a situation where you need to execute a set of commands while some condition is true. Sometimes you need to execute a set of commands until a condition is true.

**Syntax:**

until command

do

Statement(s) to be executed until command is true

done

**Code:**

a=0

until [ ! $a -lt 5 ]

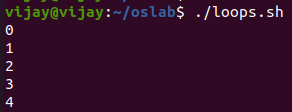
do

echo "$a"

a=`expr $a + 1`

done

**Output:**



**d) Select loop:**

The **select** loop provides an easy way to create a numbered menu from which users can select options. It is useful when you need to ask the user to choose one or more items from a list of choices.

**Syntax:**

select var in word1 word2 ... wordN

do

Statement(s) to be executed for every word.

done

**Code:**

PS3="Enter your choice ==>"

select PLAY in cricket football basketball chess carrom badminton all none

do

case $PLAY in

cricket|football|basketball|all)

echo "Go to ground"

;;

chess|carrom|badminton)

echo "Play inside"

;;

none)

break

;;

\*)

echo "Invalid selection"

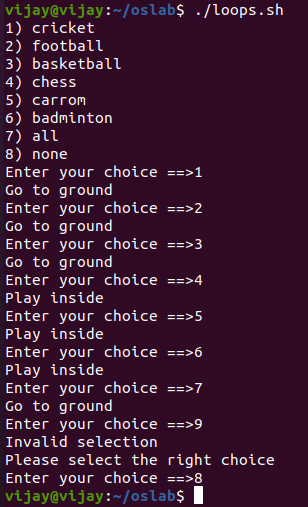
echo "Please select the right choice"

;;

esac

done

**Output:**



**RESULT:**

Hence a case study on shell scripting is done and the output has been verified.