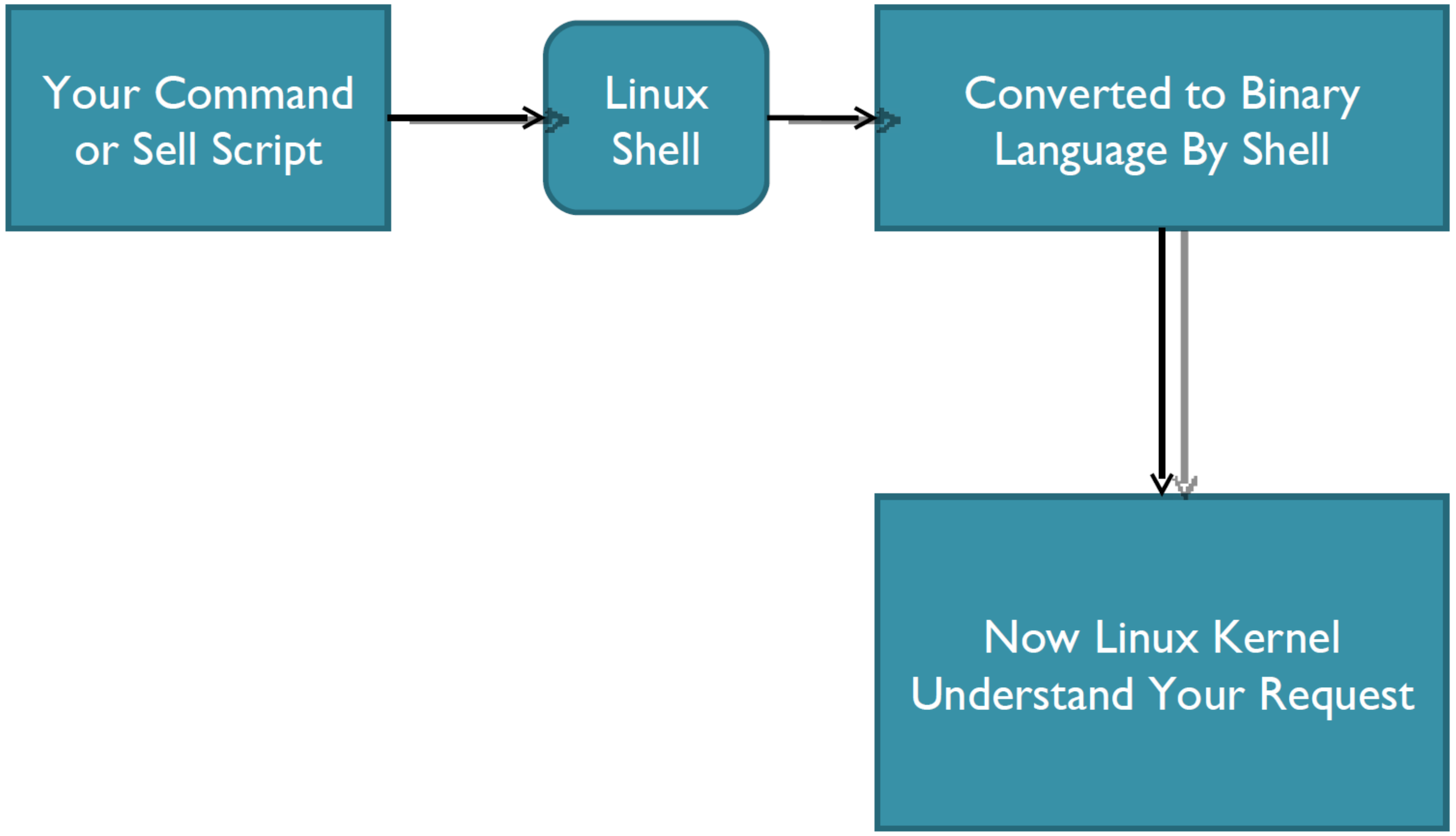
Shell Programming

What is Shell ?

- A *shell is a program that* ***takes commands*** *typed by the user and* ***calls the operating system*** *to run those commands*.
- A *shell is a program that* ***acts as the interface*** *between you and the Linux system, allowing you to enter* commands for the operating system to execute.
- Shell accepts your instruction or commands in English and translate it into computers native binary language

Task of Shell



Your Command or Sell Script
Linux Shell
Converted to Binary Language By Shell
Now Linux Kernel Understand Your Request

Shell Programs

- There are two ways of writing shell programs.

1. You can **type a sequence of commands** and allow the shell to execute them interactively.
2. You can **store those commands in a file** that you can then invoke as a program(shell script).

Shell Scripting

- Shell script is a series of command(s) stored in a plain text file.

