

EXPERIMENT NO:4

AIM :: SHELL SCRIPTING

Bash:-Bash is a command language interpreter. It is widely available on various operating systems and is a default command interpreter on most GNU/Linux systems. The name is an acronym for the 'Bourne-Again SHell'.

Shell:-Shell is a macro processor which allows for an interactive or non-interactive command execution.

Scripting:-Scripting allows for an automatic commands execution that would otherwise be executed interactively one-by-one.

SHELL SCRIPTING

A shell script is a text file that contains a sequence of commands for a UNIX-based operating system. It is called a shell script because it combines a sequence of commands, that would otherwise have to be typed into the keyboard one at a time, into a single script. The shell is the operating system's command-line interface (CLI) and interpreter for the set of commands that are used to communicate with the system.

A shell script is usually created for command sequences in which a user has a need to use repeatedly in order to save time. Like other programs, the shell script can contain parameters, comments and subcommands that the shell must follow. Users initiate the sequence of commands in the shell script by simply entering the file name on a command line.

In the DOS operating system, a shell script is called a batch file. In IBM's mainframe VM operating systems, it's called an EXEC.

BASH (Bourne Again SHell), which is the most widely used shell and also the default in most Linux systems, which most servers use as their operating system. Shell scripting is particularly useful for DevOps and backend development.

There are two major types of shells –

- **Bourne shell** – If you are using a Bourne-type shell, the \$ character is the default prompt.
- **C shell** – If you are using a C-type shell, the % character is the default prompt.

The basic steps involved with shell scripting are writing the script, making the script accessible to the shell and giving the shell execute permission.

Shell scripts contain ASCII text and are written using a text editor, word processor or graphical user interface (GUI). The content of the script is a series of commands in a language that can be interpreted by the shell. Functions that shell scripts support include loops, variables, if/then/else statements, arrays and shortcuts. Once complete, the file is saved typically with a .txt or .sh extension and in a location that the shell can access.

ADVANTAGES

- Easy to use.

- Quick start, and interactive debugging.
- Time Saving.
- Sys Admin task automation.(system admin)
- Shell scripts can execute without any additional effort on nearly any modern UNIX / Linux / BSD / Mac OS X operating system as they are written in an interpreted language.

DISADVANTAGES

- Compatibility problems between different platforms.
- Slow execution speed.
- A new process launched for almost every shell command executed.

Shell Environment

A shell maintains an environment that includes a set of variables defined by the login program, the system initialization file, and the user initialization files. In addition, some variables are defined by default.

A shell can have two types of variables:

- **Environment variables** – Variables that are exported to all processes spawned by the shell. Their settings can be seen with the env command. A subset of environment variables, such as PATH, affects the behavior of the shell itself.
- **Shell (local) variables** – Variables that affect only the current shell. In the C shell, a set of these shell variables have a special

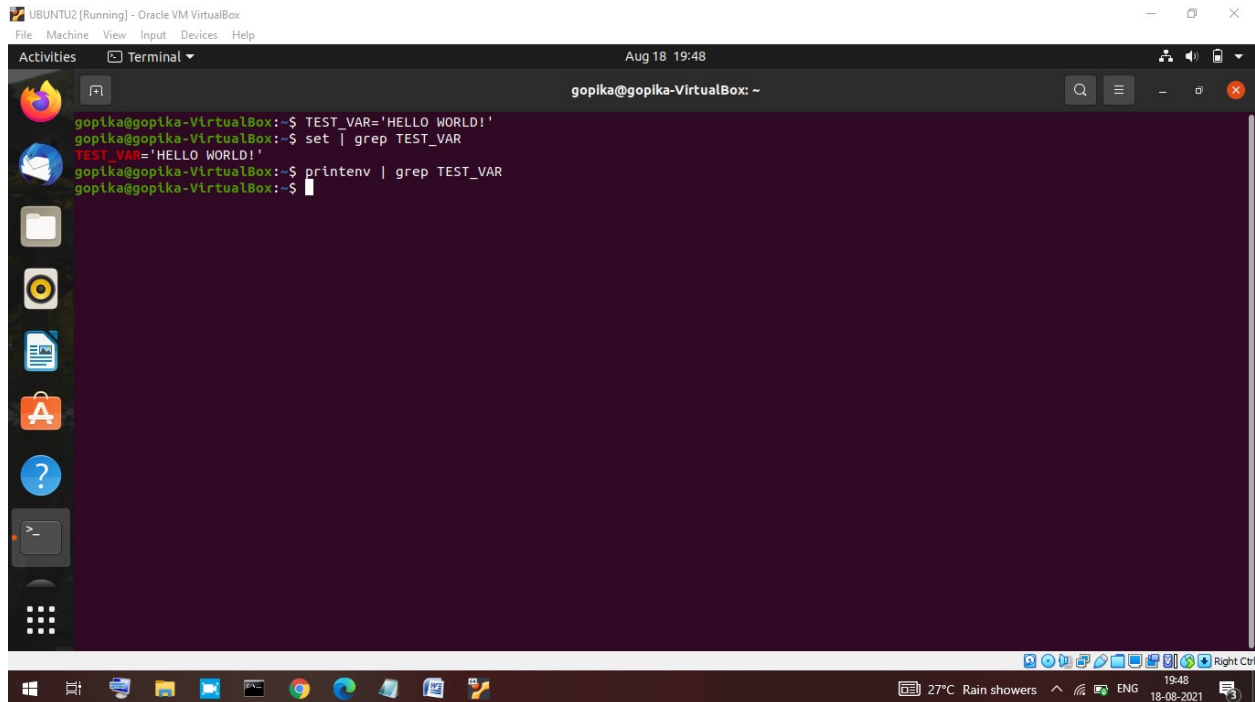
relationship to a corresponding set of environment variables. These shell variables are user, term, home, and path. The value of the environment variable counterpart is initially used to set the shell variable.

