# OS Lab Report – Week 2

PES1201800366

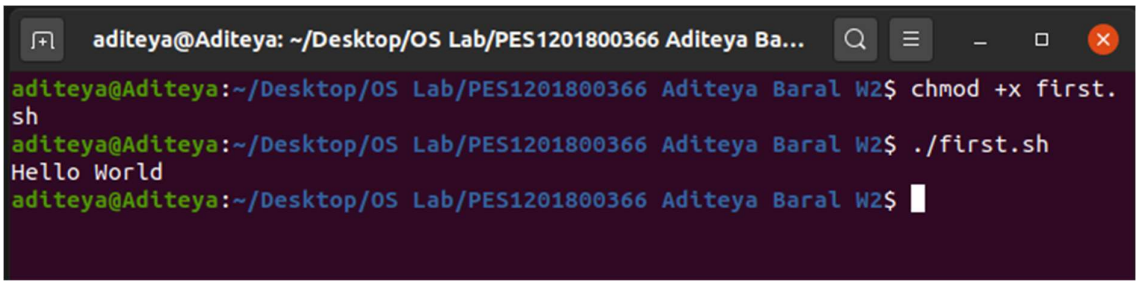
Aditeya Baral

## 1. First.sh

### 1.1 Code

```
1. #!/bin/sh
2. # This is a comment!
3. echo Hello World        # This is a comment, too!
```

### 1.2 Output

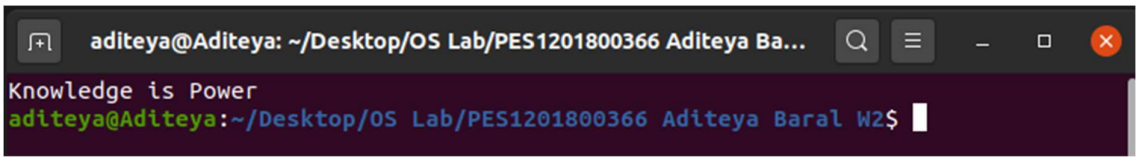



## 2. Second.sh

### 2.1 Code

```
1. #!/bin/sh
2. clear
3. echo "Knowledge is Power
```

### 2.2 Output

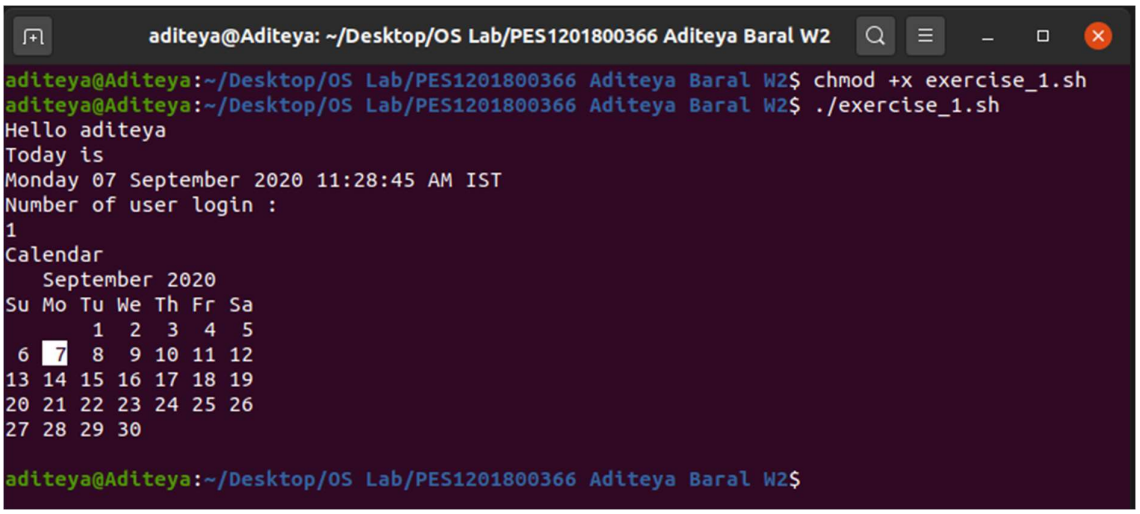


# 3. Exercise 1

## 3.1 Code

```
1. #!/bin/sh
2. # Script to print user information who currently login, current date
   & time
3. # Enter the following commands in a file
4. echo "Hello $USER"
5. echo "Today is ";date
6. echo "Number of user login : " ; who | wc -l
7. echo "Calendar"
8. cal
9. exit 0
```

## 3.2 Output




## 3.3 Reason behind `exit 0`

Every Linux or Unix command executed by the shell script or user has an exit status which is an integer number. `exit 0` status means the command was successful without any errors. A non-zero (1-255 values) exit status means the command was a failure.