Information Command

```
man commandName
info commandName
commandName -h
commandName --help
```

Each shell script consists of

- Shell keywords such as if..else, do..while.
- Shell commands such as pwd, test, echo, continue, type.
- Linux binary commands such as w, who, free etc..
- Text processing utilities such as grep, awk, cut.

Keywords in Shell Programming

echo	read	set	unset
readonly	shift	export	if
fi	else	while	do
done	for	until	case
esac	break	continue	exit
return	trap	wait	eval
exec	ulimit	umask	

Create a Script

- Terminal

- Vi filename.sh

What is the VI editor?

- The VI editor is the most popular and classic text editor in the Linux family.

  1) It is available in almost all Linux Distributions
  2) It works the same across different platforms and Distributions
  3) It is user-friendly.

Operation modes.

- **vi Command mode:**
- The vi editor opens in this mode, and it only **understands commands**
- you can, **move the cursor and cut, copy, paste the text**
- This mode **also saves the changes** you have made to the file
- **Commands are case sensitive.**
- You should use the right letter case.

Operation modes.

- **vi Editor Insert mode:**
- To insert text in the file.
- You can switch to the Insert mode from the command mode **by pressing 'i' on the keyboard**
- Once you are in Insert mode, any key would be taken as an input for the file on which you are currently working.
- **To return to the command mode** and save the changes you have made **you need to press the Esc key**

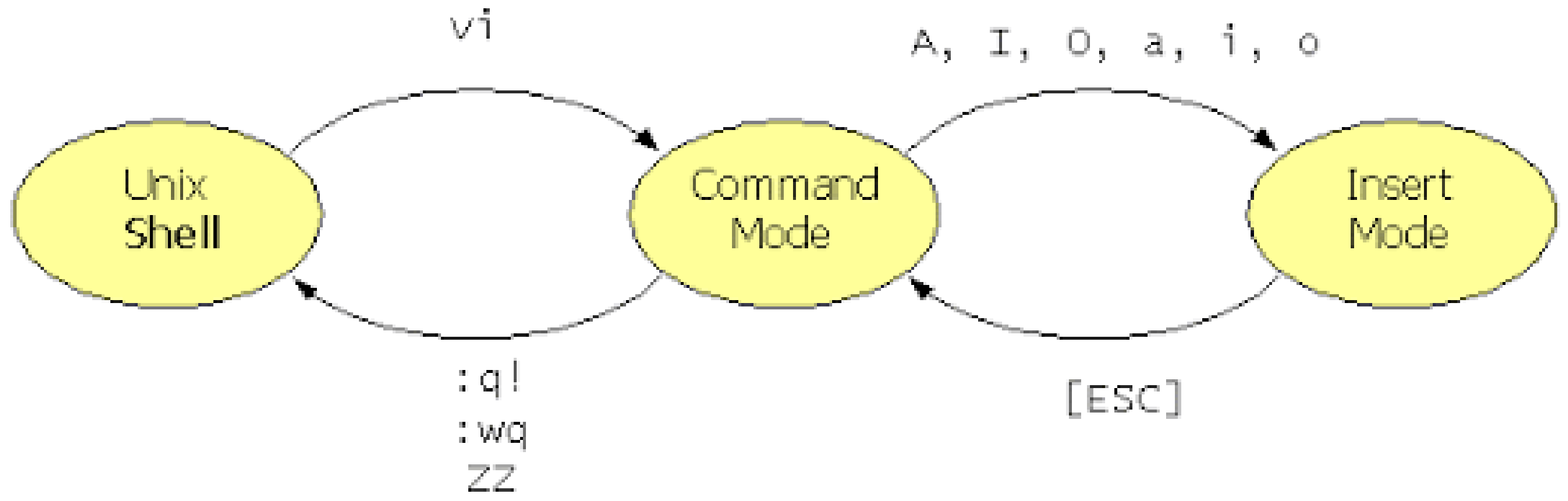
VI Editing commands

- i - Insert at cursor (goes into insert mode)
- a - Write after cursor (goes into insert mode)
- A - Write at the end of line (goes into insert mode)
- ESC - Terminate insert mode
- u - Undo last change
- U - Undo all changes to the entire line
- o - Open a new line (goes into insert mode)
- dd - Delete line
- 3dd - Delete 3 lines.
- D - Delete contents of line after the cursor