Common Environmental and Shell Variables

- SHELL: This describes the shell that will be interpreting any commands you type in. In most cases, this will be bash by default, but other values can be set if you prefer other options.
- TERM: This specifies the type of terminal to emulate when running the shell. Different hardware terminals can be emulated for different operating requirements. You usually won't need to worry about this though.
- USER: The current logged in user.
- PWD: The current working directory.
- OLDPWD: The previous working directory. This is kept by the shell in order to switch back to your previous directory by running `cd -`.
- LS_COLORS: This defines color codes that are used to optionally add colored output to the `ls` command. This is used to distinguish different file types and provide more info to the user at a glance.
- MAIL: The path to the current user's mailbox.
- PATH: A list of directories that the system will check when looking for commands. When a user types in a command, the system will check directories in this order for the executable.
- LANG: The current language and localization settings, including character encoding.

- HOME: The current user's home directory.
- _: The most recent previously executed command

CREATING SHELL AND ENVIRONMENTAL VARIABLES

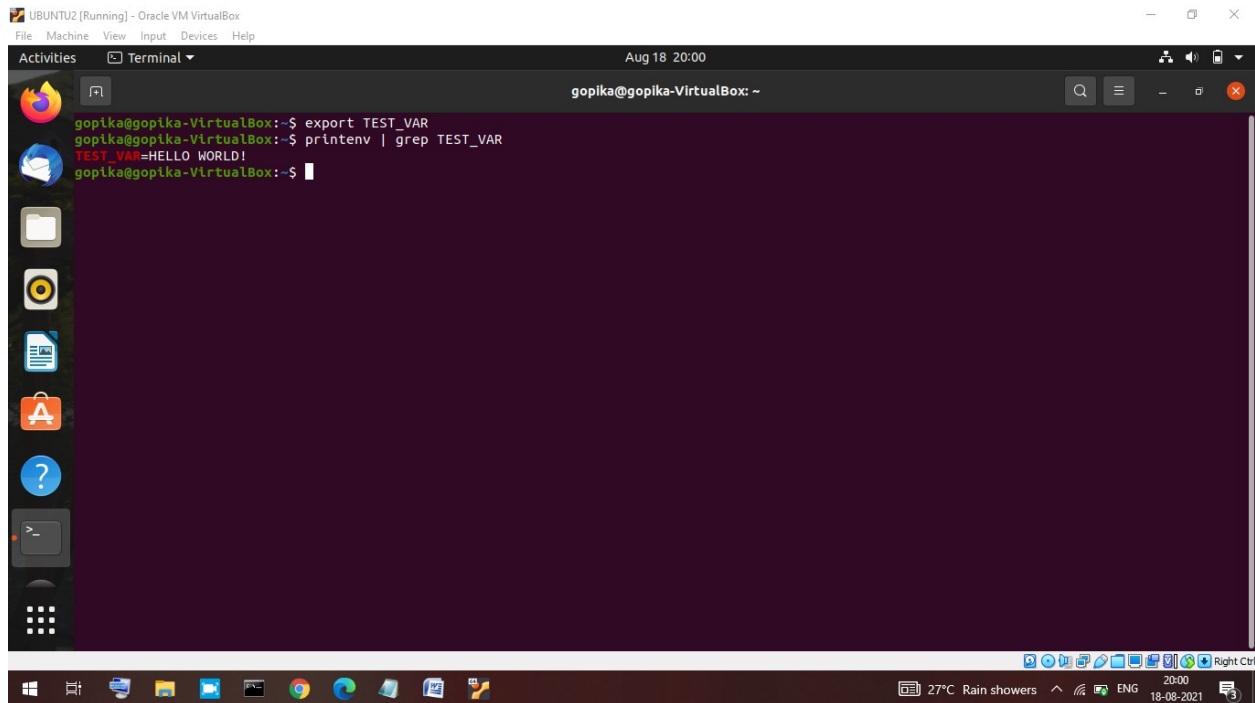


The screenshot shows a terminal window titled "gopika@gopika-VirtualBox: ~" with the following commands and output:

```
gopika@gopika-VirtualBox:~$ TEST_VAR='HELLO WORLD!'  
gopika@gopika-VirtualBox:~$ set | grep TEST_VAR  
TEST_VAR='HELLO WORLD!'  
gopika@gopika-VirtualBox:~$ printenv | grep TEST_VAR  
gopika@gopika-VirtualBox:~$
```

The terminal window is part of a desktop environment with a taskbar at the bottom showing various application icons and system status information (27°C, Rain showers, 19:48, 18-08-2021).

Since we are not setted the environment variable so the “printenv” statement willnot return any output .



```
gopika@gopika-VirtualBox:~$ export TEST_VAR
gopika@gopika-VirtualBox:~$ printenv | grep TEST_VAR
TEST_VAR=HELLO WORLD!
gopika@gopika-VirtualBox:~$
```

VARIABLES

Shell variables provide us the ability to store and manipulate information within a shell program. Variables are used to change the flow of the program and to maintain the state.

Types of Variables

1) Environment variables:

These are the variables that are visible to the child processes of a shell program. These include special environment variables that are set by the shell and are required for the shell to function properly.

Example:

- \$PATH – The set of paths to search for commands.

- \$HOME – The path to the current user’s home folder.

2) Predefined variables:

When running commands, the shell expands wildcards, and then assigns the arguments to these predefined variables or ‘positional variables’.

Example:

- \$0 – The name of the command being executed.
- \$1 ... \$9 – The first to ninth arguments.

