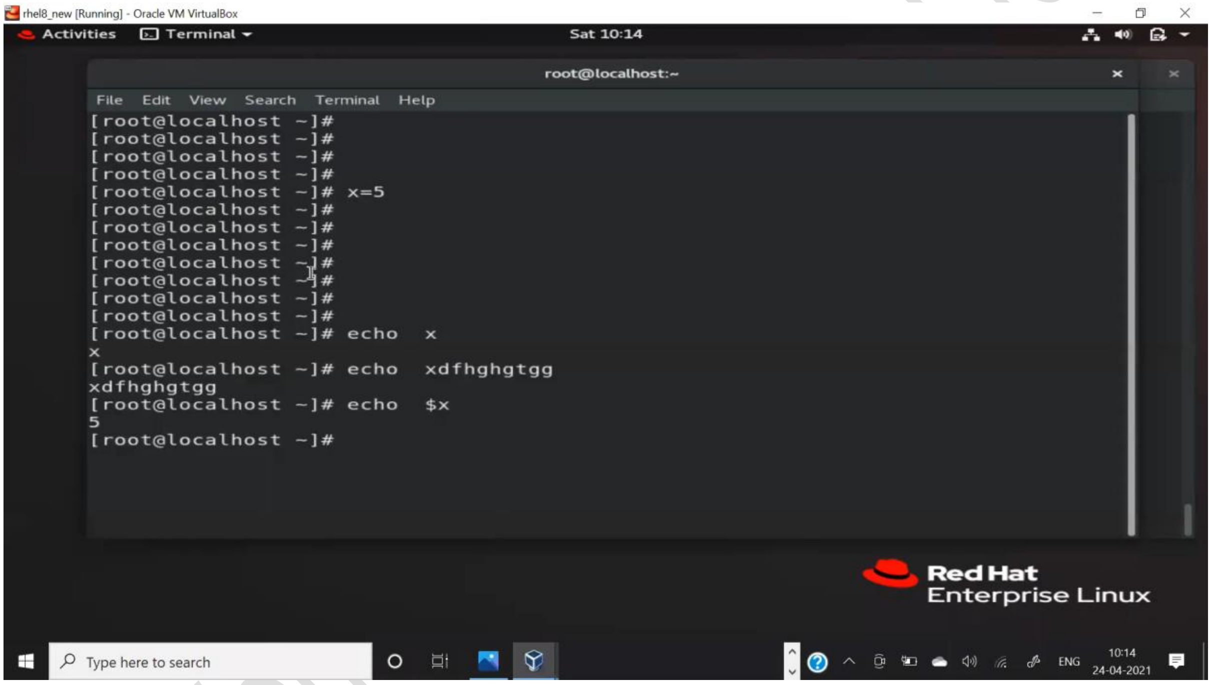
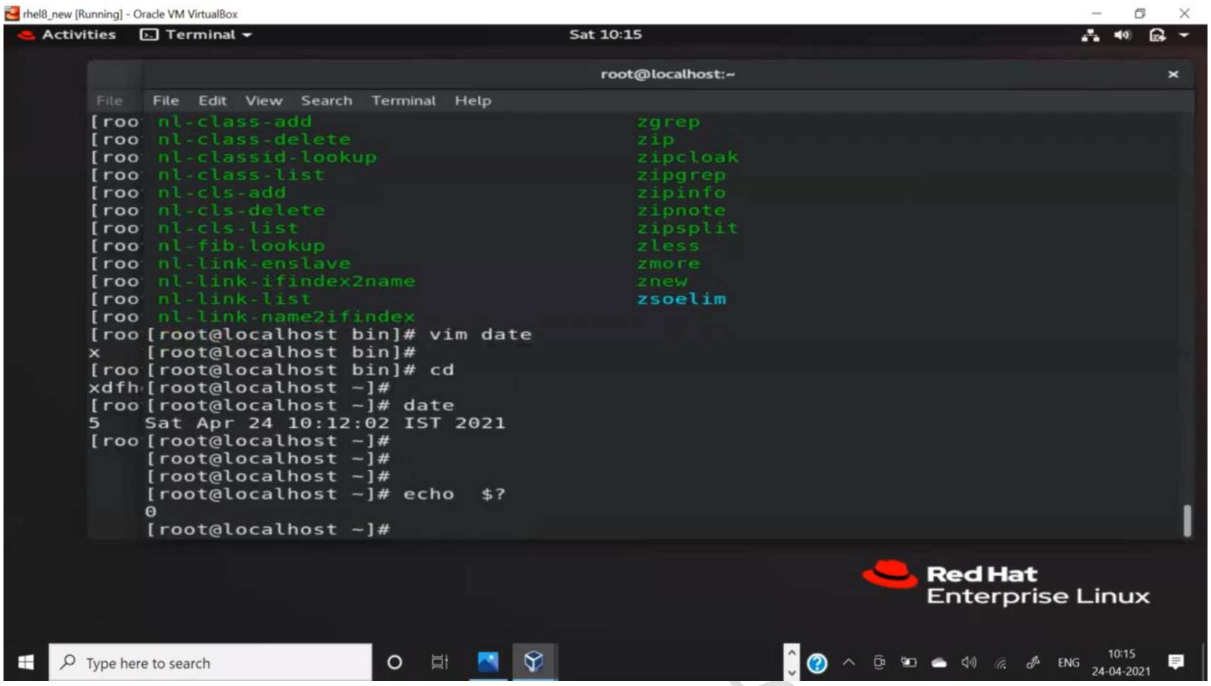
Shell and Shell Scripting Session No.1.2