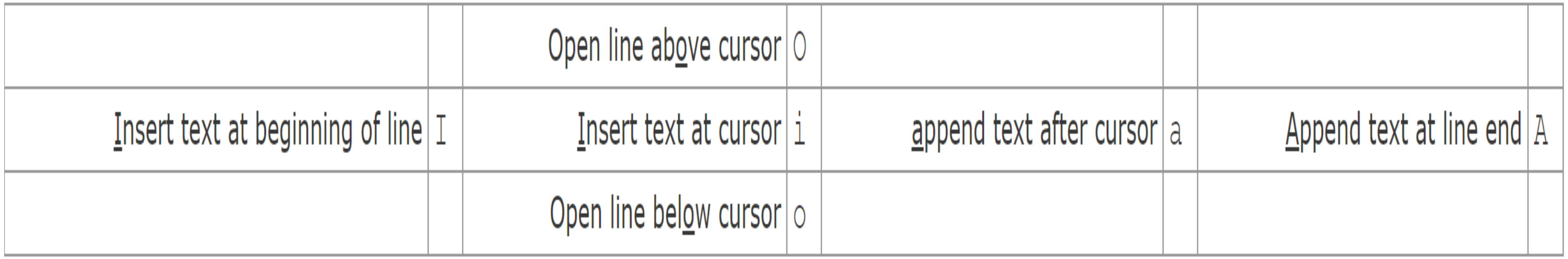
VI Editing commands

- C - Delete contents of a line after the cursor and insert new text. Press ESC key to end insertion.
- dw - Delete word
- 4dw - Delete 4 words
- cw - Change word
- x - Delete character at the cursor
- r - Replace character
- R - Overwrite characters from cursor onward
- s - Substitute one character under cursor continue to insert
- S - Substitute entire line and begin to insert at the beginning of the line
- ~ - Change case of individual character

Vi Mode



Insert Text



		Open line above cursor	O				
Insert text at beginning of line	I	Insert text at cursor	i	append text after cursor	a	Append text at line end	A
		Open line below cursor	o				

Switch to Command mode:

Switch to command mode: [ESC]

Execute Shell

- Syntax to setup executable permission:
- $ chmod +x your-script-name.
- $ chmod 755 your-script-name.
- Execute By:

```
sh your-script-name

./your-script-name
```

Example

```
$ vi first
```

```
#
# My first shell script
#
clear
echo “This is my First
script”
```

```
$ chmod 755 first

$ ./first
```

Comments

- # Symbol for Comments in Shell

x=12   Echo $x   Echo $y $y=0

Echo $y Echo $0

Ksh

Echo $0

Echo $x

Exit

Export x : make x global

Variable

- 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 letters.

User Defined Variable

- To define UDV use following **syntax:**
- variable name = value $ no=10
- **Rules for Naming variable name**
- Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character .
- You **can define NULL** variable
- Variables are case-sensitive.

Print or Access USD Variable

```
$a = 10
echo $a
```

defined string variable in linux the following example

```
$name ='Vishwa'
echo $name;
```

- You must ***use $*** *followed by variable name*.

