APPLIED OPERATING SYSTEM LABORATORY

FEU ALABANG FEU DILIMAN FEU TECH

Technology Driven by Innovation

MODULE 8

LINUX SHELL SCRIPTING

FEU ALABANG FEU DILIMAN FEU TECH

Technology Driven by Innovation

OBJECTIVES

Upon completion of this module, the student will be able to:

- Create and execute shell scripts using LINUX input/output commands, arithmetic operations and condition statements in solving problems

TOPIC OUTLINE

- **Introduction to Shell Script**
    - What is a Linux Shell?
    - What is a Shell Script?
    - Why to write a Shell Script?
    - How to write a Shell Script?
    - How to execute a Shell Script?
- **Actual Shell Scripting**
    - Variables in Shell
    - Arithmetic Operators
    - Input / Output Statements
    - Conditional Statements

INTRO TO SHELL SCRIPT

What is a LINUX SHELL?

- Shell is a user program or its an environment provided for user interaction.

- Shell is a command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

- Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

INTRO TO SHELL SCRIPT

There are several shells are available for Linux systems like –

Shell Name	Developed by	Where	Remark
BASH (Bourne-Again SHell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	
TCSH	See the man page. Type $ man tcsh		TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

INTRO TO SHELL SCRIPT

What is a SHELL SCRIPT?

- Shell Script is a file that contains ASCII text
- It is writing a series of commands for the shell to execute.
- Shell script is similar to batch file in MS-DOS.

INTRO TO SHELL SCRIPT

Why to write a SHELL SCRIPT?

- Shell script can take input from user, file and output them on screen.
- Useful to create our own commands.
- Save lots of time.
- To automate some task of day today life.
- Customizing administrative tasks.
- Creating simple applications.

INTRO TO SHELL SCRIPT

Practical examples where shell scripting actively used

- Monitoring your Linux system.
- Data backup and creating snapshots.
- Creating email based alert system.
- Find out what processes are eating up your system resources.
- Find out all logged in users and what they are doing.
- Find out all failed login attempts, if login attempts are continue repeatedly from same network IP automatically block all those IPs accessing your network/service via firewall.

INTRO TO SHELL SCRIPT

How to write a SHELL SCRIPT?

Following steps are required to write shell script:

- Use any editor like vi or gedit to write shell script.
- After writing shell script set execute permission for your script as follows:

Syntax: **chmod** permission your-script-name

Examples:

```
$ chmod +x your-script-name
$ chmod 755 your-script-name
```

Note: This will set read write execute(7) permission for owner, for group and other permission is read and execute only(5).

INTRO TO SHELL SCRIPT

How to execute a SHELL SCRIPT?

Syntax:

```
bash your-script-name
sh your-script-name
./your-script-name
```

Examples:

```
bash bar
sh bar
./bar
```

Note: In the last syntax ./ means current directory, But only . (dot) means execute given command file in current shell without starting the new copy of shell.

SHELL SCRIPTING

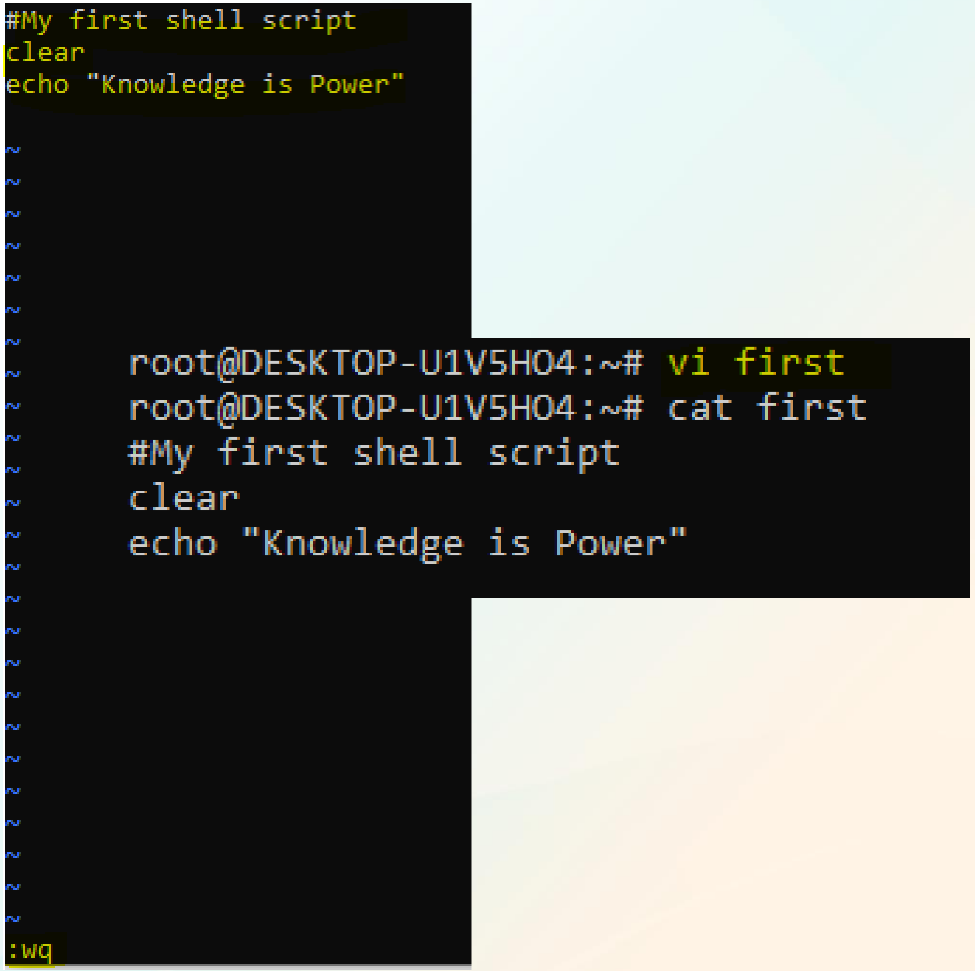
Sample Coding of SHELL SCRIPT

- To print **"Knowledge is Power“** on screen.

Syntax: $ vi first

Type:
```
# My first shell script
clear
echo "Knowledge is Power“
```

To Save, press **Esc** *key* : wq



```
#My first shell script
clear
echo "Knowledge is Power"
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

```
root@DESKTOP-U1V5HO4:~# vi first
root@DESKTOP-U1V5HO4:~# cat first
#My first shell script
clear
echo "Knowledge is Power"
```

SHELL SCRIPTING

Sample Coding of SHELL SCRIPT

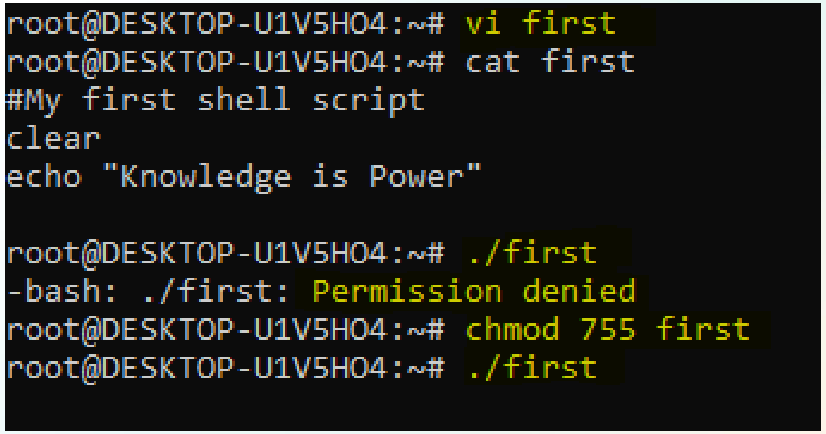
- After saving the script, you can run the script as follows:

  Syntax: **./first**

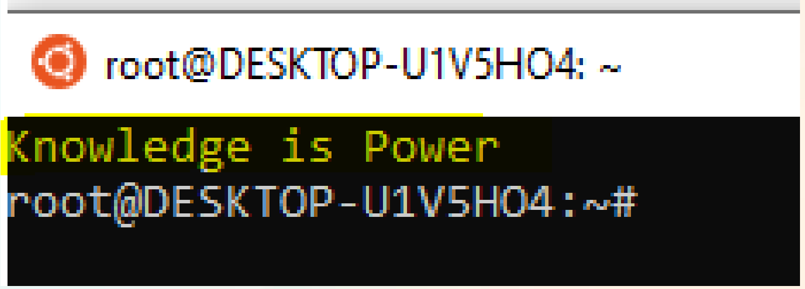
- This will not run script since we have not set execute permission for our script ***first***; to do this type command

  Syntax: **chmod 755 first**
  ./first

- First screen will be clear, then **Knowledge is Power** is printed on screen.