- A variable in Linux is like a box that stores some value, like x=5, here x is a variable that stores value 5. To know the value of x there is a symbol "$" that prints the value of a variable otherwise echo x will print x on the screen



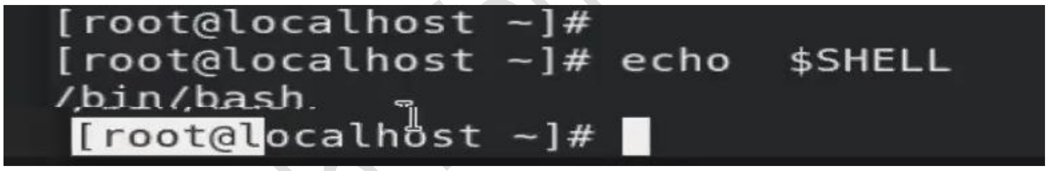
```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# x=5
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# echo  x
x
[root@localhost ~]# echo  xdfhghgtgg
xdfhghgtgg
[root@localhost ~]# echo  $x
5
[root@localhost ~]#
```

- And as this variable is created by us, so it is also called user-defined variables.
- Whenever you run any command, it returns the output but along with this, it also returns the status code, if the command is successful then 0, and if the command is unsuccessful then 1.
- And we can get the status code of the last command we ran with the help of a variable "?"