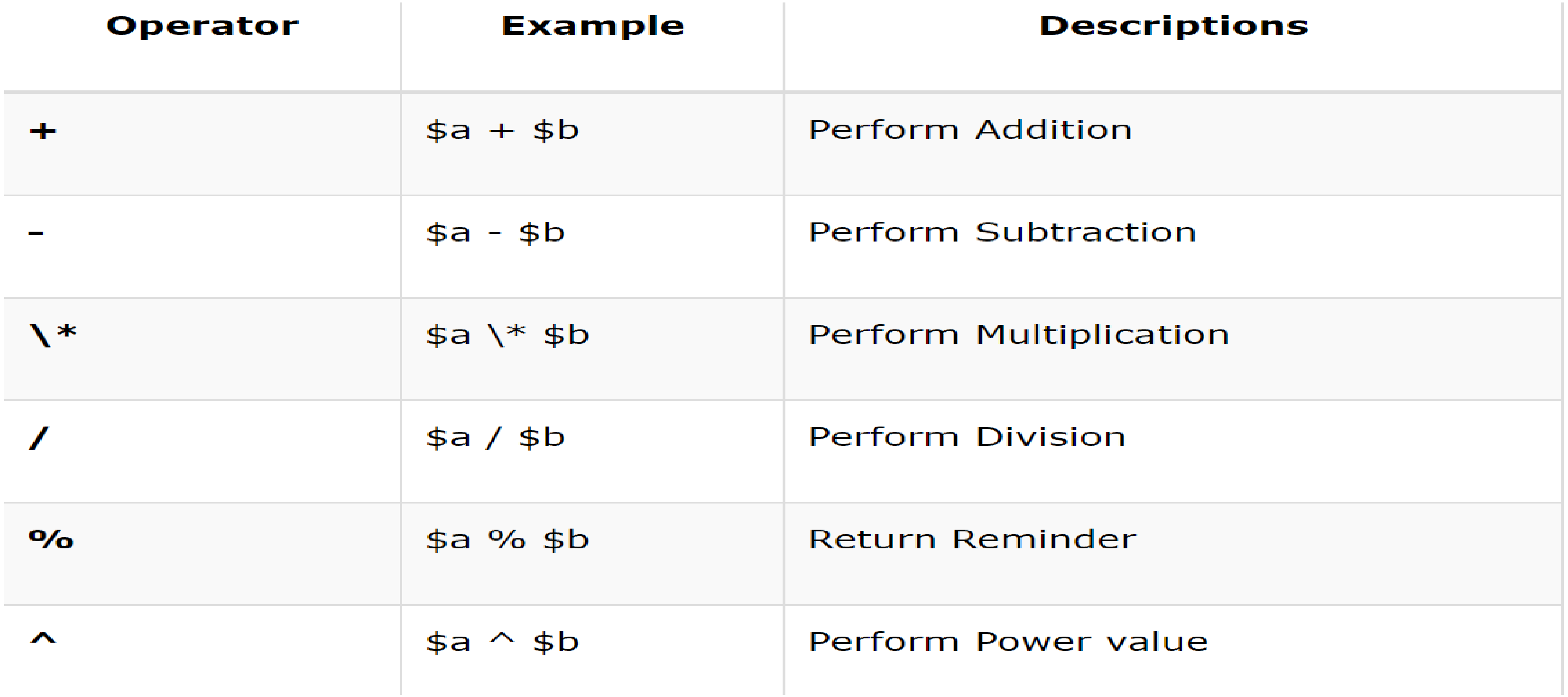
Currently log in users

- Users
- Users| wc –w
- ---
- **Vi s1.sh**
    - Echo current logged in users `who | wc-l `
    - Echo current logged in users `who |  -q `
    - Echo current logged in users count is `users | wc –w  `
    - Echo “done”

Example Shell Script :**s2.sh**

- Hostname –d **#domain Name**
- Hostname –I **#IP address**
- Date +%u **#current week Number**
- Uname –p **#Name of Processor**
- Date +%F **#date in yyyy/mm/dd format**
- Cal –j **#day number in a year**

Arithmetic Operators



Operator	Example	Descriptions
+	$a + $b	Perform Addition
-	$a - $b	Perform Subtraction
*	$a * $b	Perform Multiplication
/	$a / $b	Perform Division
%	$a % $b	Return Reminder
^	$a ^ $b	Perform Power value

Shell Arithmetic

- ***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`

Read Statement

- Use to get input (data from user) from keyboard and store (data) to variable.
- ***Syntax:***
- read variable1, variable2,...variable
- **Example:**
- echo "Your first name please:"
- read fname
- echo "Hello $fname !"

