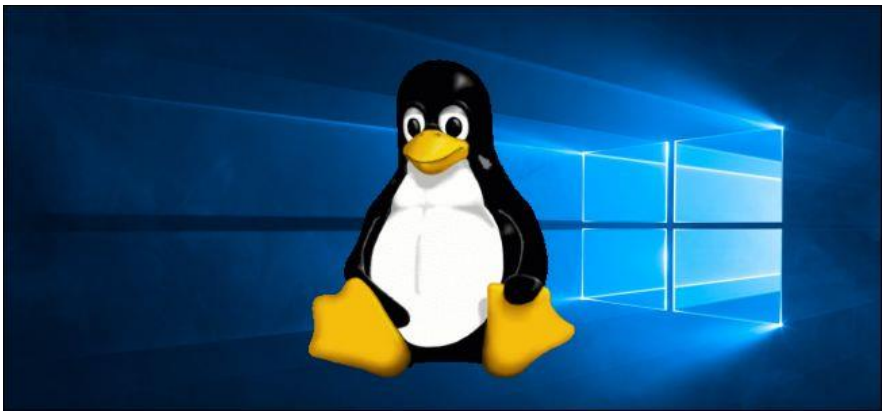
UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

SOFTWARE ENGINEERING DEPARTMENT



Operating Systems

Experiment 5

Shell Programming –II (If-Else Statement)
Command Line Arguments Passing.
Using Linux Calculator in Shell Programming

CLO 2. ***Use modern tools and languages.***
CLO 3. ***Demonstrate an original solution of problem under discussion.***
CLO 4. ***Work individually as well as in teams***

UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

SOFTWARE ENGINEERING DEPARTMENT

Shell Programming and if-else Statement

Shells (bash) Structured Language Constructs

bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations. Arithmetic operations are the most basic in any kind of programming language. Linux or Unix operating system provides the **bc command** and **expr command** for doing arithmetic calculations. You can use these commands in bash{Terminal} or shell script also for evaluating arithmetic expressions.

The bc command supports the following features:

- Arithmetic operators
- Increment or Decrement operators
- Assignment operators
- Comparison or Relational operators
- Logical or Boolean operators
- Math functions
- Conditional statements
- Iterative statements

$ bc
After this command bc is started and waiting for your commands, i.e. give it some calculation as

follows type 5 + 2 as:
5 + 2
7
7 is response of bc i.e. addition of 5 + 2.
You can even try:
5 - 2
5 / 2
See what happened if you type 5 > 2 as follows:
5 > 2
1
1 (One?) is response of bc, How? bc compare 5 with 2 as, Is 5 is greater than 2, and print the answer by showing 1 value. Now try
5 < 2
0
0 (Zero) indicates the false i.e. Is 5 is less than 2?

Try following in bc to clear your idea and see bc's response

5 > 12
5 == 10
5 != 2
5 == 5
12 < 2

Expression	Meaning to us	Your Answer	BC's Response
5 > 12	Is 5 greater than 12	NO	0
5 == 10	Is 5 is equal to 10	NO	0
5 != 2	Is 5 is NOT equal to 2	YES	1
5 == 5	Is 5 is equal to 5	YES	1
1 < 2	Is 1 is less than 2	Yes	1

It means whenever there is any type of comparison in Linux Shell, it gives only two answer one is YES and other is NO.

In Linux Shell Value	Meaning	Example
Zero Value (0)	Yes/True	0
NON-ZERO Value	No/False	-1, 32, 55 anything but not zero

Remember both bc and Linux Shell uses different ways to show True/False values.

Value	Shown in bc as	Shown in Linux Shell as
True/Yes	1	0
False/No	0	Non - zero value

For Mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements	But in Shell	
			For test statement with if command	**For [expr] statement with if command**
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]

-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

NOTE: == is equal, != is not equal.

Using 'bc' in a Script

```
variable=`echo "options; expression" | bc`
```

Examples:

```
$ x=20
$ y=10

$ echo $x+$y | bc
$ echo $x\*$y | bc
```

Note: The result does not contain the fractional part. bc does not give the fraction of the division by default. This can be obtained using one of the special variables of the bc command, **scale.** scale variable lets us to define the precision we would like to have it.

```
$ echo "scale=2;$x/$y" | bc
```

How to store the result of complete operation in variable?
Example:

```
Input:
$ x=`echo "12+5" | bc`
$ echo $x
Output:17
```

Explanation: Stores the result of first line of input in variable x and then display variable x as **$x**.