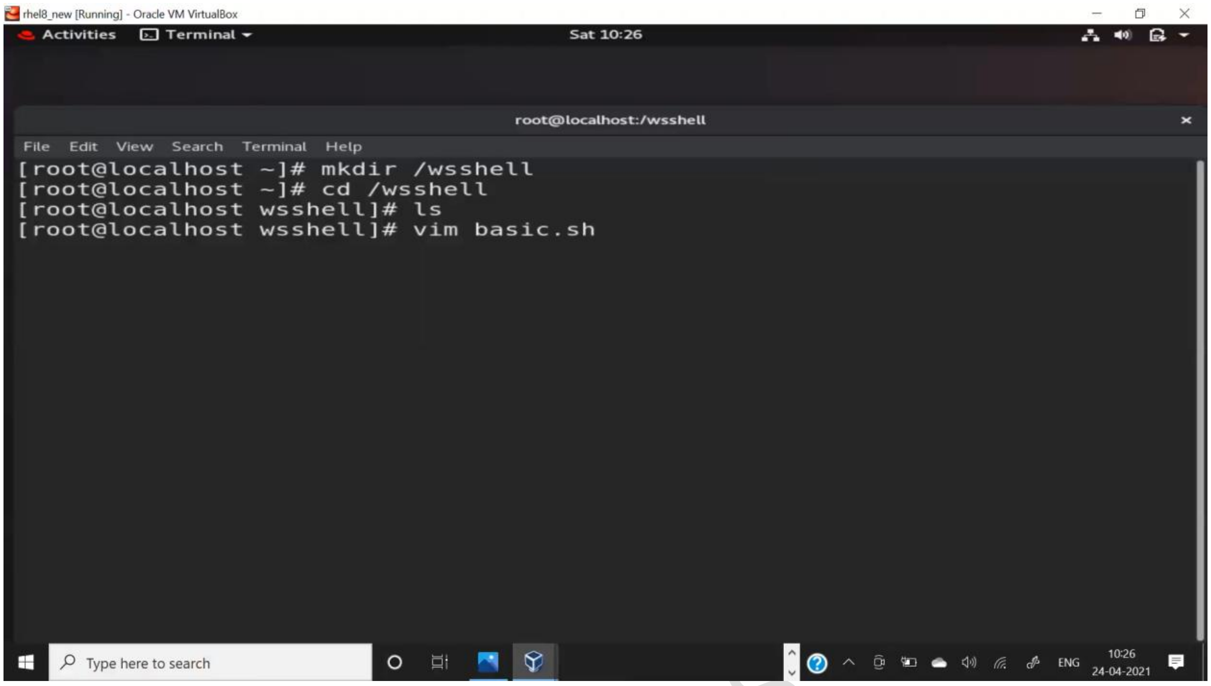
```
[roo nl-class-add                         zgrep
[roo nl-class-delete                      zip
[roo nl-classid-lookup                    zipcloak
[roo nl-class-list                        zipgrep
[roo nl-cls-add                           zipinfo
[roo nl-cls-delete                        zipnote
[roo nl-cls-list                          zipsplit
[roo nl-fib-lookup                        zless
[roo nl-link-enslave                      zmore
[roo nl-link-ifindex2name                 znew
[roo nl-link-list                         zsoelim
[roo nl-link-name2ifindex
[roo [root@localhost bin]# vim date
x    [root@localhost bin]#
[roo [root@localhost bin]# cd
xdfh [root@localhost ~]#
[roo [root@localhost ~]# date
5    Sat Apr 24 10:12:02 IST 2021
[roo [root@localhost ~]#
     [root@localhost ~]#
     [root@localhost ~]#
     [root@localhost ~]# echo  $?
     0
     [root@localhost ~]#
```

- The use case for this is like in the scripting world you don't run commands one by one, everything is end-to-end automated. So if any one command fails in that case the entire script fails, here the status code helps us. We can set some if-or-else conditions for this.
- There is also one variable SHELL that tells us our current shell



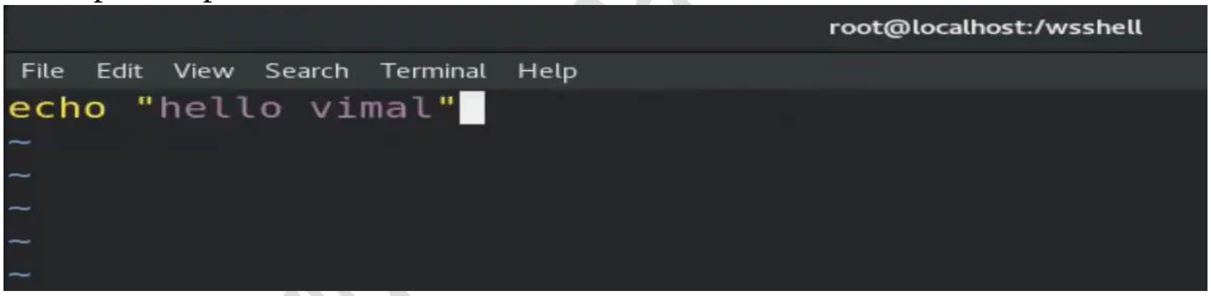
```
[root@localhost ~]#
[root@localhost ~]# echo  $SHELL
/bin/bash
 [root@localhost ~]#
```

- No matter how many terminals you open, each one has its shell, so if you create one variable in one terminal, that variable is not present in another terminal.
- We have created one working directory for this program and now own wards we will do everything in this directory only