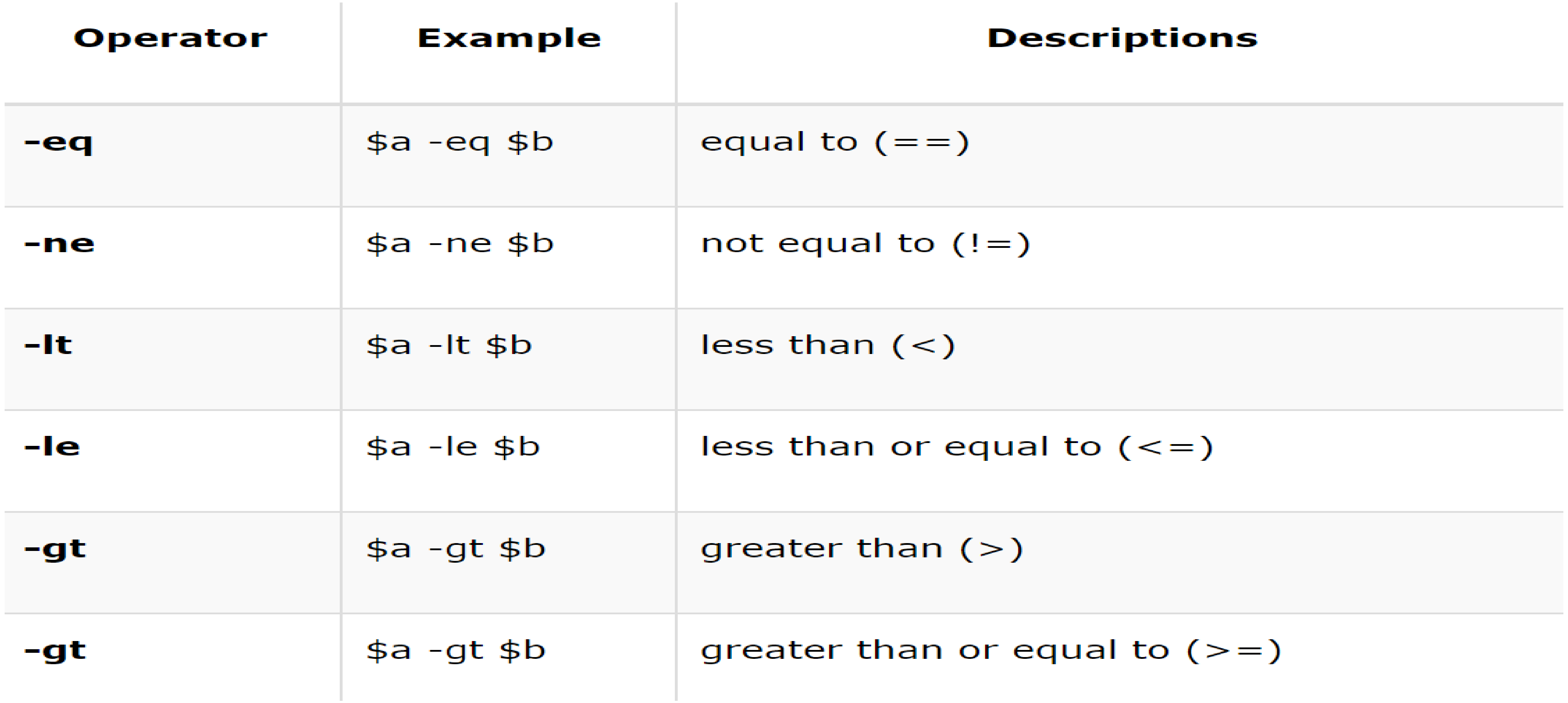
Example : s3.sh

- Echo –n “Enter Number”
- Read no1
- Echo –n “Enter Number”
- Read no2
- Echo “sum of 2 Number is:” $[no1+no2]
- Echo “diff of 2 Number is:” $[no1-no2]
- Echo “Product of 2 Number is:” $[no1*no2]

Shorthand

`$ ls *`	will show all files
`$ ls a*`	will show all files whose first name is starting with letter 'a'
`$ ls *.c`	will show all files having extension .c
`$ ls ut*.c`	will show all files having extension .c but file name must begin with 'ut'.
`$ ls ?`	will show all files whose names are 1 character long
`$ ls fo?`	will show all files whose names are 3 character long and file name begin with fo

