BASH SCRIPTING

Operators

CONTENTS

How to input values?

```
$ echo -n "Enter Fullname: "
$ read fullname
$ echo -n "Enter a number "
$ read number
```

- In case of Bash, you don't have to define the data type of any variable at the time of variable declaration. Bash variables are **untyped**, which means just type the variable name by assigning its value, and it will automatically consider that data type.
- Each variable can be referenced for its value by using $ symbol before it.

Arithmetic Operators in Shell

```
#!/bin/sh

a=10

b=20

val=`expr $a + $b`

echo "a + b : $val"
```

Arithmetic Operators in Shell-Expected Output

+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

If-then Operator in Shell

```
if [ $a == $b ]
then
  echo "a is equal to b"
fi
if [ $a != $b ]
then
  echo "a is not equal to b"
 fi
```

Points to Remember

The following points need to be considered when using the Arithmetic Operators –

- There must be spaces between the operators and the expressions. For example, 2+2 is not correct; it should be written as 2 + 2.
- Complete expression should be enclosed between ‘ ‘, called the inverted commas.
- You should use \ on the * symbol for multiplication.

Command Chaining

1. && Operator (AND Operator)

The syntax to use AND Operator is :

```
# command 1 && command 2
```

```
$ ping -c 5 localhost && df -h
```

2. OR Operator (||)

On our list the Second Bash shell command line Chaining Operators (Linux operator) is OR Operator. The syntax to use OR Operator is :

```
# command 1 || command 2
```

Command Chaining

3. AND & OR Operator (&& and ||)

Combination of && Operator & OR Operator (||) is quite interesting and will give you a nice output if you use it properly. The combination of these 2 operators like if…else statement in programming. Let's take an example so that you will get more Idea on this :

Example : 1

Here I tried to success the First command and after a successful execution of the first command, we will get a message Directory created successfully.

```
$ mkdir data && echo "Directory created successfully" || echo "Directory creation failed"
```

Command Chaining

Example : 2

Here I tried to Fail the First command and after Failure of the first command we will get a message User creation failed.

```
$ useradd helpdesk  && echo "User created successfully" || echo "User creation failed"
```

PIPE

4. PIPE Operator (|)

The Fourth Bash shell command line Chaining Operators (Linux operator) is PIPE Operator (|). PIPE is a kind of operator that can be used to display output of the first command by taking input to the second command. For example, you want to check all currently running system process using command ps -ef. but as the processes list is so long that it cannot be covered in one screen. In that case, you can use the Filter Operator with command more. Refer the command below :

```
$ ps -ef | more
$ who | cut -c 1-10
```

Semicolon Operator

Fifth Bash shell command line Chaining Operators (Linux operator) is Semicolon Operator (;). Semicolon Operator executes multiple commands at once sequentially as it is mentioned and provides output without depending on the success & failure of other commands like && Operator and OR Operator (||).

```
$ ping -c 5 localhost ; ifconfig ens33 ; df -h
```

Test Command

A test command is a command that is used to test the validity of a command. It checks whether the command/expression is true or false. It is used to check the type of file and the permissions related to a file. Test command returns 0 as a successful exit status if the command/expression is true, and returns 1 if the command/expression is false.

Test Command

```
#!/bin/bash

a=20

b=20

# using test command to
check if numbers

# are equal

if test "$a" -eq "$b"

then

  echo "a is equal to b"

else

  echo "a is not equal to b"

fi
```

Test string expressions

```
#!/bin/bash
# Example to check if two strings are equal or not
# first string
a="Hello"
b="Hello"
if test "$a" = "$b"
then
  echo "a is equal to b"
else
  echo "a is not equal to b"
    fi
```

Test files

```
if test -s $filename
then
  echo "File is not empty"
else
  echo "File is empty"
fi
```

Test user is root or not

Script to check if the script is running as root or not, using the test command.

```
# check if the script is run as root or not
if test "$(id -u)" == "0"
 then
    echo "This script is running as root"
 else
    echo "This script is not running as root"
    exit 1
fi
```