Examples:

```
Input: $ echo "var=10;var" | bc
Output: 10
```

Explanation: Assign 10 to the variable and print the value on the terminal.

```
Input: $ echo "var=10;var^=2;var" | bc
Output: 100
```

Explanation: Squares the value of the variable and print the value on the terminal.

How to store the result of complete operation in variable?
Example:

```
Input:
$ x=`echo "var=500;var%=7;var" | bc`
$ echo $x
Output:3
```

Explanation: Stores the result of 500 modulo 7 i.e. remainder of 500/7 in variable x and then display variable x as **$x**.

```
Input: $ echo "var=10;++var" | bc
Output: 11
```

Explanation: Variable is increased first and then result of variable is stored.

Relational operators
are used to compare 2 numbers. If the comparison is true, then result is **1**. Otherwise(false), returns **0**. These operators are generally used in conditional statements like **if**.

Examples:

```
Input: $ echo "10>5" | bc
Output: 1

Input: $ echo "1==2" | bc
Output: 0
```

Logical operators

are mostly used in conditional statements. The result of the logical operators is either **1**(TRUE) or **0**(FALSE).

```
Input: $ echo "10 && 5" | bc
Output: 1

Input: $ echo "0 || 0" | bc
Output: 0
```

Example:

```
num1=75.5
num2=20
add=(echo "scale=4; $num1+$num2" | bc)
sub=(echo "scale=4; $num1-$num2" | bc)
echo "Addition of $num1 and $num2 is $add"
echo "Subtraction of $num1 and $num2 is $sub"
```

Output of the above program

```
Addition of 75.5 and 20 is 95.5
Subtraction of 75.5 and 20 is 55.5
```

Conditional Statements

Conditional Statements are used to take decisions and execute statements based on these decisions. bc command supports the if condition.

Syntax:

```
if(condition) {statements} else {statemnts}
```

Example:

```
Input: $ echo 'n=8; m=10; if(n>m) print "n is greater" else
print "m is greater" ' | bc -l
Output: m is greater
```

Iterative statements

bc command supports the for loop and while loop for doing iterations.

Syntax:

```
for(assignment; condition; updation)
{
      statements.....
      .......
      ........
}

OR
```

```
while(condition)
{
      statements.....
      .......
      ........
}
```

Examples:

```
Input: $ echo "for(i=1; i<=10; i++) {i;}" | bc
Output:
1
2
3
4
5
6
7
8
9
10
```

```
Input: $ echo "i=1;while(i<=10) {i; i+=1}" | bc
Output:
1
2
```

```
3
4
5
6
7
8
9
10
```

Explanation: Both examples prints numbers from 1 to 10 using the respective looping syntax

The read Statement

Use to get input from keyboard and store them to variable.

Syntax: read varible1, varible2,...varibleN

in addition to the math functions, the following functions are also supported:

- **length(x)** : returns the number of digits in x.
- **read()** : Reads the number from the standard input.
- **scale(expression)** : The value of the scale function is the number of digits after the decimal point in the expression.
- **ibase** and **obase** define the conversion base for input and output numbers. The default for both input and output is base 10.
- **last** (an extension) is a variable that has the value of the last printed number.

Example:
Create script as

```
$ cat > sayH
#Script to read your name from key-board
echo "Your first name please:"
read fname
echo "Hello $fname, Lets be friend!"
```

Run it as follows

```
$ chmod +x sayH
$ ./sayH
```

Note:

relative pathname– that uses either the current or parent directory as reference and specifies the path relative to it. A relative path-name uses one of these cryptic symbols:

```
.(a single dot) - this represents the current directory.
..(two dots) - this represents the parent directory.
```

This script first asks you your name and then waits to enter name from the user, then user enters name from keyboard (After giving name you have to press ENTER key) and this entered name through keyboard is stored (assigned) to variable fname.

1. Bash If..then..fi statement

```
if [ conditional expression ]
then
       statement1
       statement2
       .
fi
```

This if statement is also called as simple if statement. If the given conditional expression is true, it enters and executes the statements enclosed between the keywords “then” and “fi”. If the given expression returns zero, then consequent statement list is executed.