Relational Operators

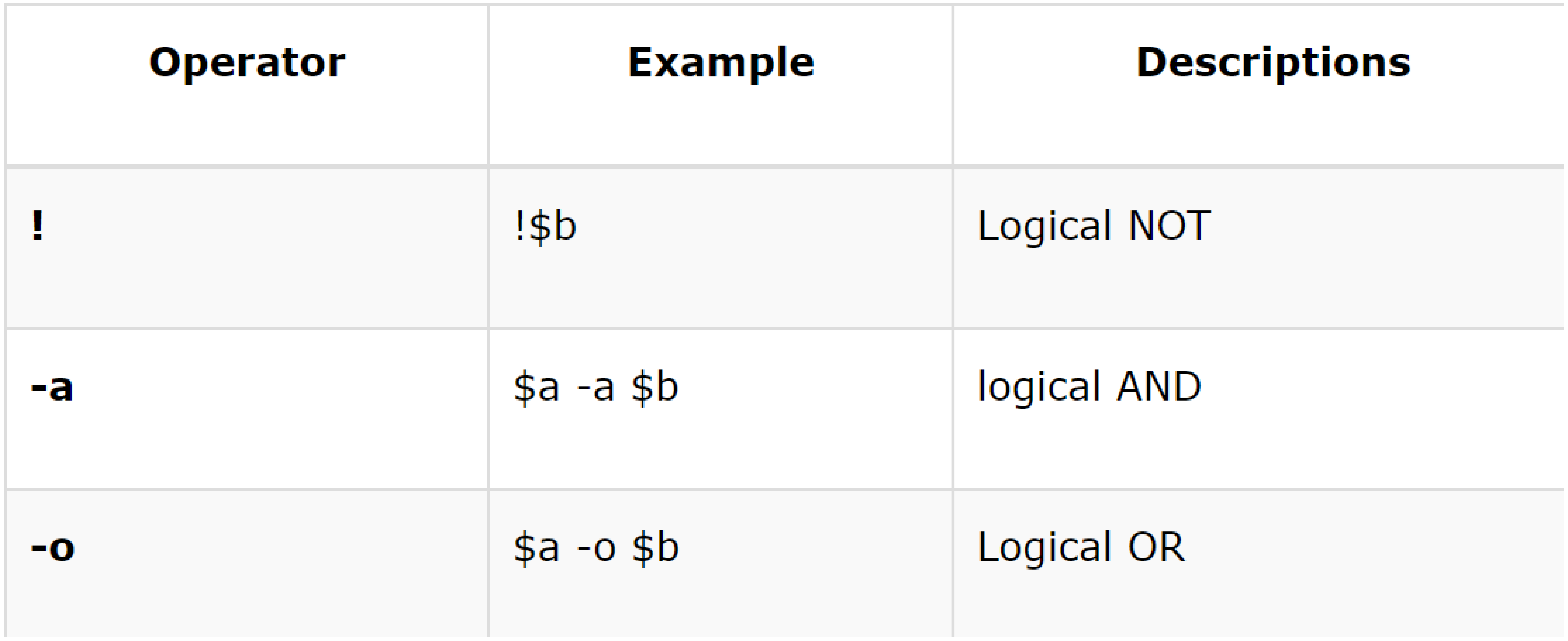


Operator	Example	Descriptions
-eq	$a -eq $b	equal to (==)
-ne	$a -ne $b	not equal to (!=)
-lt	$a -lt $b	less than (<)
-le	$a -le $b	less than or equal to (<=)
-gt	$a -gt $b	greater than (>)
-gt	$a -gt $b	greater than or equal to (>=)

Example

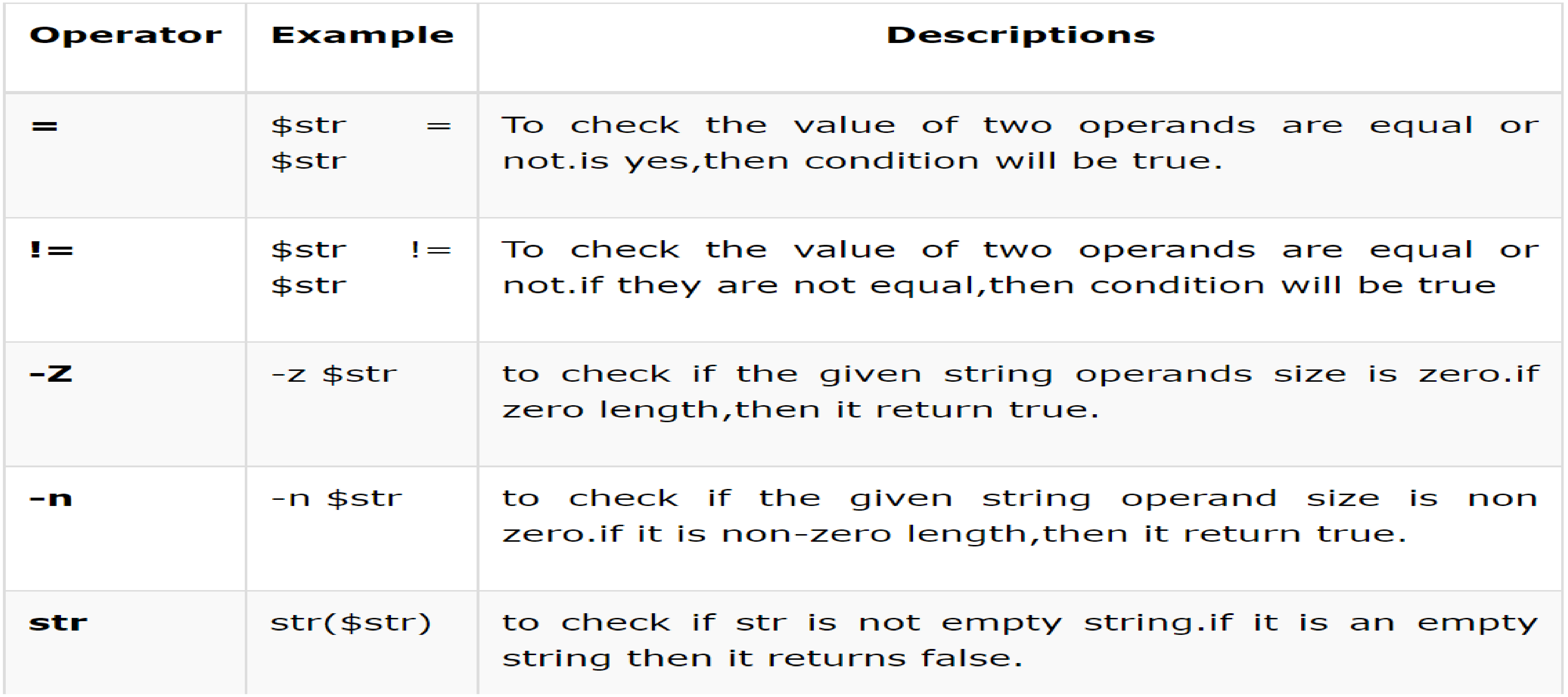
```
$ vi  myscript.sh

 read choice

        if [ $choice -gt 0 ] ; then

                echo "$choice number is positive"

        else

                echo "$ choice number is negative"

        fi
```

Logical Operators



Operator	Example	Descriptions
!	!$b	Logical NOT
-a	$a -a $b	logical AND
-o	$a -o $b	Logical OR