3) User-defined variables:

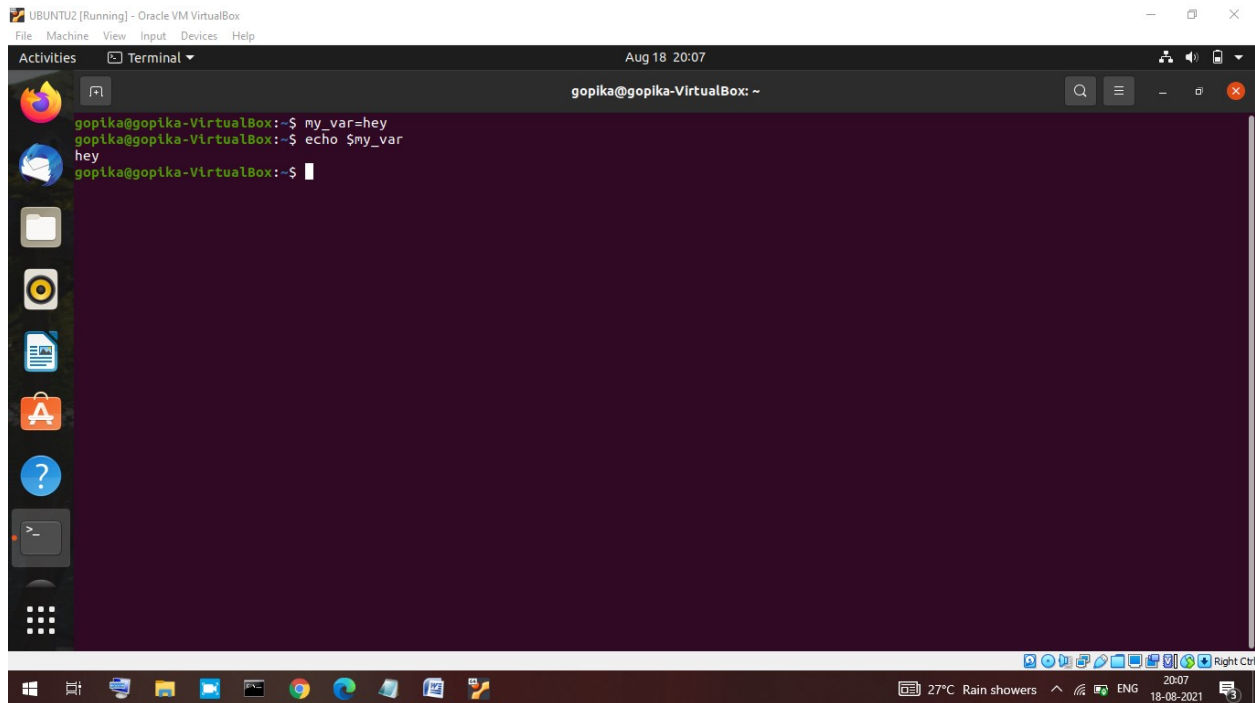
These are the variables that are visible to the current instance of the shell. The ‘export’ command is used to expose local variables to the environment variables.

DEFINING VARIABLE

A variable is defined by simply assigning a value to a name using the ‘=’ operator. A variable name is a series of alphanumeric characters starting with a letter or ‘_’. Variables are all treated as text strings unless the context requires them to be treated as a numeric value.

ACCESSING A VARIABLE

A variable name is de-referenced by simply adding a ‘\$’ prefix to it. The ‘echo’ command is often used to print the value of a variable.



CONTROL STRUCTURES

The shell processes the commands in a script sequentially, one after another in the order they are written in the file. Often, however, we will need to change the way that commands are processed. You may want to choose to run one command or another, depending on the circumstances; or you may want to run a command more than once. To alter the normal sequential execution of commands, the shell offers a variety of control structure.