Example:

```
count=100
if [ $count -eq 100 ]
then
  echo "Count is 100"
fi
```

Example:

Type following commands (assumes you have file called **hello**)
$ cat hello

A file hello is shown
$ echo $?
The cat command return zero (0) i.e. exit status, on successful, this can be used, in if condition as follows,
Write shell script as:

```
$ cat > showfile
#Script to print file
if cat $1
then
echo  "File $1, found and successfully echoed"
fi
```

Run above script as:

```
$ chmod 755 showfile
$./showfile hello
```

Shell script name is showfile ($0) and hello is argument (which is $1). Then shell compare it as follows:
if cat $1 which is expanded to if cat hello.

Detailed Explanation:
if cat command finds hello file and if its successfully shown on screen, it means our cat command is successful and its exist status is 0 (indicates success), So our if condition is also true and hence statement echo " File $1, found and successfully echoed" is proceed by shell. Now if cat command is not successful then it returns non-zero value (indicates some sort of failure) and this statement echo “File $1, found and successfully echoed" is skipped by our shell.

Why Command Line arguments required:

Let's take rm command, which is used to remove file, But which file you want to remove and how you will tell this to rm command (Even rm command does not ask you name of file that would like to remove). So what we do is we write as command as follows:

```
$ rm {file-name}
```

Here rm is command and file-name is file which you would like to remove. This way you tell this to rm command which file you would like to remove. So we are doing one way communication with our command by specifying file-name. Also you can pass command line

arguments to your script to make it more user friendly. But how we address or access command line argument in our script.

Lets take ls command

```
$ ls -a /*
```

This command has 2 command line argument -a and /* is another. For shell script,

```
$ myshell a b
```

- Shell Script name i.e. myshell
- First command line argument passed to myshell i.e. a
- Second command line argument passed to myshell i.e. b

In shell if we wish to refer this command line argument we refer above as follows

```
myshell it is $0
```

a it is $1
b it is $2

Here $# will be 2 (Since a and b are only two Arguments). At a time such 9 arguments can be used from $0..$9, You can also refer all of them by using $* (which expand to $0,$1,$2...$9`) .

For e.g. now will write script to print command ling argument and we will see how to access them

```
$ cat > demo
#!/bin/sh
#
# Script that demos, command line args
#
echo "Total number of command line argument are $#"
echo "$0 is script name"
echo "$1 is first argument"
echo $2 is second argument"
echo "All of them are :- $*"
```

Save the above script by pressing ctrl+d, now make it executable

$ chmod +x demo
$ ./demo Hello World

if...else...fi

If given condition is true then command1 is executed otherwise command2 is executed.

Syntax:

```
if condition
then
        condition is zero (true - 0)
        execute all commands up to else statement
else
        if condition is not true then
        execute all commands up to fi
fi
```

Example:

Write Script as follows:

```
$ vi isnump_n
#
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
echo "$0 : You must give one integer"
else
if test $1 -gt 0
then
echo "$1 number is positive"
else
echo "$1 number is negative"
fi

Fi
```

Try it as follows:

$ chmod 755 isnump_n

$ ./isnump_n 5

5 number is positive

$ ./isnump_n -45

-45 number is negative

$ ./isnump_n

./isnump_n : You must give one integer

$ isnump_n 0
0 number is negative

Detailed Explanation

Here o/p: 0 number is negative

- Here first we see if no command line argument is given then it print error message as "*./isnump_n*: **You must give/supply one integers".**

- if statement checks whether number of argument **($#)** passed to script is not equal (-eq) to 0, if we passed any argument to script then this if statement is false and if no command line argument is given then this if statement is true.

- echo command i.e.

echo "**$0** : You must give one integer"
| |
1 ***2***

- 1 will print Name of script
- 2 will print this error message

- The last sample run **$ isnump_n 0,** gives output as "0 number is negative", because given argument is not > 0, hence condition is false and it's taken as negative number. To avoid this, replace second if statement with if test $1 -**ge** 0.