String Operators



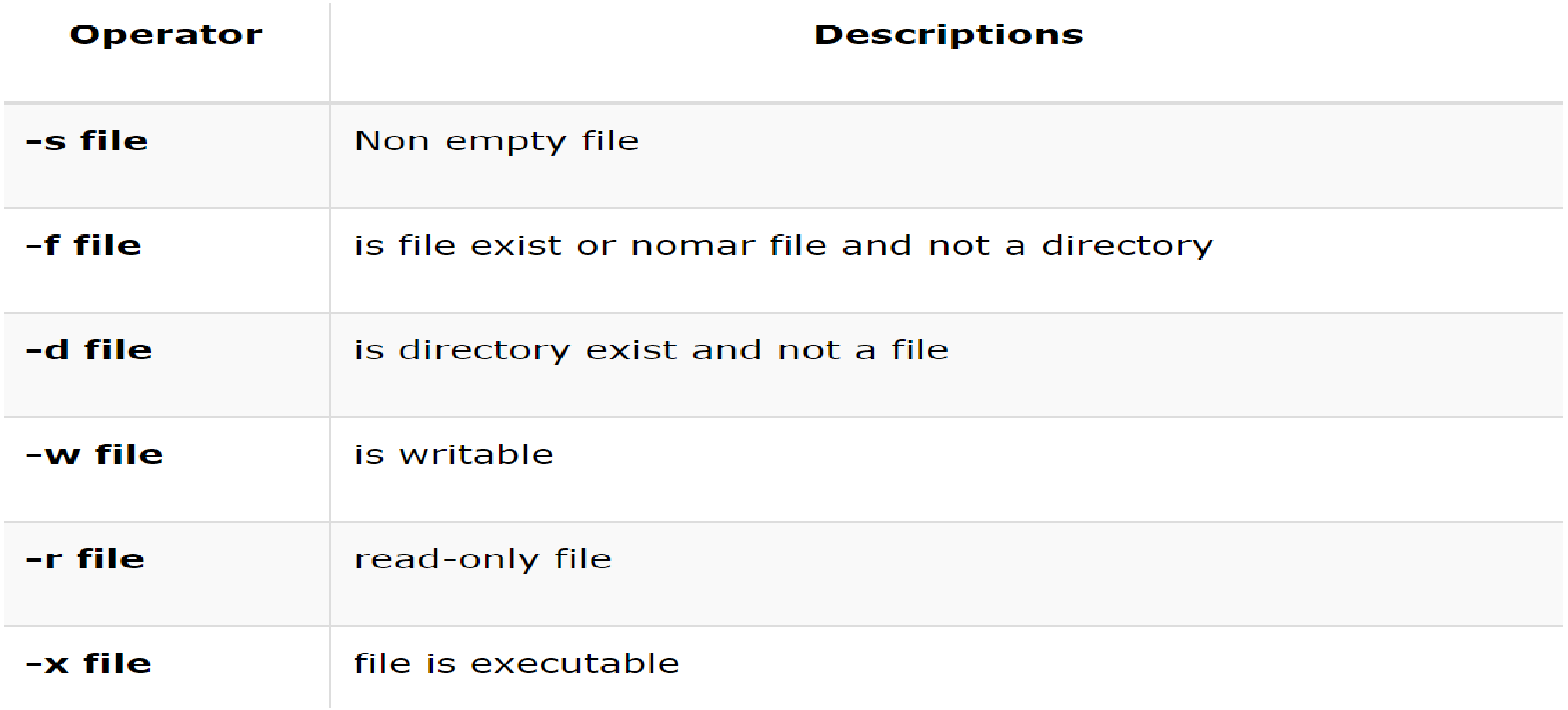
Operator	Example	Descriptions
=	$str = $str	To check the value of two operands are equal or not.is yes,then condition will be true.
!=	$str != $str	To check the value of two operands are equal or not.if they are not equal,then condition will be true
-Z	-z $str	to check if the given string operands size is zero.if zero length,then it return true.
-n	-n $str	to check if the given string operand size is non zero.if it is non-zero length,then it return true.
str	str($str)	to check if str is not empty string.if it is an empty string then it returns false.

String Operators

```
str="UVPCE";
if[ -n $str]
then
    echo"String is not empty";
else
    echo"String is empty";
fi
```

```
str=" ";
if [ -z  $str ]
then
 echo"String is empty";
else
echo"String is  not empty";
fi
```

Test File & Directory Type



Operator	Descriptions
-s file	Non empty file
-f file	is file exist or nomar file and not a directory
-d file	is directory exist and not a file
-w file	is writable
-r file	read-only file
-x file	file is executable

Input output

- **I/O Redirection**
- **> -** Output to given file
- **< -** Read input from given file
- **>> -** Append output to given file
- **I/O Commands**
- **echo** – To print to stdout
- **read** – To obtain values from stdin