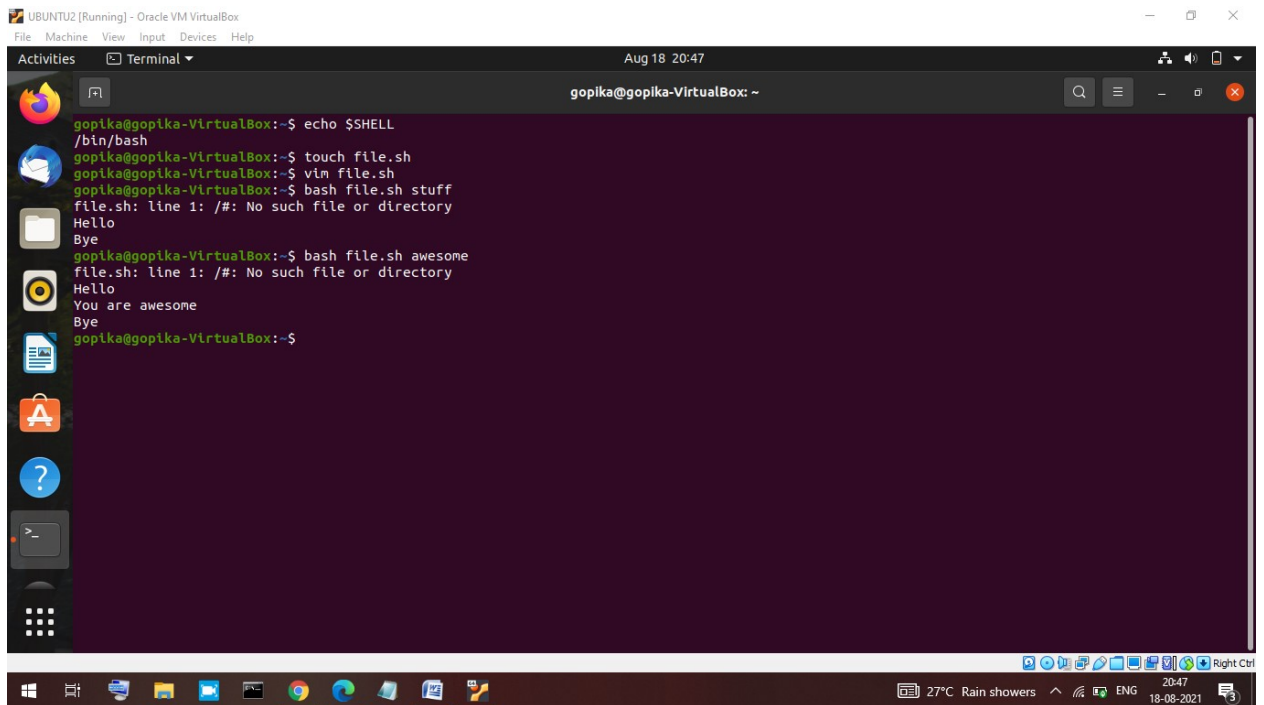
If statement

- The if statement lets us choose whether to run a particular command (or group of commands), depending on some condition.

- The simplest version of this structure has the general form

if conditional expression then command(s) fi

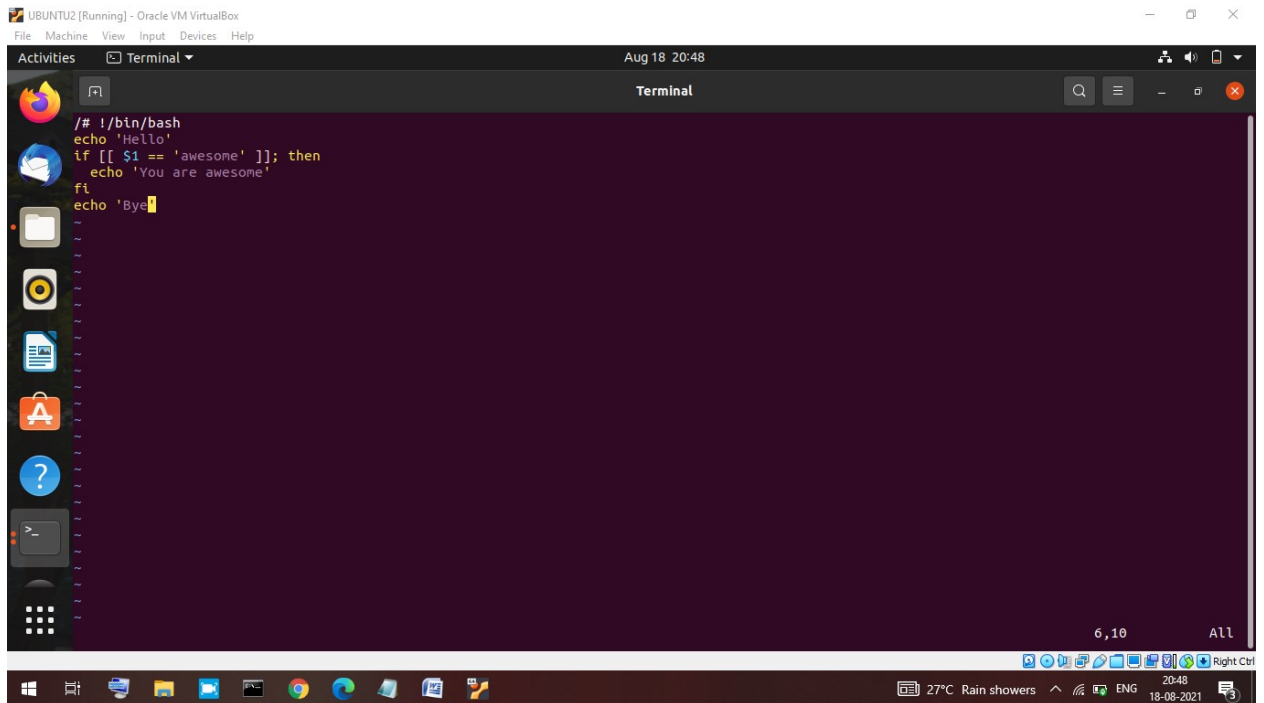
- When the shell encounters a structure such as this, it first checks to see whether the conditional expression is true.
- If so, the shell runs any commands that it finds between the then and the fi (which is just if spelled backwards).
- If the conditional expression is not true, the shell skips the commands between then and fi



The screenshot shows a terminal window titled "gopika@gopika-VirtualBox: ~" with a dark purple background. The terminal displays the following commands and their outputs:

```
gopika@gopika-VirtualBox:~$ echo $SHELL
/bin/bash
gopika@gopika-VirtualBox:~$ touch file.sh
gopika@gopika-VirtualBox:~$ vim file.sh
gopika@gopika-VirtualBox:~$ bash file.sh stuff
file.sh: line 1: /#: No such file or directory
Hello
Bye
gopika@gopika-VirtualBox:~$ bash file.sh awesome
file.sh: line 1: /#: No such file or directory
Hello
You are awesome
Bye
gopika@gopika-VirtualBox:~$
```

The terminal window is part of a larger application window titled "UBUNTU2 [Running] - Oracle VM VirtualBox". The window has a menu bar with "File", "Machine", "View", "Input", "Devices", and "Help". The status bar at the bottom shows the date and time as "Aug 18 20:47".