```
root@DESKTOP-U1V5HO4:~# vi first
root@DESKTOP-U1V5HO4:~# cat first
#My first shell script
clear
echo "Knowledge is Power"

root@DESKTOP-U1V5HO4:~# ./first
-bash: ./first: Permission denied
root@DESKTOP-U1V5HO4:~# chmod 755 first
root@DESKTOP-U1V5HO4:~# ./first
```



root@DESKTOP-U1V5HO4: ~

```
Knowledge is Power
root@DESKTOP-U1V5HO4:~#
```

SHELL SCRIPTING

Script Command(s)	Meaning
$ vi first	Start vi editor
# My first shell script	# followed by any text is considered as comment. Comment gives more information about script, logical explanation about shell script. *Syntax:* # comment-text
clear	clear the screen
echo "Knowledge is Power"	To print message or value of variables on screen, we use echo command, general form of echo command is as follows *syntax:* echo "Message"

SHELL SCRIPTING

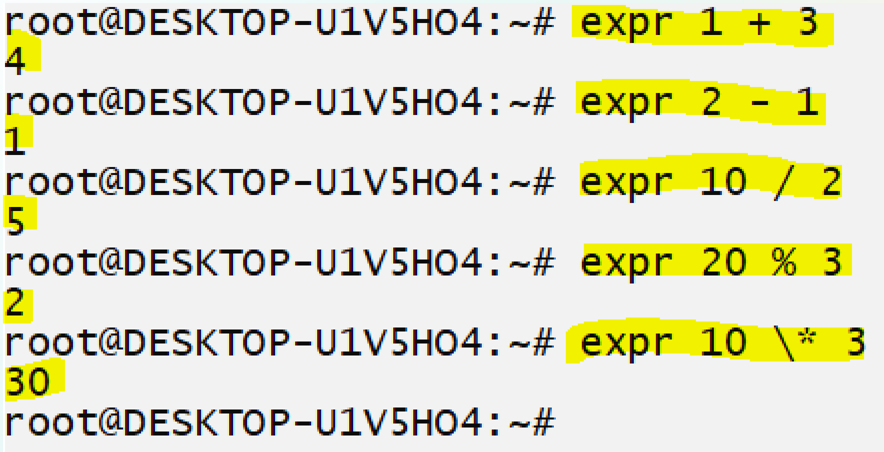
SHELL Arithmetic Operations

- Use to perform arithmetic operations.

Syntax: **expr** op1 math-operator op2

Examples:

```
expr 1 + 3
expr 2 - 1
expr 10 / 2
expr 20 % 3
expr 10 \* 3
echo `expr 6 + 3`
```

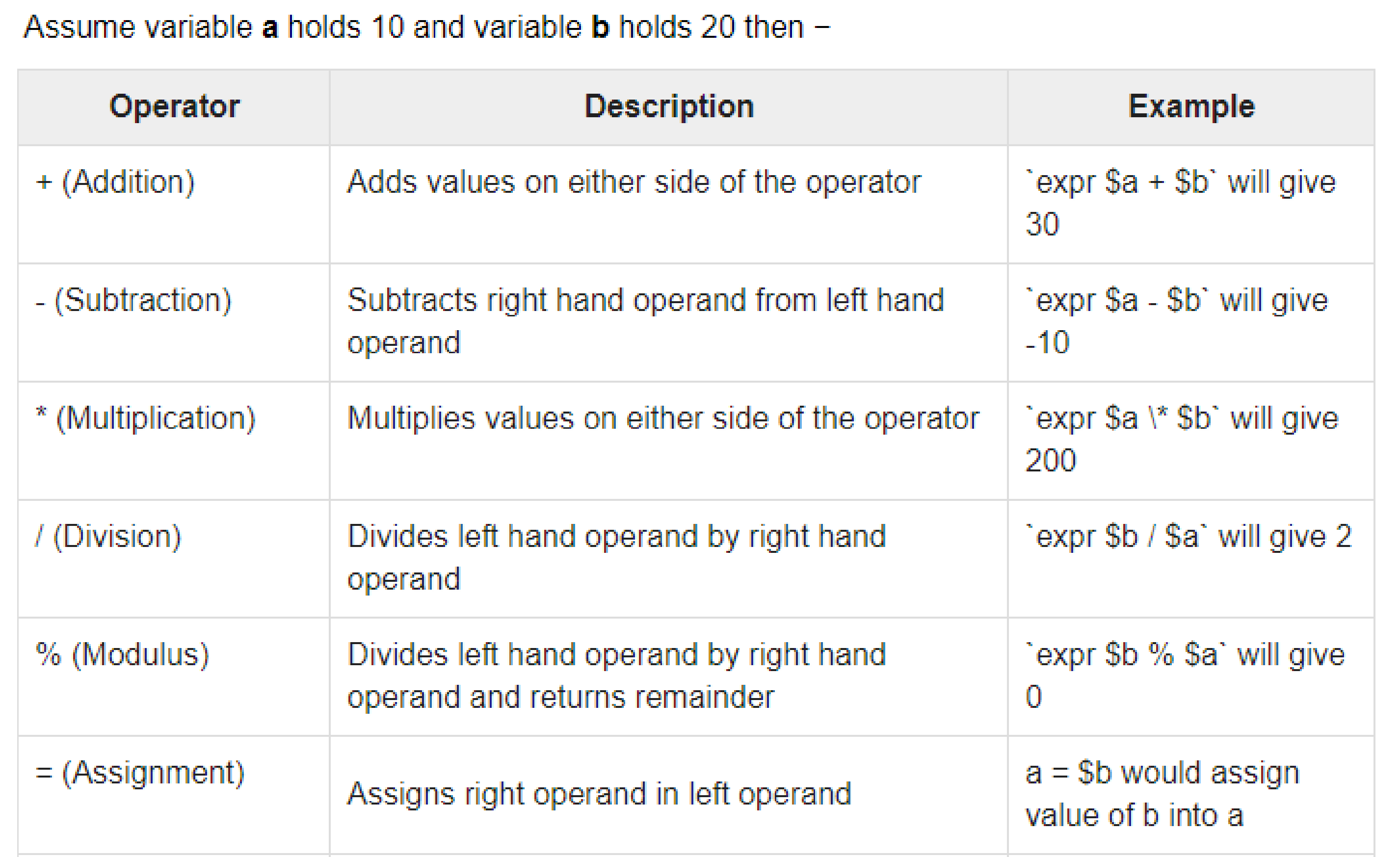


```
root@DESKTOP-U1V5HO4:~# expr 1 + 3
4
root@DESKTOP-U1V5HO4:~# expr 2 - 1
1
root@DESKTOP-U1V5HO4:~# expr 10 / 2
5
root@DESKTOP-U1V5HO4:~# expr 20 % 3
2
root@DESKTOP-U1V5HO4:~# expr 10 \* 3
30
root@DESKTOP-U1V5HO4:~#
```

Note: expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2.
expr 10 * 3 - Multiplication use * and not * since its wild card.

SHELL SCRIPTING

Arithmetic Operations



Assume variable **a** holds 10 and variable **b** holds 20 then −

Operator	Description	Example
+ (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

SHELL SCRIPTING

Variables in SHELL

- In Linux (Shell), there are **two types of variable**:
    - **System Variables** - Created and maintained by Linux itself. This type of variable defined in **CAPITAL LETTERS**.
    - **User Defined Variables** (UDV) - Created and maintained by user. This type of variable defined in **lower case letters**.

SHELL SCRIPTING

How to create a Shell Script using User Defined Variables?

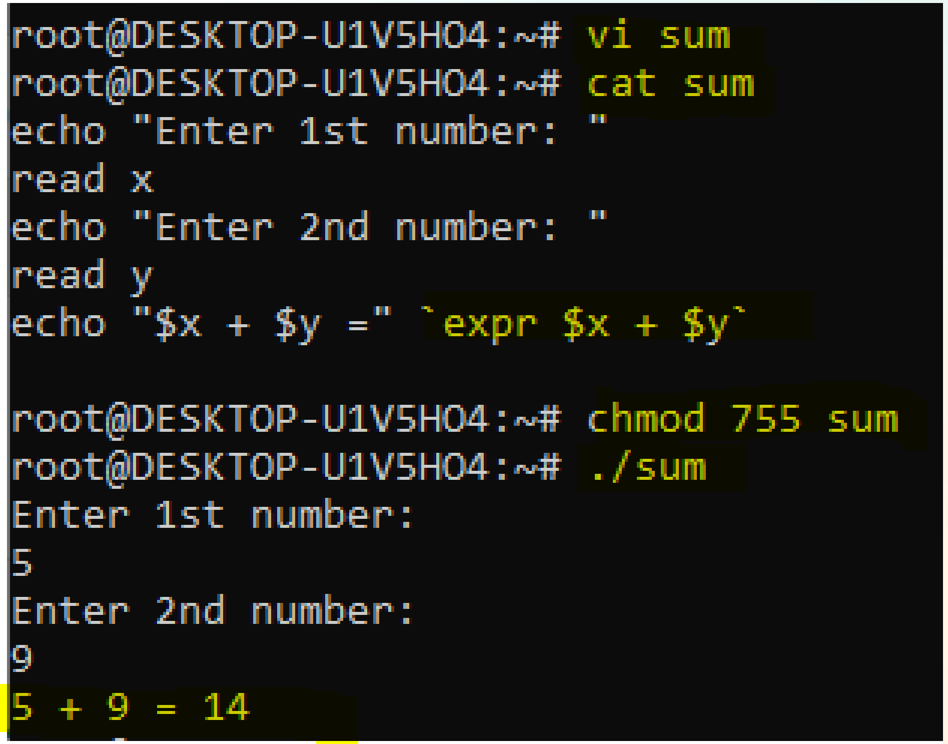
Syntax: vi sum

Type:

```
echo “Enter a number: “
read x
echo “Enter another number: “
read y
echo “$x + $y =” `expr $x + $y`
```

To Save & Exit: ***Esc** key*: wq

To Execute: **./sum**

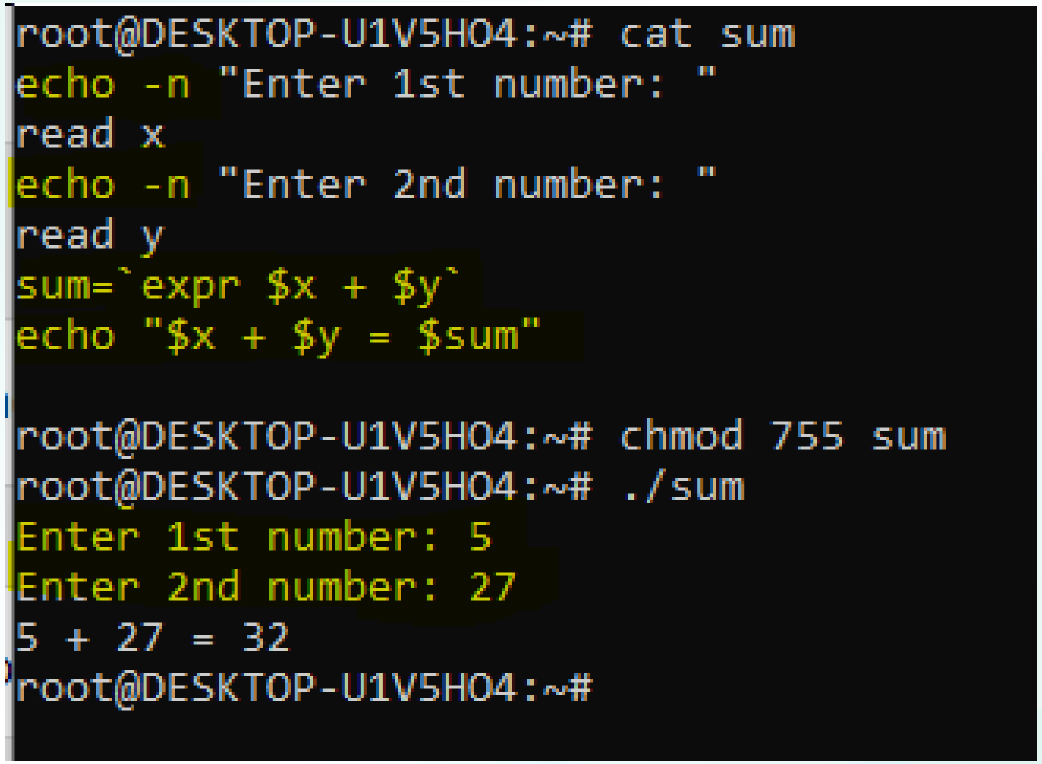


```
root@DESKTOP-U1V5HO4:~# vi sum
root@DESKTOP-U1V5HO4:~# cat sum
echo "Enter 1st number: "
read x
echo "Enter 2nd number: "
read y
echo "$x + $y =" `expr $x + $y`

root@DESKTOP-U1V5HO4:~# chmod 755 sum
root@DESKTOP-U1V5HO4:~# ./sum
Enter 1st number:
5
Enter 2nd number:
9
5 + 9 = 14
```

SHELL SCRIPTING

How to assign expression in a User Defined Variable?

Syntax: **sum=`expr  $x  +  $y`**



```
root@DESKTOP-U1V5HO4:~# cat sum
echo -n "Enter 1st number: "
read x
echo -n "Enter 2nd number: "
read y
sum=`expr $x + $y`
echo "$x + $y = $sum"

root@DESKTOP-U1V5HO4:~# chmod 755 sum
root@DESKTOP-U1V5HO4:~# ./sum
Enter 1st number: 5
Enter 2nd number: 27
5 + 27 = 32
root@DESKTOP-U1V5HO4:~#
```

SHELL SCRIPTING

Condition Statements

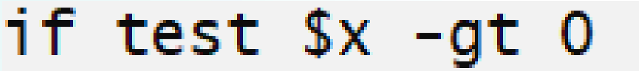
- if condition
- if-else statement
- if..elif..else..fi statement (Else If ladder)
- if..then..else..if..then..fi..fi..(Nested if)
- switch statement

SHELL SCRIPTING

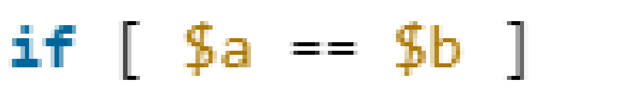
Condition Statements

- **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 **]**