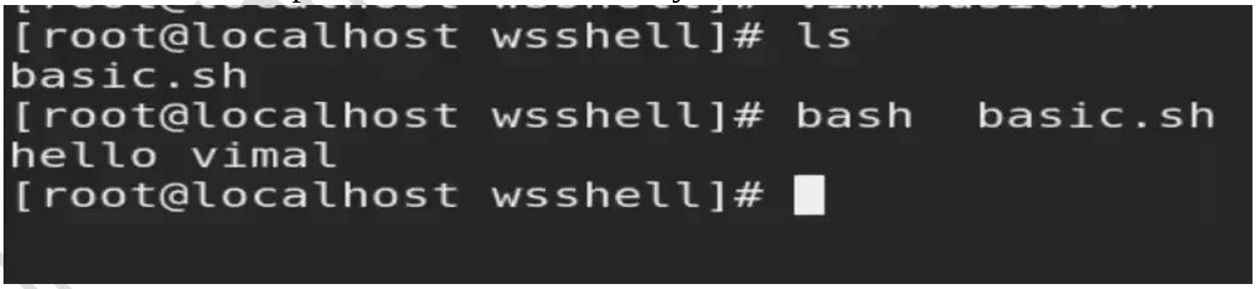
```
[root@localhost ~]# mkdir /wsshell
[root@localhost ~]# cd /wsshell
[root@localhost wsshell]# ls
[root@localhost wsshell]# vim basic.sh
```

- It is good practice to give the extension ".sh" to clarify that it is a shell script file.
- A simple script we have created here in this file



```
echo "hello vimal"
~
~
~
~
~
```

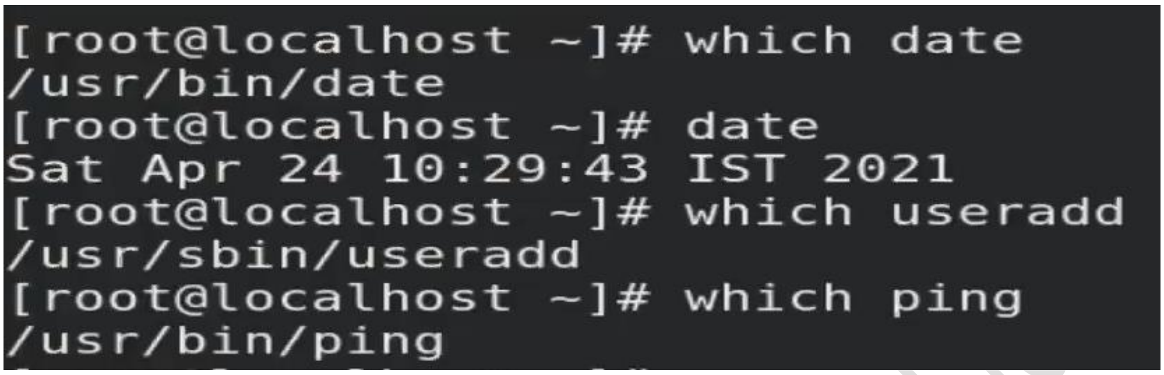
- And to run this script we can use the bash keyword.



```
[root@localhost wsshell]# ls
basic.sh
[root@localhost wsshell]# bash  basic.sh
hello vimal
[root@localhost wsshell]# 
```

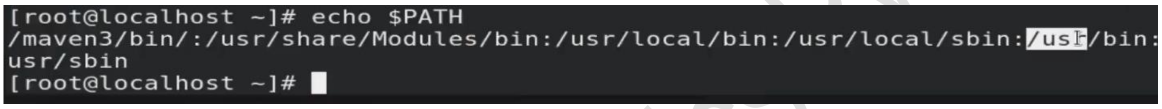
- Now this is a script but I want to convert it into a command. Although a command is also a program or script the advantage that a command enjoys over here is it can be run from anywhere but a script can only be run in its present directory or can be run by specifying its absolute path.
- So I want here that my script can also be run from anywhere without any limitations just like a command.

- So we will first see that the commands like date or cal or useradd store in which location.



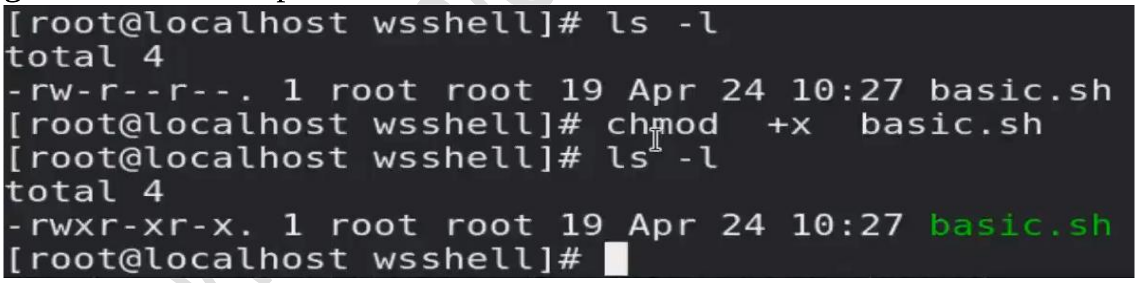
```
[root@localhost ~]# which date
/usr/bin/date
[root@localhost ~]# date
Sat Apr 24 10:29:43 IST 2021
[root@localhost ~]# which useradd
/usr/sbin/useradd
[root@localhost ~]# which ping
/usr/bin/ping
```