The screenshot shows a terminal window titled 'Terminal' within an 'Ubuntu2 [Running] - Oracle VM VirtualBox' environment. The terminal has a dark purple background and displays the following shell script:

```
#!/bin/bash
echo 'Hello'
if [[ $1 == 'awesome' ]]; then
    echo 'You are awesome'
fi
echo 'Bye'
```

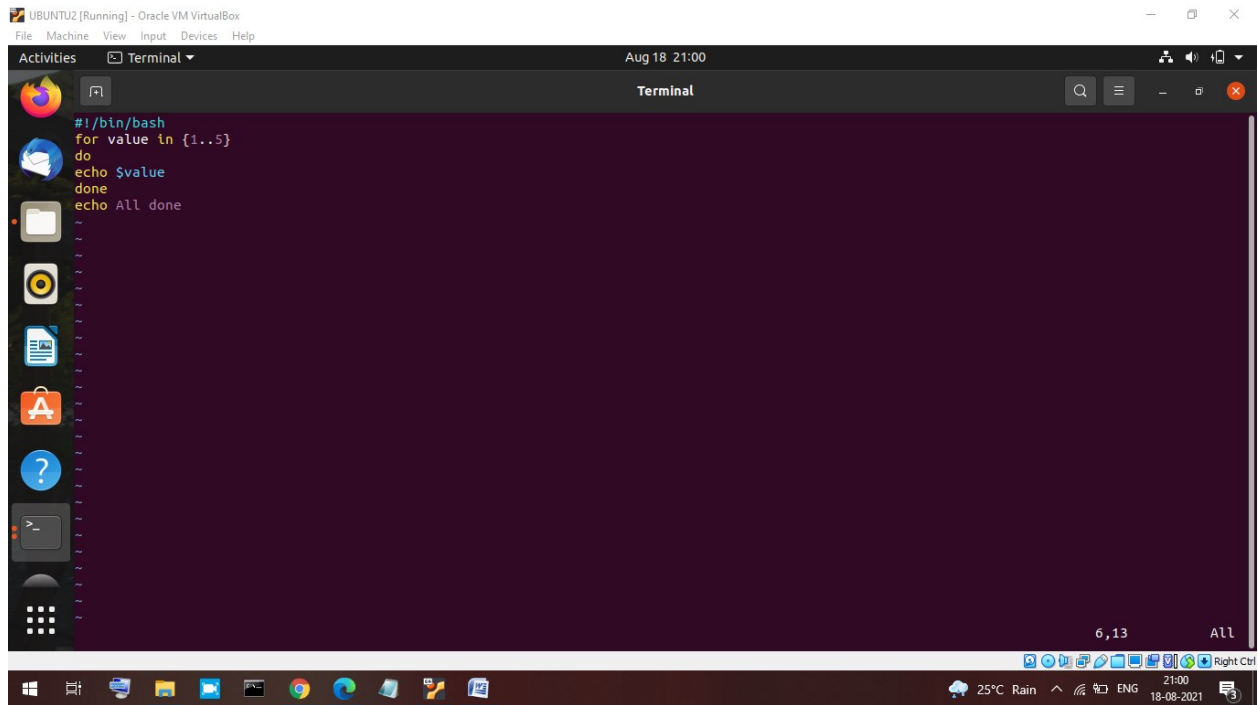
The terminal window includes a menu bar with 'File', 'Machine', 'View', 'Input', 'Devices', and 'Help'. The top status bar shows 'Aug 18 20:48'. The bottom status bar displays system information: '6, 10', 'All', '27°C Rain showers', 'ENG', '20:48', and '18-08-2021'.

- **for loop**

Sometimes we want to run a command (or group of commands) over and over. This is called iteration, repetition, or looping. The most commonly used shell repetition structure is the for loop, which has the general for:

Syntax:-

```
for var in <list>
do
<commands>
done
```



UBUNTU2 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

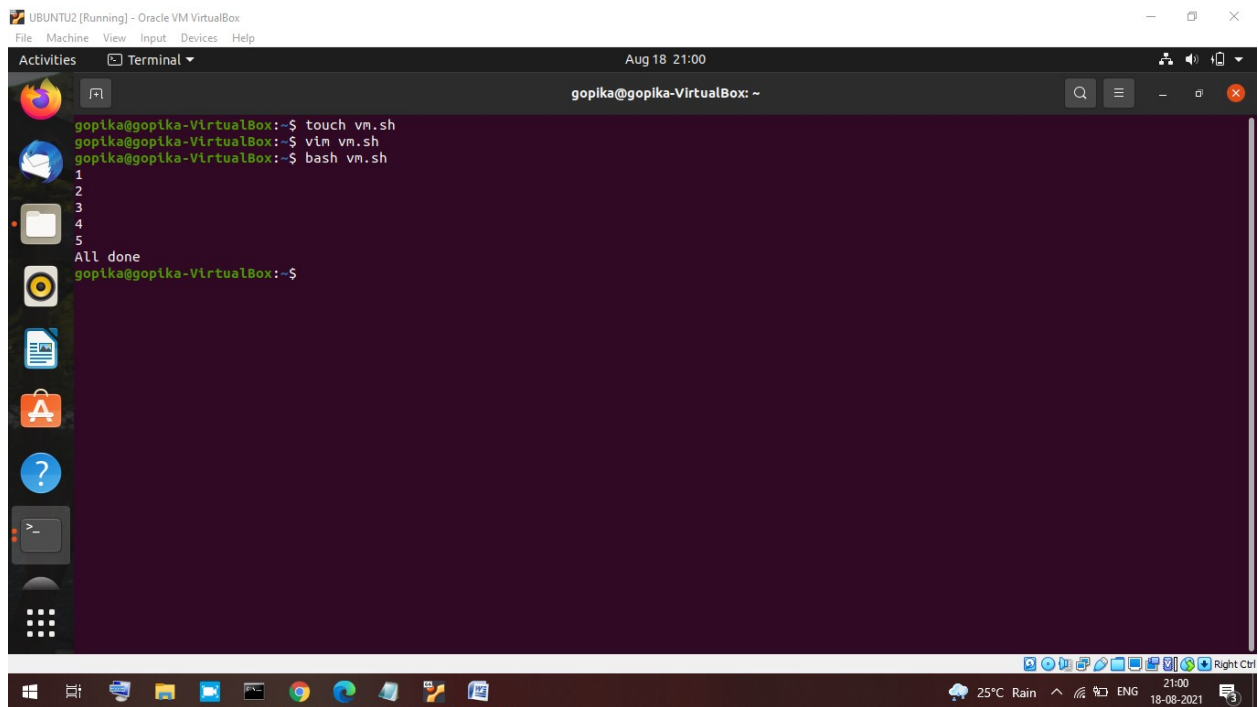
Activities Terminal

Aug 18 21:00

```
#!/bin/bash
for value in {1..5}
do
echo $value
done
echo All done
```