```
if test $x -gt 0
```



```
if [ $a == $b ]
```

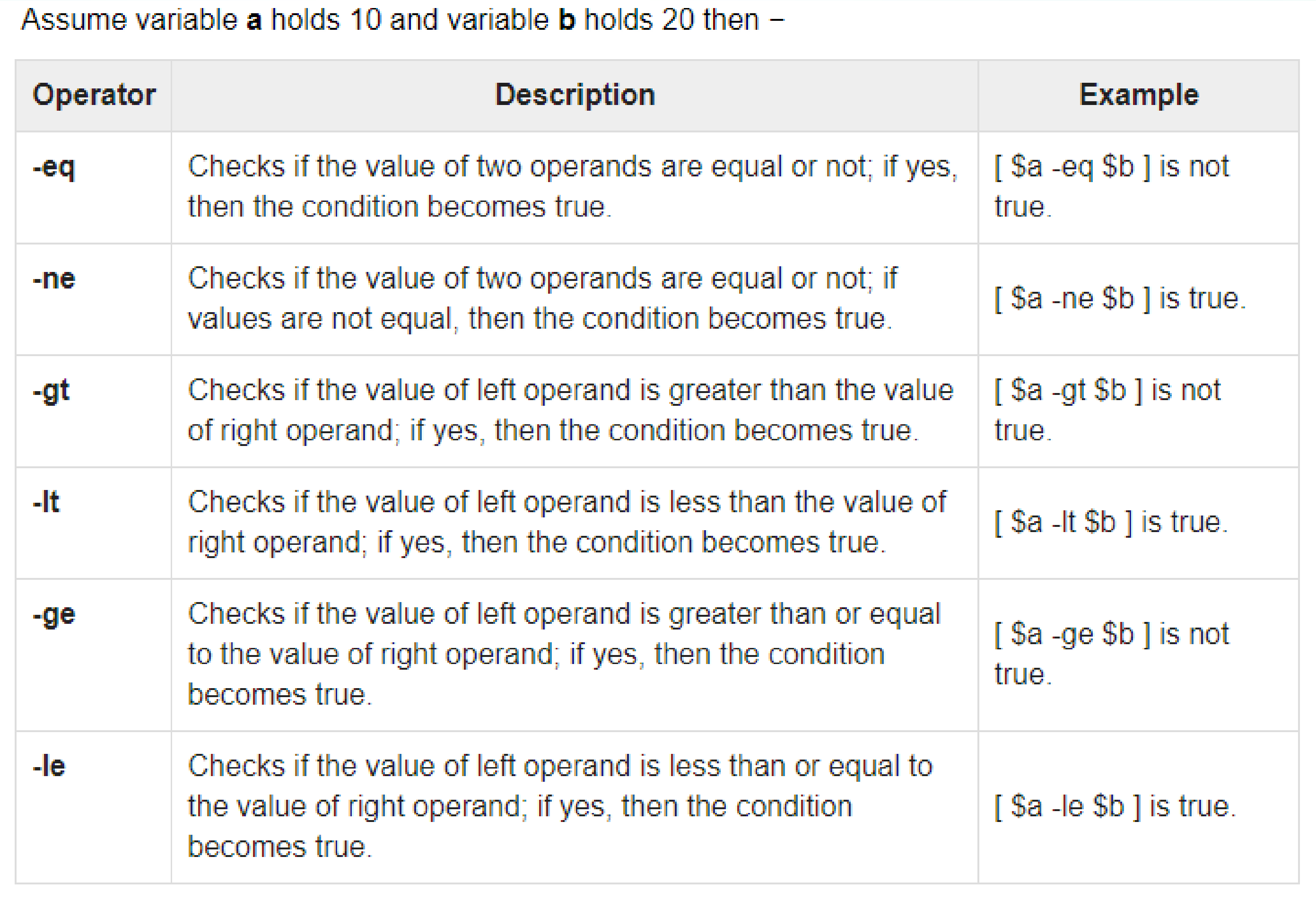
SHELL SCRIPTING

Condition Statements

- Relational Operators

Note: It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them.

For example**,** [$a -gt $b] is **correct** whereas, [$a -gt $b] is incorrect.



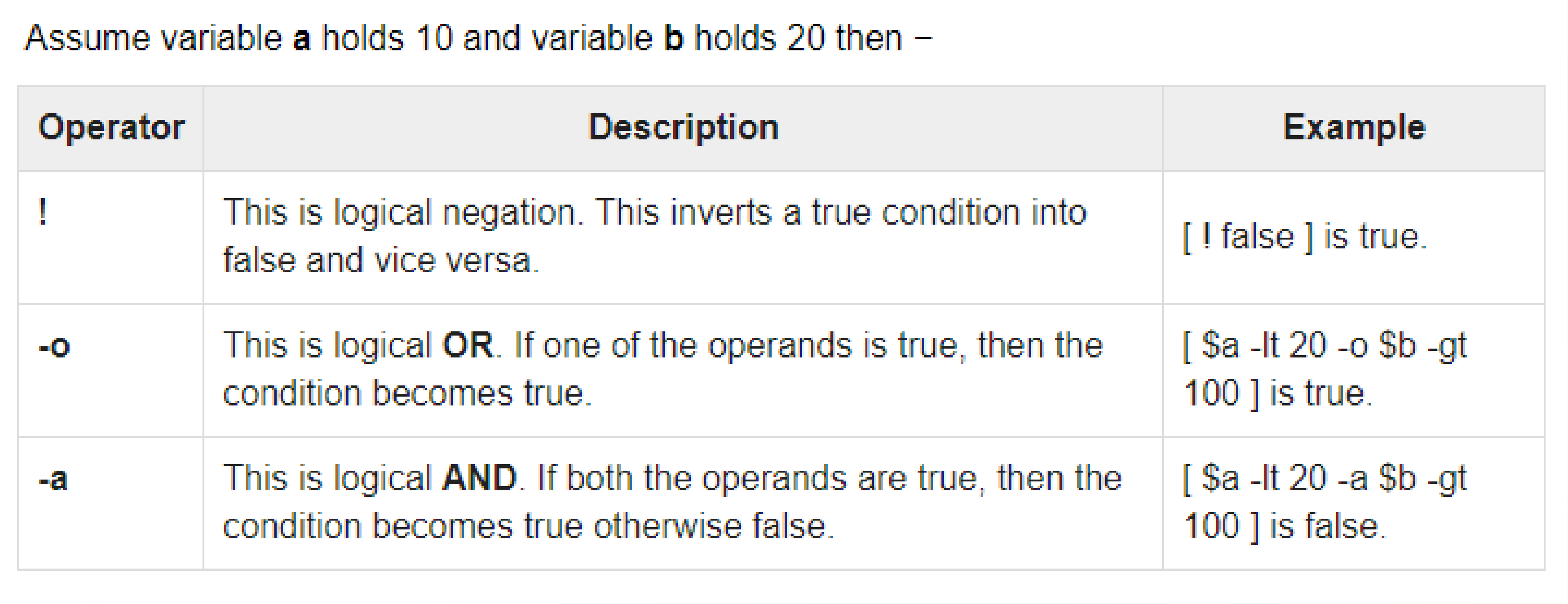
Assume variable **a** holds 10 and variable **b** holds 20 then −

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

SHELL SCRIPTING

Condition Statements

- Logical Operators



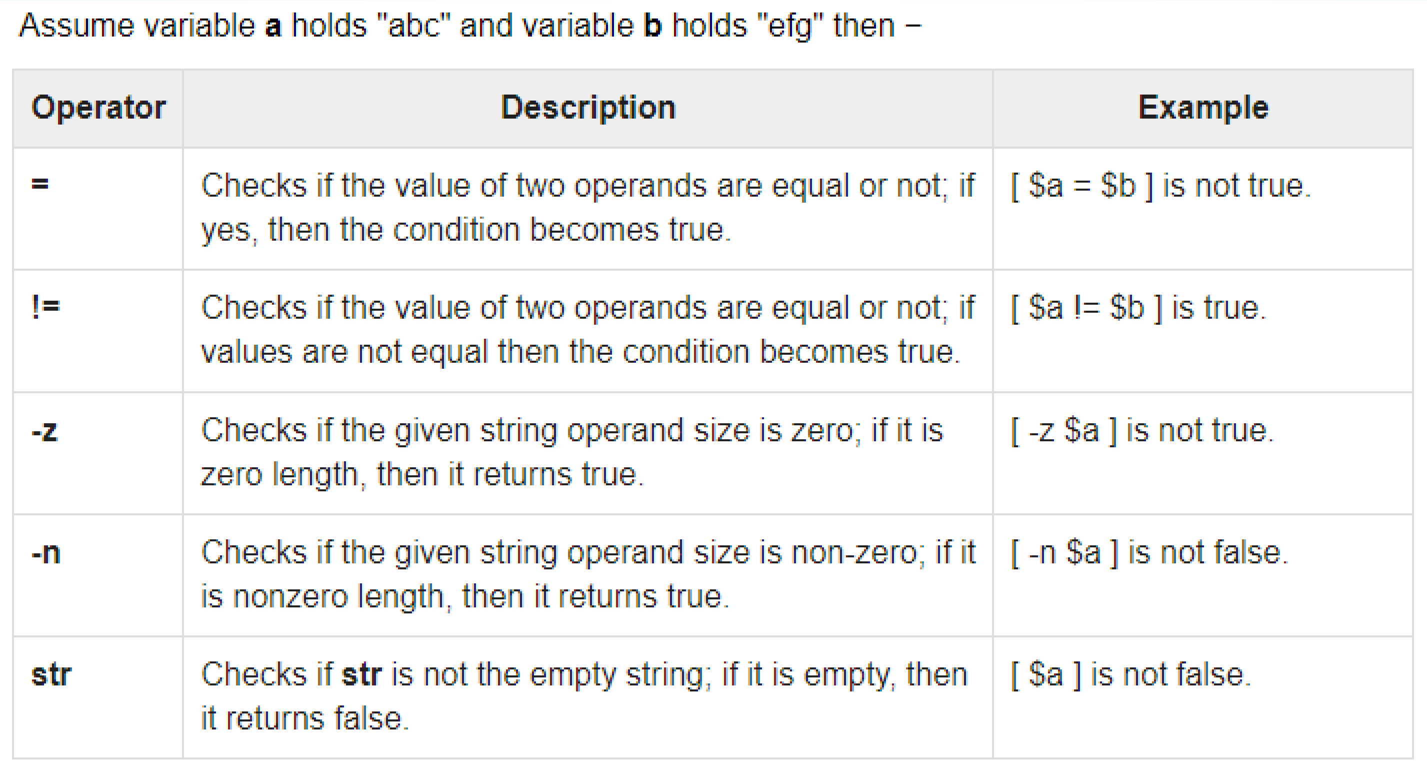
Assume variable **a** holds 10 and variable **b** holds 20 then −

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

SHELL SCRIPTING

Condition Statements

- String Comparison Operators



Assume variable **a** holds "abc" and variable **b** holds "efg" then −

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
str	Checks if **str** is not the empty string; if it is empty, then it returns false.	[$a] is not false.

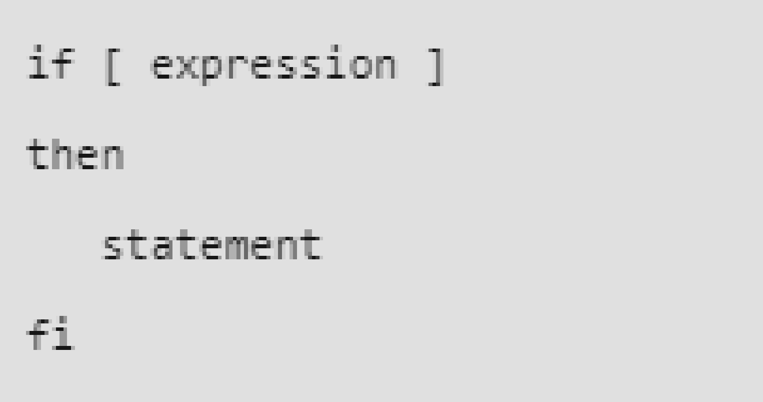
SHELL SCRIPTING

Condition Statements

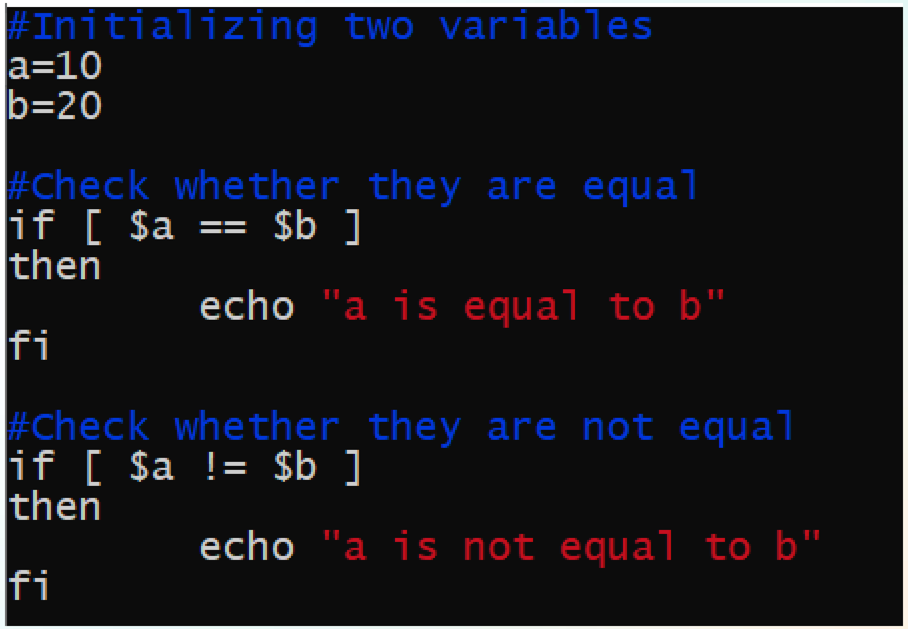
- **if condition**

  It is used for decision making in shell script, if given condition is true then command1 is executed.

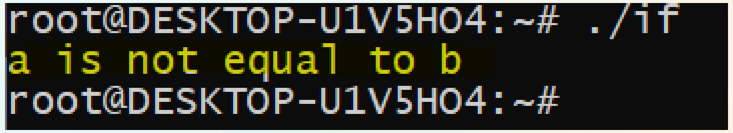
  Syntax:



```
if [ expression ]

then

   statement

fi
```



```
#Initializing two variables
a=10
b=20

#Check whether they are equal
if [ $a == $b ]
then
        echo "a is equal to b"
fi

#Check whether they are not equal
if [ $a != $b ]
then
        echo "a is not equal to b"
fi
```



```
root@DESKTOP-U1V5HO4:~# ./if
a is not equal to b
root@DESKTOP-U1V5HO4:~#
```

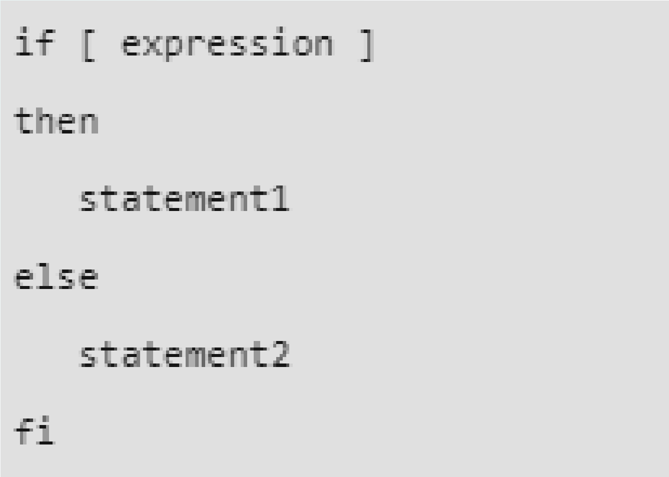
SHELL SCRIPTING

Condition Statements

- **if-else statement**

  If specified condition is not true in if part, then else part will be executed.

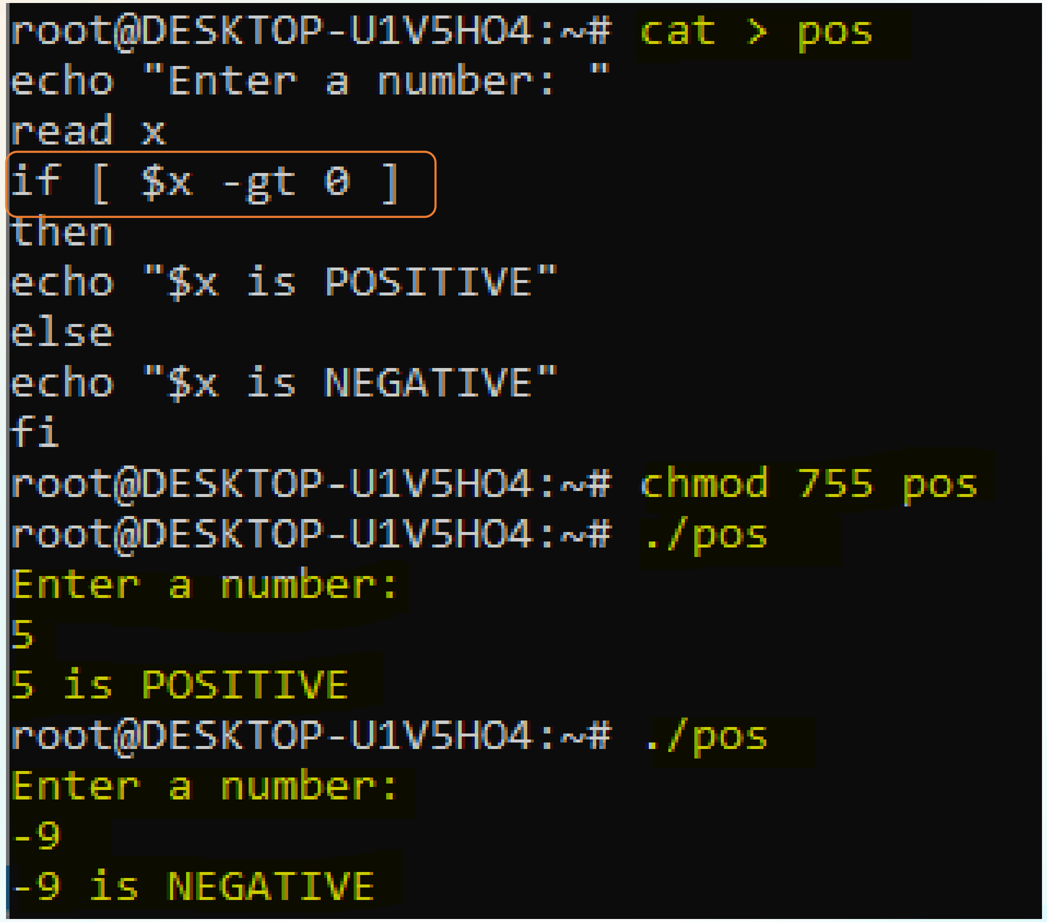
Syntax:



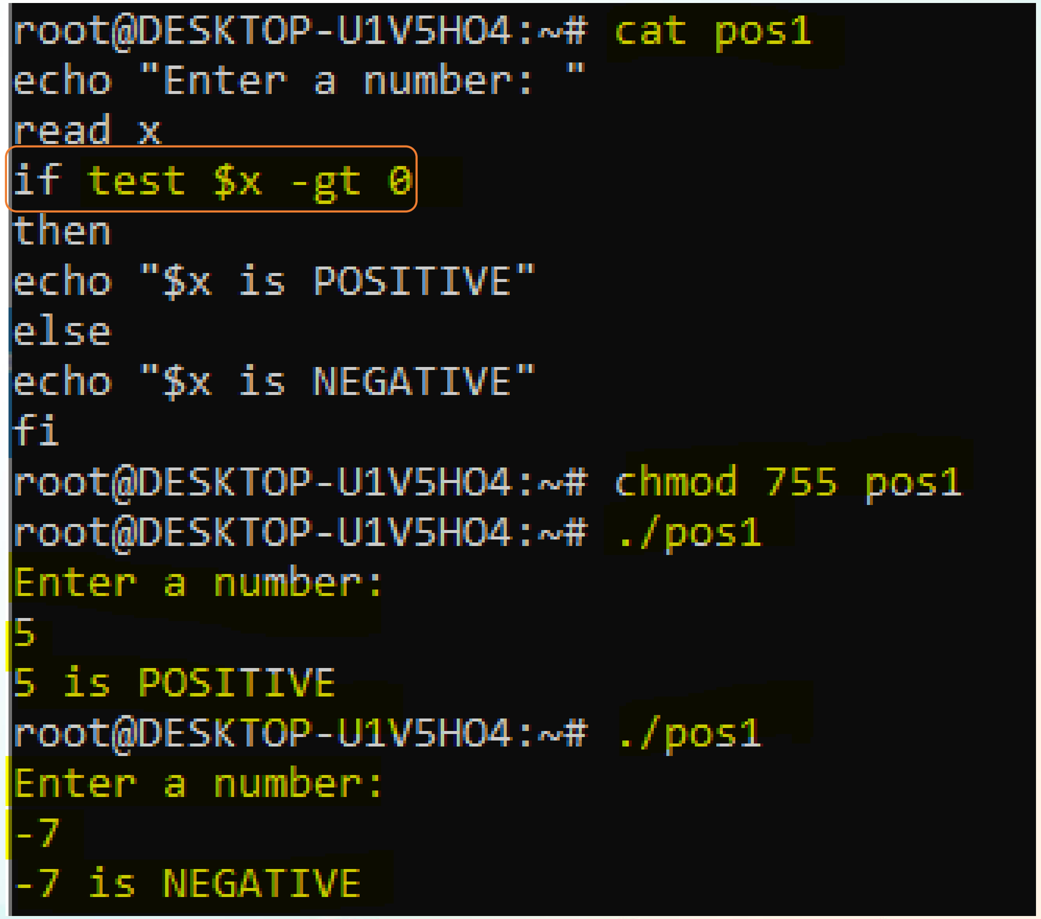
```
if [ expression ]