- That means there are some default paths already set for storing these programs so that they can be run from anywhere.
- And to know that locations we have another variable available PATH.



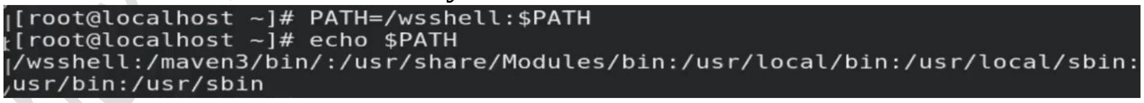
```
[root@localhost ~]# echo $PATH
/maven3/bin/:/usr/share/Modules/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:
usr/sbin
[root@localhost ~]#
```

- That means anything present inside these directories will become a command that can be run from anywhere.
- So we will also add our directory in this variable. Also, we want this file to be executed so we have to give it executable power or permission, as we don't want to use bash command to run this file that's why we have to give it executable permission.



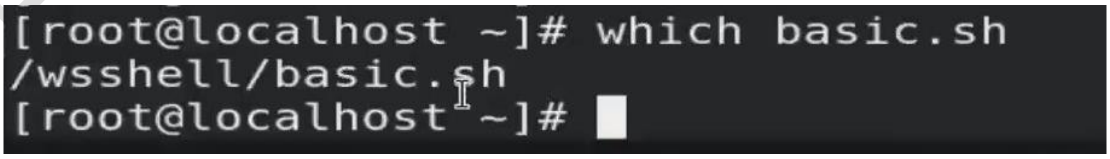
```
[root@localhost wsshell]# ls -l
total 4
-rw-r--r--. 1 root root 19 Apr 24 10:27 basic.sh
[root@localhost wsshell]# chmod  +x  basic.sh
[root@localhost wsshell]# ls -l
total 4
-rwxr-xr-x. 1 root root 19 Apr 24 10:27 basic.sh
[root@localhost wsshell]#
```

- Command to add our directory in the PATH variable.



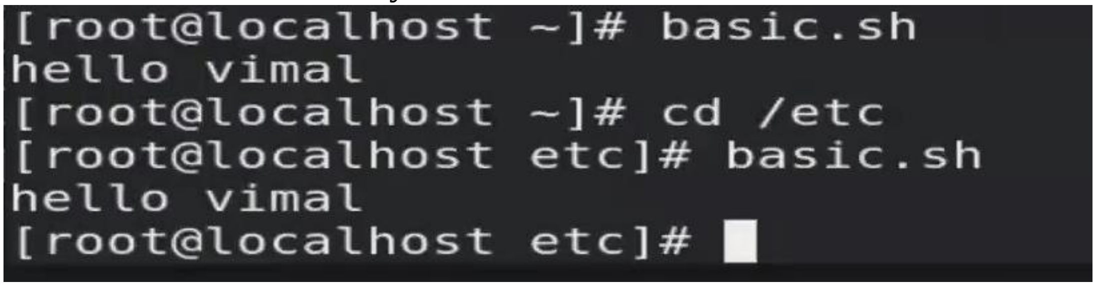
```
[root@localhost ~]# PATH=/wsshell:$PATH
[root@localhost ~]# echo $PATH
/wsshell:/maven3/bin/:/usr/share/Modules/bin:/usr/local/bin:/usr/local/sbin:
usr/bin:/usr/sbin
```

- Now if you try to search for the basic. sh from anywhere you will find its location.