6,13 All

25°C Rain 21:00 18-08-2021



UBUNTU2 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Terminal

Aug 18 21:00

gopika@gopika-VirtualBox: ~

```
gopika@gopika-VirtualBox:~$ touch vm.sh
gopika@gopika-VirtualBox:~$ vim vm.sh
gopika@gopika-VirtualBox:~$ bash vm.sh
1
2
3
4
5
All done
gopika@gopika-VirtualBox:~$
```

25°C Rain 21:00 18-08-2021

While loop

The general form of the while loop is:

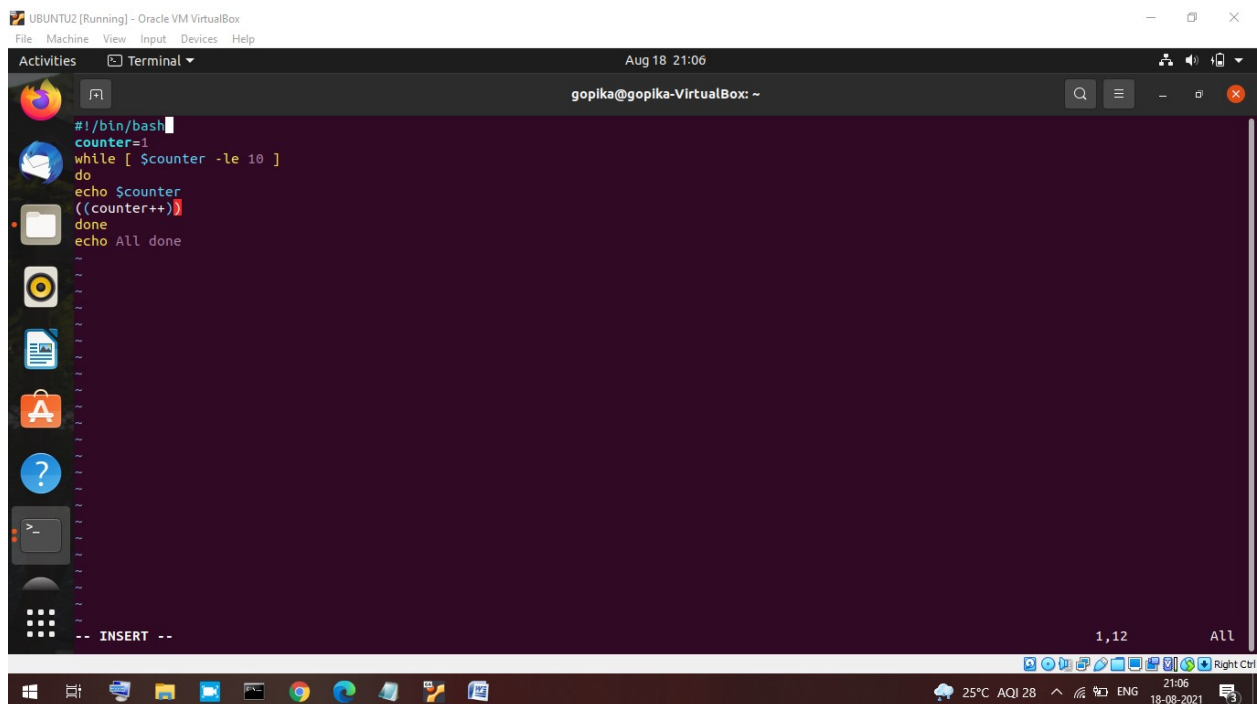
while [<condition>]