then

   statement1

else

   statement2

fi
```

SHELL SCRIPTING

if – else statement



```
root@DESKTOP-U1V5HO4:~# cat > pos
echo "Enter a number: "
read x
if [ $x -gt 0 ]
then
echo "$x is POSITIVE"
else
echo "$x is NEGATIVE"
fi
root@DESKTOP-U1V5HO4:~# chmod 755 pos
root@DESKTOP-U1V5HO4:~# ./pos
Enter a number:
5
5 is POSITIVE
root@DESKTOP-U1V5HO4:~# ./pos
Enter a number:
-9
-9 is NEGATIVE
```



```
root@DESKTOP-U1V5HO4:~# cat pos1
echo "Enter a number: "
read x
if test $x -gt 0
then
echo "$x is POSITIVE"
else
echo "$x is NEGATIVE"
fi
root@DESKTOP-U1V5HO4:~# chmod 755 pos1
root@DESKTOP-U1V5HO4:~# ./pos1
Enter a number:
5
5 is POSITIVE
root@DESKTOP-U1V5HO4:~# ./pos1
Enter a number:
-7
-7 is NEGATIVE
```

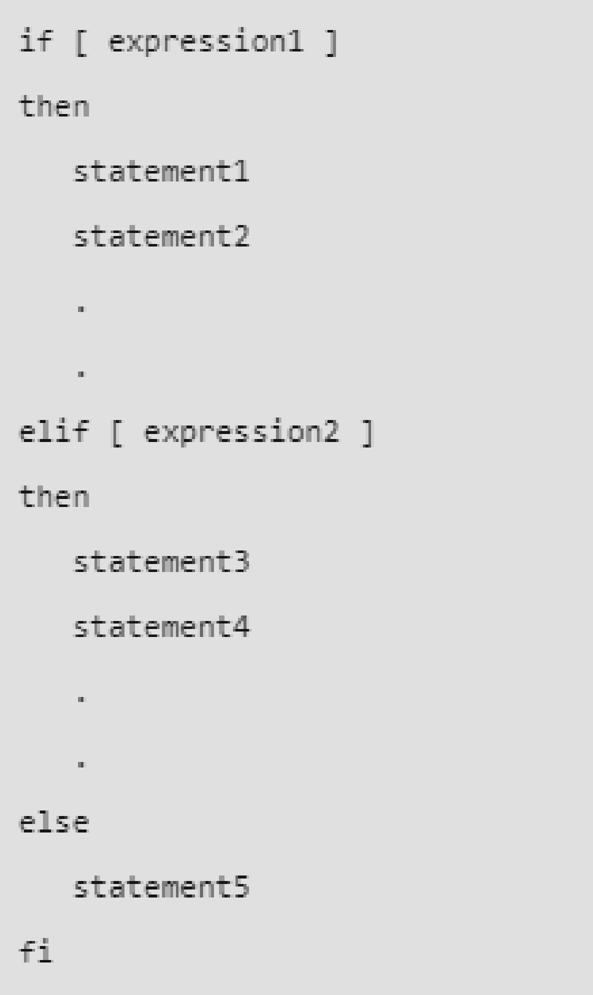
SHELL SCRIPTING

Condition Statements

if..elif..else..fi statement (Else If ladder)

- To use multiple conditions in one if-else block, then ***elif*** keyword is used in shell.
- If expression1 is true then it executes statement 1 and 2, and this process continues.
- If none of the condition is true then it processes else part.

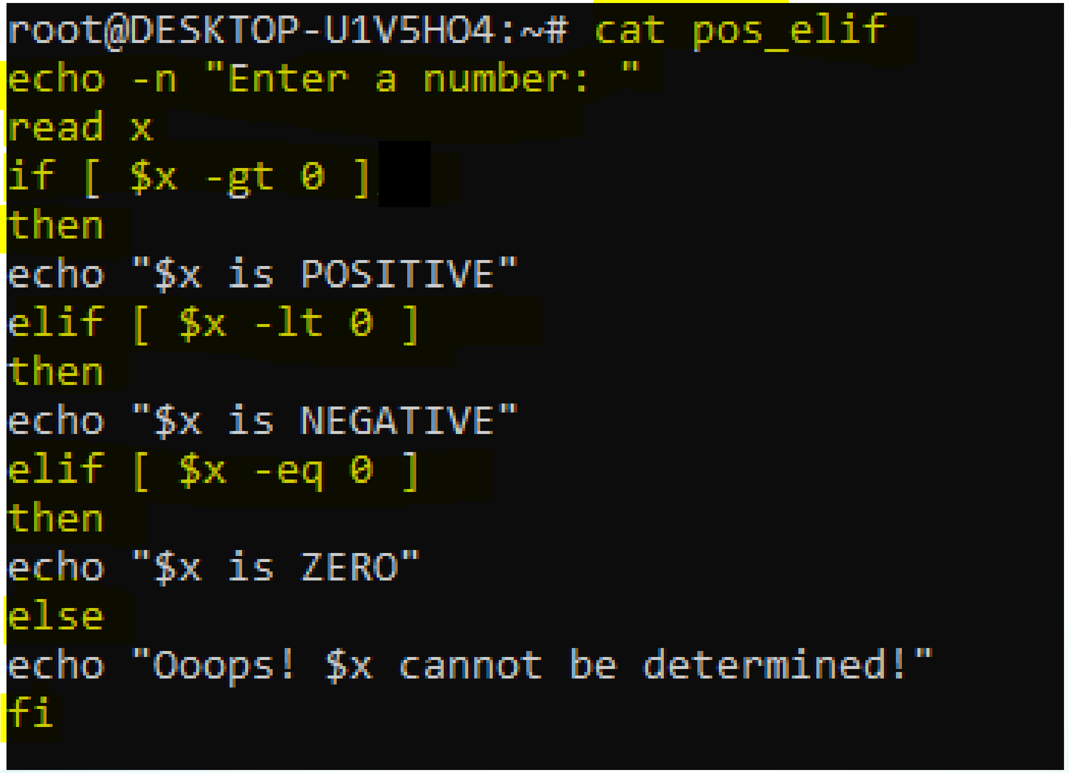
Syntax:



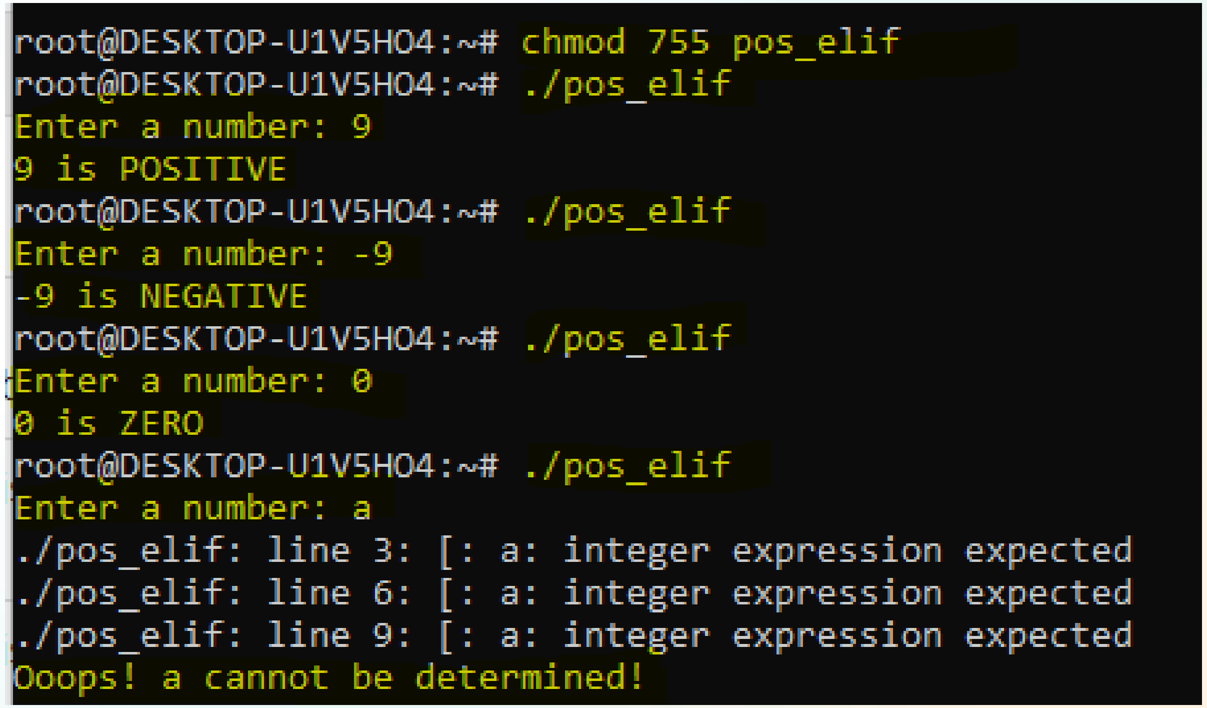
```
if [ expression1 ]
then
   statement1
   statement2
   .
   .
elif [ expression2 ]
then
   statement3
   statement4
   .
   .
else
   statement5
fi
```

SHELL SCRIPTING

if..elif..else..fi statement (Else If ladder)