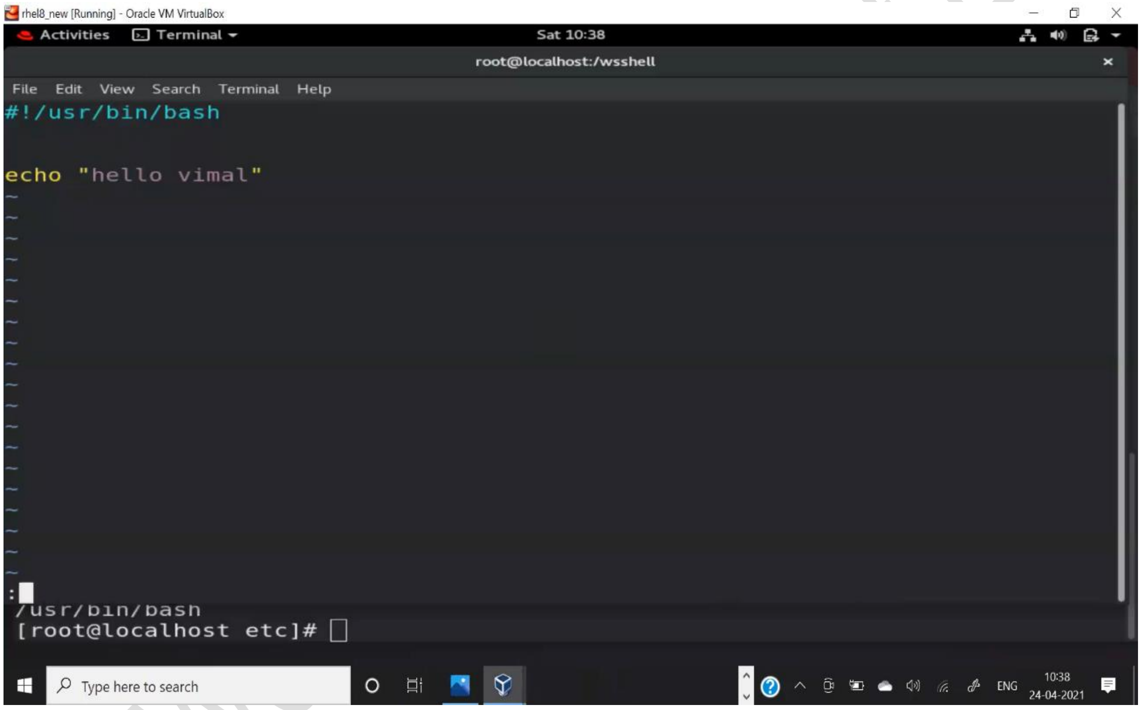
```
[root@localhost ~]# which basic.sh
/wsshell/basic.sh
[root@localhost ~]#
```

- And it can be run from anywhere.



```
[root@localhost ~]# basic.sh
hello vimal
[root@localhost ~]# cd /etc
[root@localhost etc]# basic.sh
hello vimal
[root@localhost etc]# 
```

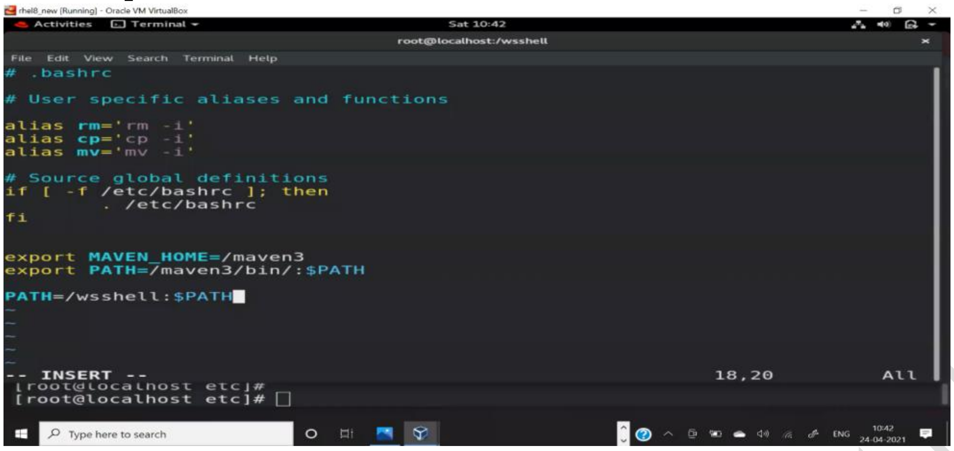
- But here, in someone's system, it could be possible that the default shell they have is sh or c-sh, so to avoid any confusion or error that could occur, we will add the hashbang symbol (#!) on the top of the script and write the name of the shell



```
#!/usr/bin/bash


echo "hello vimal"
~
~
~
:
/usr/bin/bash
[root@localhost etc]# 
```