(Simply if you replace test $1 -**gt** 0 with test $1 -**ge** 0 then it will print 0 is positive)

Nested … if

You can use the nested ifs as follows also:

Syntax:

```
if condition
then
        if condition
        then
                .....
                ..
                do this
        else
                ....
                ..
                do this
        fi
else
```

```
            ...
            .....
            do this
      fi
```

Example:

You can write the entire if-else construct within either the body of the if statement or in the body of an else statement. This is called the nesting of ifs.

```
$ vi nestedif.sh
a=0

echo "1. Unix (Sun Os)"
echo "2. Linux (Red Hat)"
echo "Select your os choice [1 or 2]?"
read a

if [ $a -eq 1 ]
then
     echo "You Pick up Unix (Sun Os)"

else #### nested if i.e. if within if ######

       if [ $a -eq 2 ]
       then
             echo "You Pick up Linux (Red Hat)"
       else
             echo "What you don't like Unix/Linux OS."
       fi
fi
```

Run the above shell script as follows:

$ chmod 755 nestedif.sh

$ ./nestedif.sh

1. Unix (Sun Os)
2. Linux (Red Hat)

Select you os choice [1 or 2]? **1**

You Pick up Unix (Sun Os)

$ ./nestedif.sh

1. Unix (Sun Os)
2. Linux (Red Hat)

Select you os choice [1 or 2]? **2**

You Pick up Linux (Red Hat)

$ ./nestedif.sh

1. Unix (Sun Os)

2. Linux (Red Hat)
Select you os choice [1 or 2]? **3**
What you don't like Unix/Linux OS.

test command or [expr]

test command or [expr] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.

Syntax:

test expression OR [expression]

Example:

Following script determine whether given argument number is positive.

```
$ cat > ispostive
# Script to see whether argument is positive
if test $1 -gt 0
then
echo "$1 number is positive"
fi
```

Run it as follows:

$ chmod 755 ispostive
$./ ispostive 5
5 number is positive
$./ispostive -45
Nothing is printed
$./ispostive
./ispostive: test: -gt: unary operator expected

Multilevel if-then-else

Syntax:

```
if condition
then
        condition is zero (true - 0)
        execute all commands up to elif statement
elif condition1
then
        condition1 is zero (true - 0)
        execute all commands up to elif statement
elif condition2
then
```

```
        condition2 is zero (true - 0)
        execute all commands up to elif statement
    else
        None of the above condtion,condtion1,condtion2 are true (i.e.
        all of the above nonzero or false)
        execute all commands up to fi
    fi
```

For multilevel if-then-else statement try the following script:
Example:

```
$ cat > elf
# Script to test if..elif...else
if [ $1 -gt 0 ]  then
  echo "$1 is positive"
elif [ $1 -lt 0 ]
then
  echo "$1 is negative"
elif [ $1 -eq 0 ]
then
  echo "$1 is zero"
else
  echo "Opps! $1 is not number, give number"
fi
```

Try above script as follows:
$ chmod 755 elf
$ ./elf 1
$ ./elf -2
$ ./elf 0
$ ./elf a
Here o/p for last sample run:
./elf: [: -gt: unary operator expected
./elf: [: -lt: unary operator expected
./elf: [: -eq: unary operator expected
Opps! a is not number, give number

Above program gives error for last run, here integer comparison is expected therefore error like "*./elf: [: -gt: unary operator expected*" occurs, but still our program notify this error to user by providing message "*Opps! a is not number, give number*".

UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

SOFTWARE ENGINEERING DEPARTMENT

LAB TASKS

TASK 1:

Write shell script as follows:

```
cat > trmif
# Script to test rm command and exist status
if rm $1
then
echo "$1 file deleted"
fi
```

Press Ctrl + d to save
$ chmod 755 trmif

Answer the following question in reference to above script:
1. hello file exists on your disk and you give command, **$ ./trmif hello** what will be output?
2. If bar file not present on your disk and you give command, **$ ./trmif bar** what will be output?
3. And if you type **$ ./trmif** What will be output?

TASK 2:

Write a shell script that computes the House Rent Allowance of an employee according to the following:
1) if basic salary is <1500 then HRA=10% of the basic salary.
2) if basic salary is >1500 then HRA=20% of the basic salary.

TASK 3:

Write a shell script to ADD two numbers taken from argument?
$ ./ADD 5 6
Output : Sum of 5 and 6 = 5+6 = 11

Note: show error if no argument or more than 2 arguments are passed