```
root@DESKTOP-U1V5HO4:~# cat pos_elif
echo -n "Enter a number: "
read x
if [ $x -gt 0 ]
then
echo "$x is POSITIVE"
elif [ $x -lt 0 ]
then
echo "$x is NEGATIVE"
elif [ $x -eq 0 ]
then
echo "$x is ZERO"
else
echo "Ooops! $x cannot be determined!"
fi
```



```
root@DESKTOP-U1V5HO4:~# chmod 755 pos_elif
root@DESKTOP-U1V5HO4:~# ./pos_elif
Enter a number: 9
9 is POSITIVE
root@DESKTOP-U1V5HO4:~# ./pos_elif
Enter a number: -9
-9 is NEGATIVE
root@DESKTOP-U1V5HO4:~# ./pos_elif
Enter a number: 0
0 is ZERO
root@DESKTOP-U1V5HO4:~# ./pos_elif
Enter a number: a
./pos_elif: line 3: [: a: integer expression expected
./pos_elif: line 6: [: a: integer expression expected
./pos_elif: line 9: [: a: integer expression expected
Ooops! a cannot be determined!
```

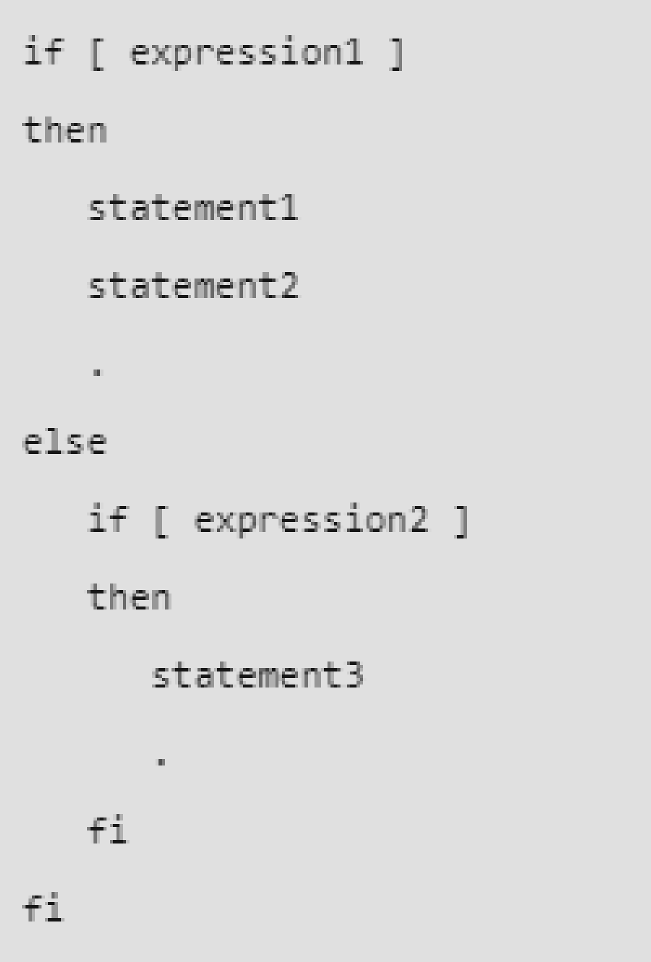
SHELL SCRIPTING

Condition Statements

if..then..else..if..then..fi..fi..(Nested if)

- Nested if-else block can be used when, one condition is satisfies then it again checks another condition.
- In the syntax, if expression1 is false then it processes else part, and again expression2 will be checked.

Syntax:

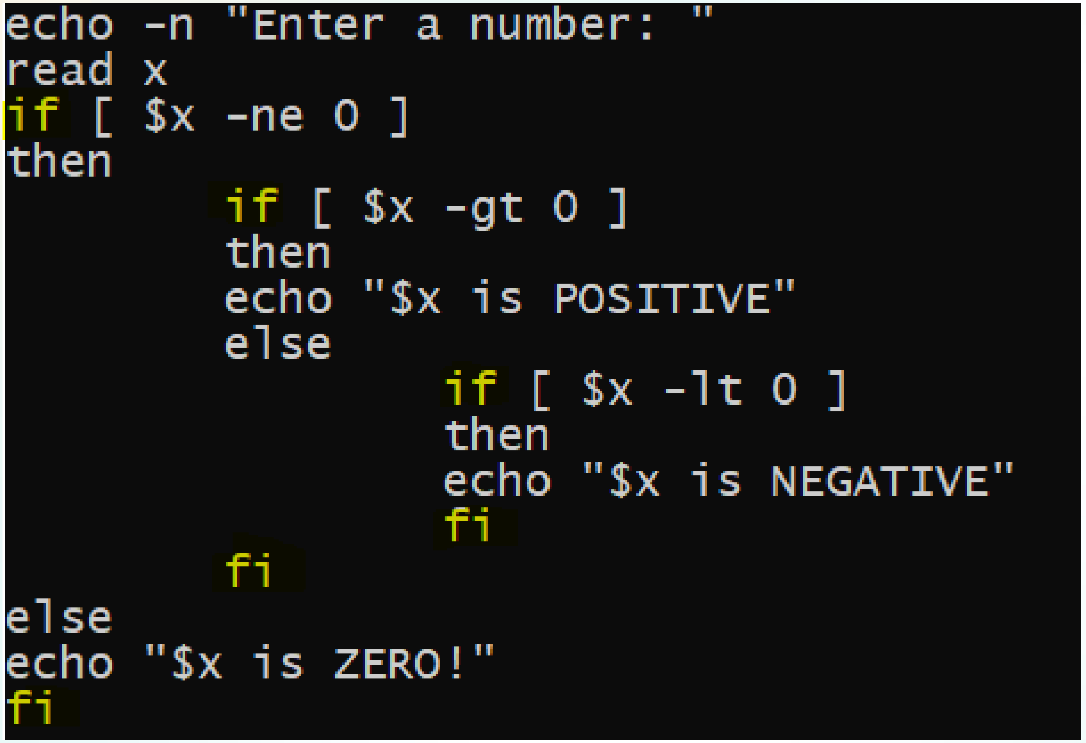