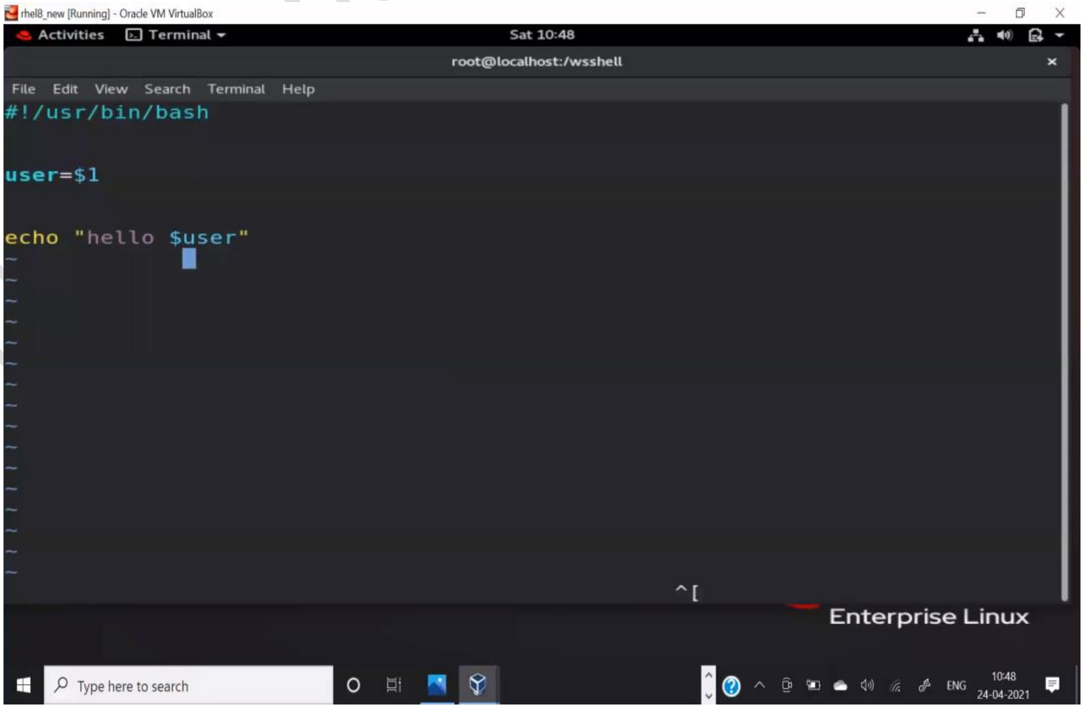
- And if you run this command from some other terminal, it will not run because the changes we have made are in this particular session, so every time you open a new terminal, you have to, again and again, add the path and then do all the changes that we did before.
- For this reason bash comes up with its configuration file located at **"/root/.bashrc"** and we will add our command in this file and we want to

make it permanent



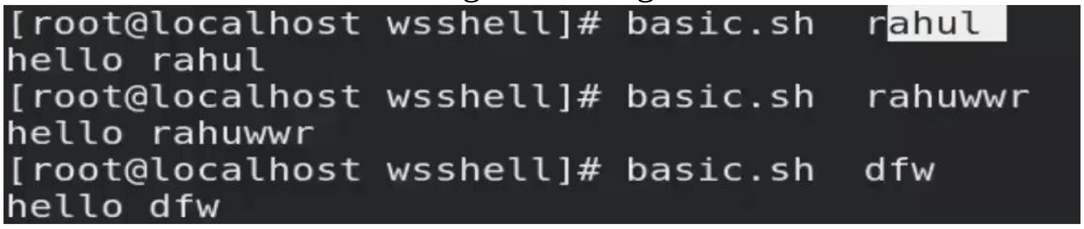
```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

export MAVEN_HOME=/maven3
export PATH=/maven3/bin/:$PATH

PATH=/wsshell:$PATH
~
~
~
~
-- INSERT --                                                  18,20         All
[root@localhost etc]#
[root@localhost etc]#
```