do

<command(s) >

done

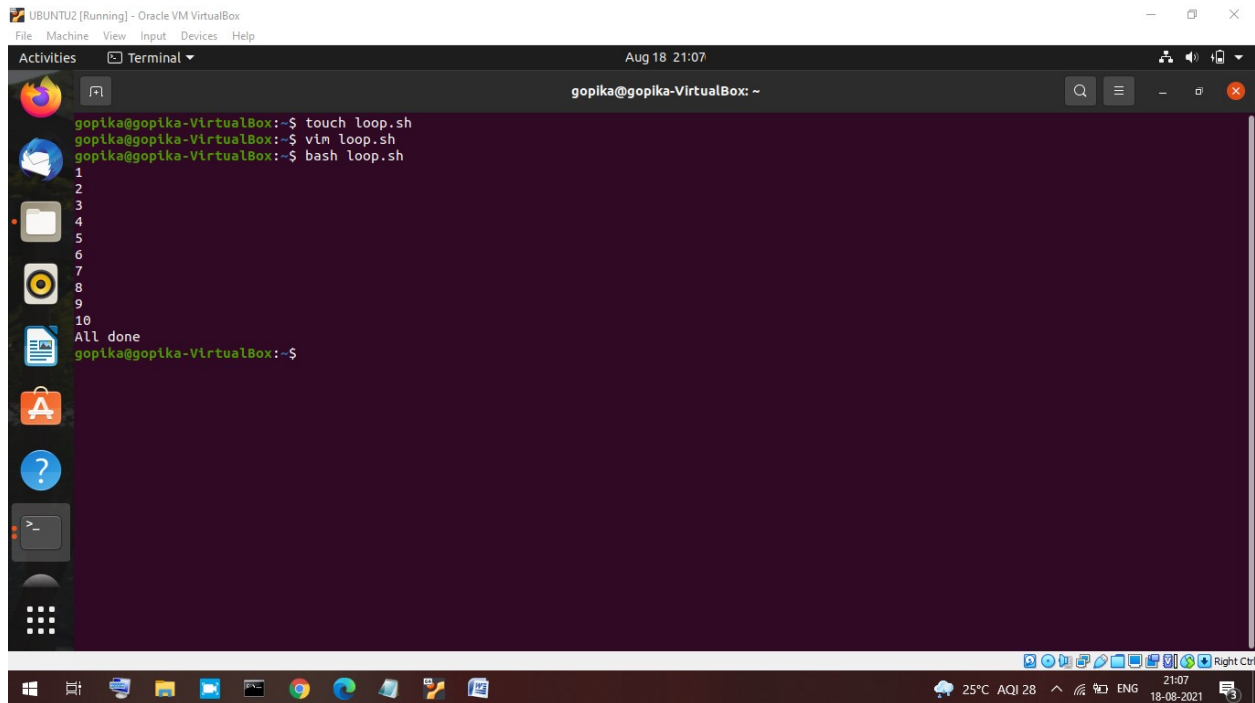
As long as the condition is true, the commands between the do and the done are executed.



The screenshot shows a terminal window titled "gopika@gopika-VirtualBox: ~" with a dark purple background. The terminal displays the following code:

```
#!/bin/bash
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo All done
```

The terminal output shows the execution of the script, with the counter increasing from 1 to 10 and the message "All done" printed at the end. The terminal window is part of a larger application window titled "UBUNTU2 [Running] - Oracle VM VirtualBox". The bottom of the image shows a Windows taskbar with various icons and a system tray displaying the date and time.



The screenshot shows a terminal window titled 'gopika@gopika-VirtualBox: ~' with a dark purple background. The terminal displays the following commands and output:

```
gopika@gopika-VirtualBox:~$ touch loop.sh
gopika@gopika-VirtualBox:~$ vim loop.sh
gopika@gopika-VirtualBox:~$ bash loop.sh
1
2
3
4
5
6
7
8
9
10
All done
gopika@gopika-VirtualBox:~$
```

The terminal window is part of a larger application window titled 'UBUNTU2 [Running] - Oracle VM VirtualBox'. The window has a menu bar with 'File', 'Machine', 'View', 'Input', 'Devices', and 'Help'. The status bar at the bottom shows the date 'Aug 18 21:07' and the time '21:07'.

‘alias’ in shell scripting

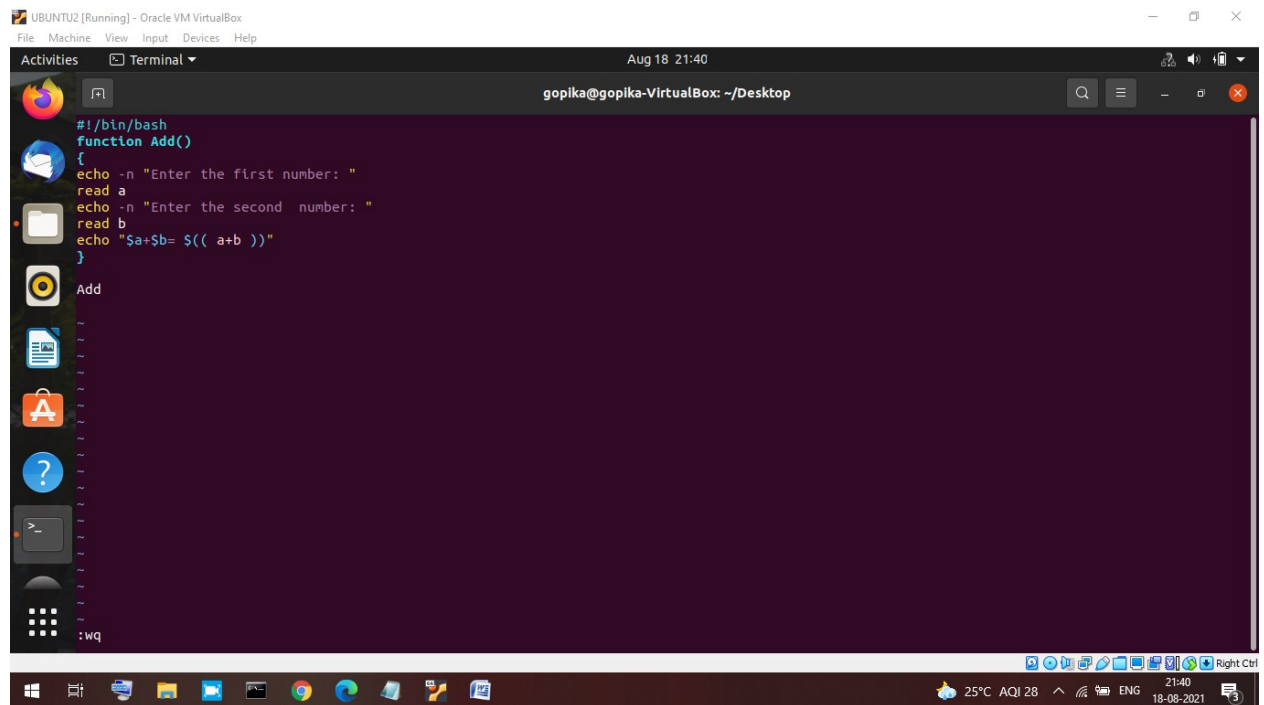
A shell alias is a shortcut to reference a command. It can be used to avoid typing long commands or as a means to correct incorrect input. For common patterns it can reduce keystrokes and improve efficiency. A simple example is setting default options on commands to avoid having to type them each time a command is run.