```
if [ expression1 ]
then
   statement1
   statement2
   .
else
   if [ expression2 ]
   then
      statement3
      .
   fi
fi
```

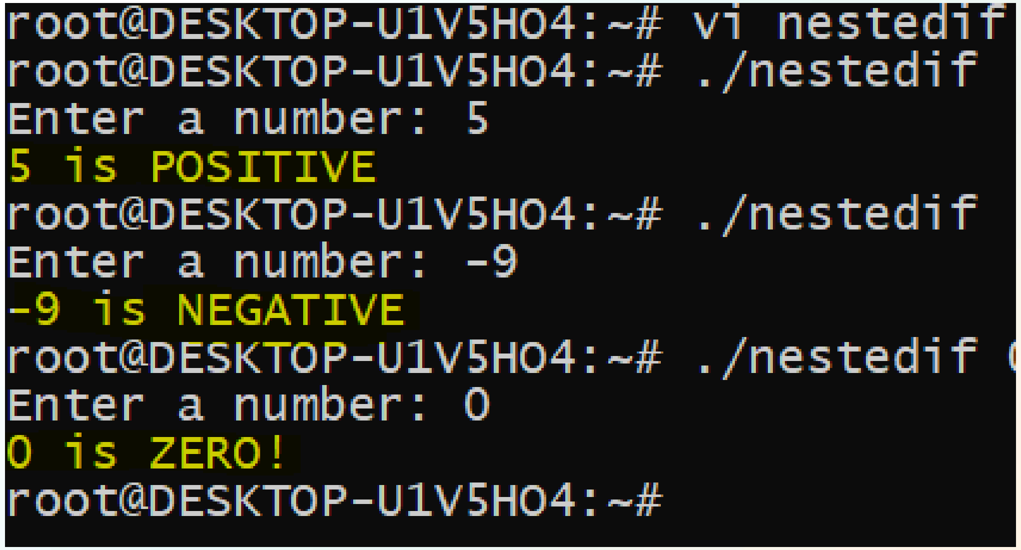
SHELL SCRIPTING

Condition Statements

- if..then..else..if..then..fi..fi..(Nested if)



```
echo -n "Enter a number: "
read x
if [ $x -ne 0 ]
then
        if [ $x -gt 0 ]
        then
        echo "$x is POSITIVE"
        else
                if [ $x -lt 0 ]
                then
                echo "$x is NEGATIVE"
                fi
        fi
else
echo "$x is ZERO!"
fi
```



```
root@DESKTOP-U1V5HO4:~# vi nestedif
root@DESKTOP-U1V5HO4:~# ./nestedif
Enter a number: 5
5 is POSITIVE
root@DESKTOP-U1V5HO4:~# ./nestedif
Enter a number: -9
-9 is NEGATIVE
root@DESKTOP-U1V5HO4:~# ./nestedif
Enter a number: 0
0 is ZERO!
root@DESKTOP-U1V5HO4:~#
```

SHELL SCRIPTING

Case Statement

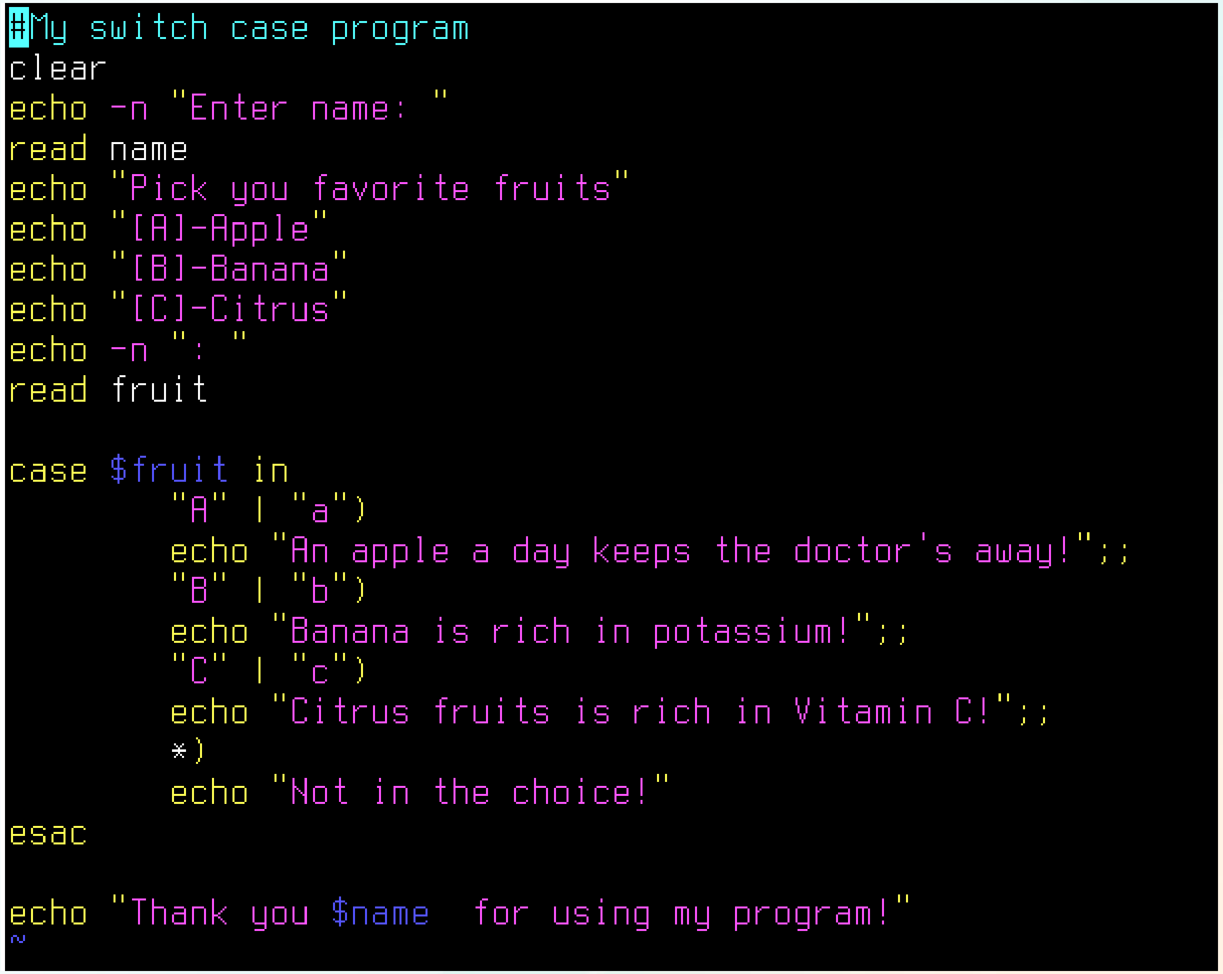
- case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern
- When a match is found all of the associated statements until the double semicolon (;;) is executed.
- A case will be terminated when the last command is executed.
- If there is no match, the exit status of the case is zero.

Syntax:

```
case  $variable-name  in
         pattern1)   statement
                  ...
                  statement;;
         pattern2) statement
                  ...
                  statement;;
         patternN) statement
                  ...
                  statement;;
          *) statement
                  ...
                  statement;;
  esac
```

SHELL SCRIPTING

Switch-case statement



```
#My switch case program
clear
echo -n "Enter name: "
read name
echo "Pick you favorite fruits"
echo "[A]-Apple"
echo "[B]-Banana"
echo "[C]-Citrus"
echo -n ": "
read fruit

case $fruit in
        "A" | "a")
        echo "An apple a day keeps the doctor's away!";;
        "B" | "b")
        echo "Banana is rich in potassium!";;
        "C" | "c")
        echo "Citrus fruits is rich in Vitamin C!";;
        *)
        echo "Not in the choice!"
esac

echo "Thank you $name  for using my program!"
~
```

REFERENCES

- Sobell, M., et al. (2017). A Practical Guide to Linux Commands, Editors, and Shell Programming, 4th Ed. Addison-Wesley Professional
- Cobbaut, P. (2016). Mastering Linux- Networking
- Blum, R., (2015). Linux Command Line and Shell Scripting Bible