File system commands

- **mkdir –** Creates a directory
- **rmdir –** Deletes a directory
- **ls –** Lists contents of given path
- **cat –** Read from given file and output to STDOUT or given path
- **find –** Search for a given file (find <path> -name <filename>)
- **chmod** – Change mode/permissions
- **cp** - Copy files (cp sourcefile destfile)
- **mv** – Move/rename files (mv oldname newname)

Quotation

- unquoted strings are normally interpreted
- “ quoted strings are basically literals -- but $variables are evaluated “
- ‘ quoted strings are absolutely literally interpreted ‘
- ` commands in quotes are executed, their output is assigned to a variable and then that variable was evaluated `
  - $ echo `expr 6 + 3`

If Statement

```
if [ condition ]
then
        Execute the statements
fi
```

```
#Check Number is 1

echo "Enter Number:-"
read no

if [ $no -eq 1]
then
        echo "Number 1 "
fi
```

If ..Else Statement

```
if [ condition ]
then
    Execute Statement if Condition is True
elif
    Execute Statement if Condition is False
fi
```

```
echo "Enter Number:"
read no

if [ $no -gt 0 ]
then
    echo "Number is Positive"
elif
    echo "Number is Negative"
fi
```

Example

```
if [ "$NAME"=“Guni" ]
        then
                printf   “ %s  is logged in" $NAME
        else
                printf  “unknown”
fi
```

Printf command

w - Minimum field width.

- p - Display number of digits after the decimal point (precision).
- L - a conversion character. It can be:
- s - String
- d - Integer
- e - Exponential
- f - Floating point

Nested if-else-fi

```
if [ condition ]
then
    Execute Statement if Condition 1
elif [ condition ]
    Execute Statement if Condition 2
elif [ condition ]
    Execute Statement if Condition 3
elif
    Else Condition
fi
```

```
echo "Enter Student Mark:-"
read mark

if [ $mark -gt 70]
then
    echo "Distinction"
elif [ $mark -gt 60]
then
    echo "First Class"
elif [ $mark -gt 50]
then
    echo "Second Class"
elif [ $mark -gt 40]
then
    echo "Pass Class"
elif
    echo "Fail"
fi
```

Nested If

```
if [ condition ]
then
    if [ condition ]
        then
            Execute Statement
        elif
            Execute Statement
        fi
elif
    Execute Statement
fi
```

```
echo "Enter Your Country:"
read cn

if [$cn -eq 'India']
then
    echo "Enter Your State:"
        read st
    if [$st -gt 'Gujarat']
        then
            echo "Welcome to Gujarat"
        elif
            echo "You are Not Gujarati"
        fi
elif
    echo "Other Country"
fi
```

Case statement (Two ;; serve as the break)

```
case $[ variable_name ] in
value1)
    Statement 1
        ;;
value2)
    Statement 2
        ;;
value3)
    Statement 3
        ;;
value4)
    Statement 4
        ;;
valueN)
    Statement N
        ;;
*)
    Default Statement
        ;;
esac
```

```
echo "Enter Country Code:"
read co

case $co in
'IN') echo "India"
    ;;
'PK') echo "Pakistan"
    ;;
*) echo "Enter Vailid Country Code"
    ;;
esac
```

Until Loop

```
until [ condition ]
do
   statement 1
   statement 2
done
```

```
i=1
while [ !$i -lt 10 ]
do
    echo $i
        i=`expr $i + 1`
done
```

While Loop

```
while [ condition ]
do
    statement 1
    statement 2
done
```

```
i = 1
while [ $i -le 10 ]
do
    echo $i
        i = `expr $i + 1`
done
```

For loop

```
for [ variable_name ] in ...
do
    statement 1
        statement 2
        statement n
done
```

```
for no in {1..10}
do
    echo $no
done
```

For loop

- ***Syntax:***
- *for { variable name } in { list }*
- *do*
- *execute one for each item in the list until the list is not finished and repeat all statement between do and done*
- *done*

```
for i in 1 2 3 4 5
        do
                echo "Welcome $i times"
done
```

For Loop

- ***Syntax:***
- for ((expr1; **expr2;** expr3))
- do
- repeat all statements between

  do and done until **expr2 is TRUE**
- Done

```
for (( i = 0 ; i <= 5; i++ ))

 do

           echo "Welcome $i times"

done
```

Practical-2

1. Write a shell script to scan your college name and display it like **My college Name is :UVPCE**
2. Write a shell script to **scan two variables** and to **display their sum, mul, div, sub and modulo division.**
3. Write a shell script to **scan two variables** and to display their sum, mul, div, sub and modulo division **as per the user choice**. (no need to continue, only once is OK)

Practical-2

4. Write a shell script to **find greatest of two**. Script must consider the case where two numbers are equal.
5. Write a shell script to **find greatest of three**. Script must consider the case where two numbers are equal.