- So what this file will do, is whenever you open a new terminal, bash will run this file first. So all the changes you want to make permanent can be done easily.
- Now I want our script to become dynamic means that I want to run "hello" as it is but the name after it should be dynamic.
- So for this first understand what is the correct syntax of a command. While running a command first we write a command and then we write an option and then an argument. Like in the command "ls -l file1", here ls is the command, -l is the option and file1 is the argument.
- Keeping options aside, we will first focus on the argument that every argument we pass in the command also stores in some variable and that is "1", the first argument is stored in 1 and the second argument stores in 2, and so on. Also the command name stores in 0.
- So we have to make changes in our script for this



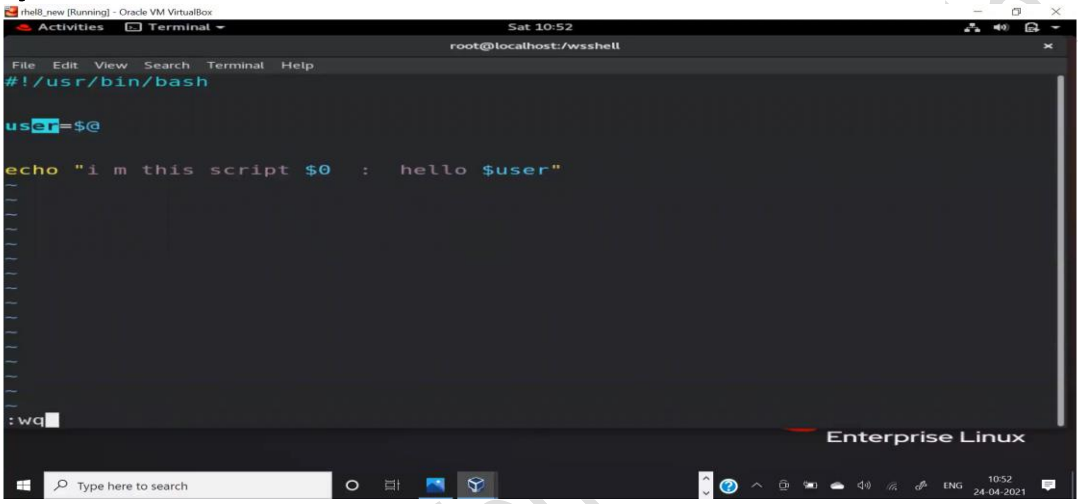
```
#!/usr/bin/bash


user=$1


echo "hello $user"
~
~
~
```

- And now onwards whatever argument we give it to return the result for us



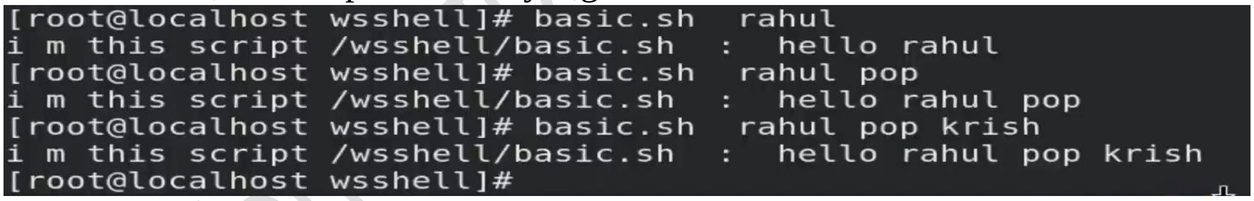
```
[root@localhost wsshell]# basic.sh  rahul
hello rahul
[root@localhost wsshell]# basic.sh  rahuwwr
hello rahuwwr
[root@localhost wsshell]# basic.sh  dfw
hello dfw
```

- But this is a very static script means it allows only 1 argument, so if the use case is like I want to give multiple arguments, there we use the “@” symbol in the code



```
#!/usr/bin/bash

user=$@

echo "i m this script $0  :  hello $user"
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

- Now onwards, we can pass as many arguments as we want.



```
[root@localhost wsshell]# basic.sh  rahul
i m this script /wsshell/basic.sh  :  hello rahul
[root@localhost wsshell]# basic.sh  rahul pop
i m this script /wsshell/basic.sh  :  hello rahul pop
[root@localhost wsshell]# basic.sh  rahul pop krish
i m this script /wsshell/basic.sh  :  hello rahul pop krish
[root@localhost wsshell]#
```