Syntax:

```
alias [alias_name]="[command]"
```

Shell Scripting Example:-

Adding two numbers

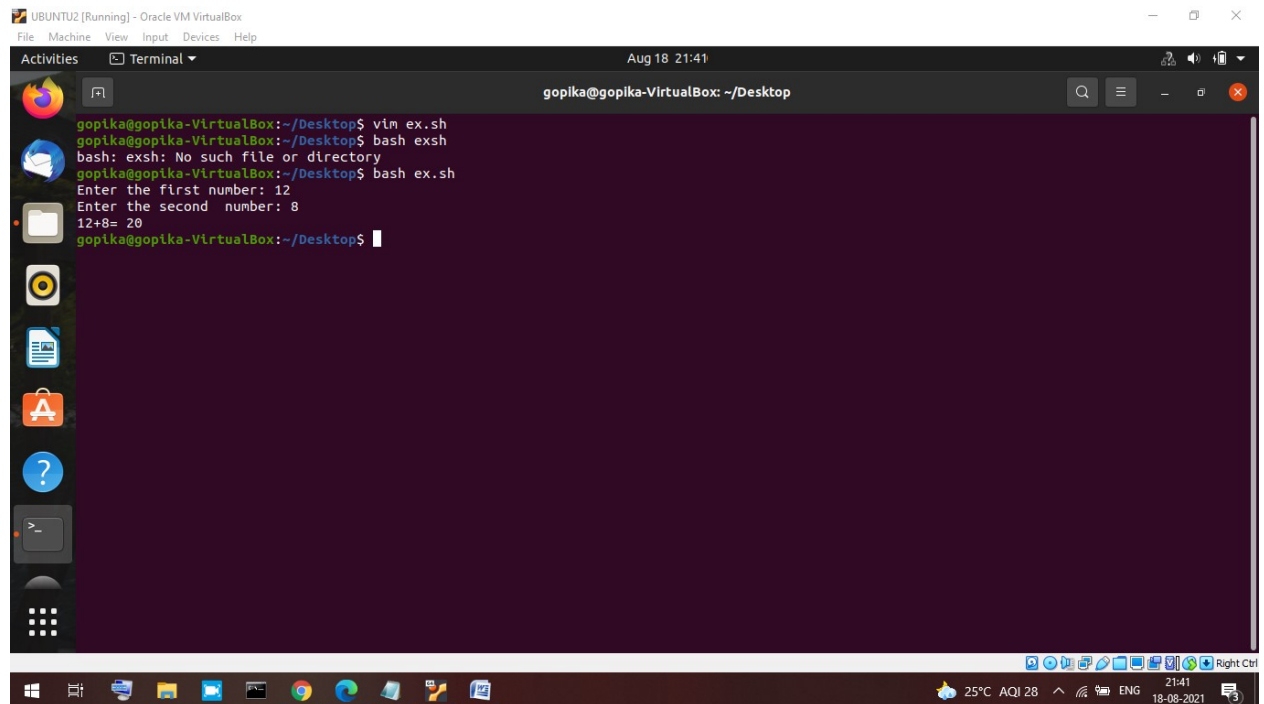


The screenshot shows a terminal window titled "gopika@gopika-VirtualBox: ~/Desktop". The terminal content is as follows:

```
#!/bin/bash
function Add()
{
echo -n "Enter the first number: "
read a
echo -n "Enter the second number: "
read b
echo "$a+$b= $(( a+b ))"
}

Add
```

The terminal is running in a Ubuntu 20.04 LTS environment. The desktop background is dark purple. The taskbar at the bottom shows various application icons and system status information including temperature (25°C), AQI (28), and time (21:40 on 18-08-2021).



The screenshot shows the same terminal window after the 'Add' function has been executed. The terminal content is as follows:

```
gopika@gopika-VirtualBox:~/Desktop$ vim ex.sh
gopika@gopika-VirtualBox:~/Desktop$ bash exsh
bash: exsh: No such file or directory
gopika@gopika-VirtualBox:~/Desktop$ bash ex.sh
Enter the first number: 12
Enter the second number: 8
12+8= 20
gopika@gopika-VirtualBox:~/Desktop$
```

The terminal shows the user editing 'ex.sh' with 'vim', then attempting to run 'exsh' (which fails with 'No such file or directory'), and finally running 'ex.sh'. The script prompts for two numbers, 12 and 8, and outputs '12+8= 20'.

Example 2:

```
UBUNTU2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Aug 18 22:00
gopika@gopika-VirtualBox: ~
/ #!/bin/bash
echo "Hey!what is your name?"
read name
echo "welcome $name"
```

```
UBUNTU2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Aug 18 22:01
gopika@gopika-VirtualBox: ~
gopika@gopika-VirtualBox:~$ touch wel.sh
gopika@gopika-VirtualBox:~$ vim wel.sh
gopika@gopika-VirtualBox:~$ bash wel.sh
wel.sh: line 1: /: Is a directory
Hey!what is your name?
gopika
welcome
gopika@gopika-VirtualBox:~$ vim wel.sh
gopika@gopika-VirtualBox:~$ bash wel.sh
wel.sh: line 1: /: Is a directory
Hey!what is your name?
gopika
welcome
gopika@gopika-VirtualBox:~$ vim wel.sh
gopika@gopika-VirtualBox:~$ bash wel.sh
wel.sh: line 1: /: Is a directory
Hey!what is your name?
gopika
welcome gopika
gopika@gopika-VirtualBox:~$
```