## 3.4 Significance of `$`

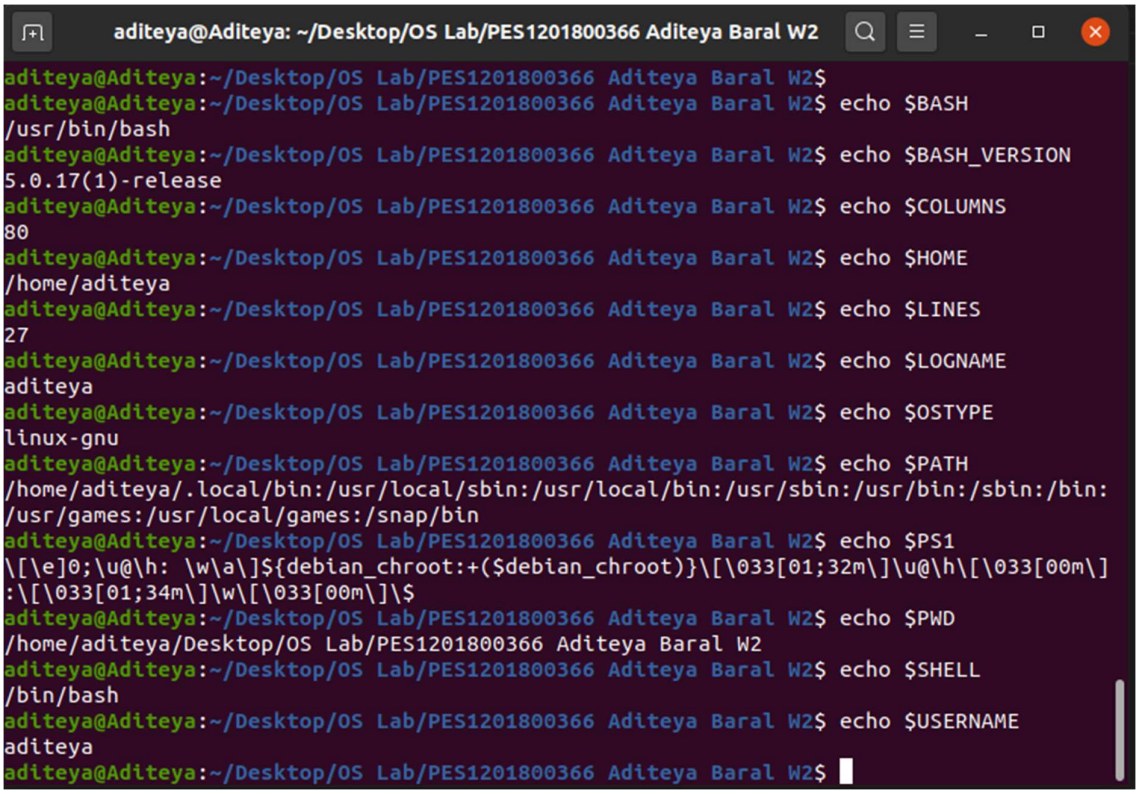
In a shell script, the `$` symbol is used to access the value stored in an identifier or variable. If the `$` symbol is not used as a prefix before the variable name, the shell will just display the name of a variable since the shell will read it as an ordinary string to be displayed.

# 4. System Variables

### 4.1 Code

```
1. #!/bin/sh
2. echo $BASH
3. echo $BASH_VERSION
4. echo $COLUMNS
5. echo $HOME
6. echo $LINES
7. echo $LOGNAME
8. echo $OSTYPE
9.  echo $PATH
10.  echo $PS1
11.  echo $PWD
12.  echo $SHELL
13.  echo $USERNAME
```

### 4.2 Output



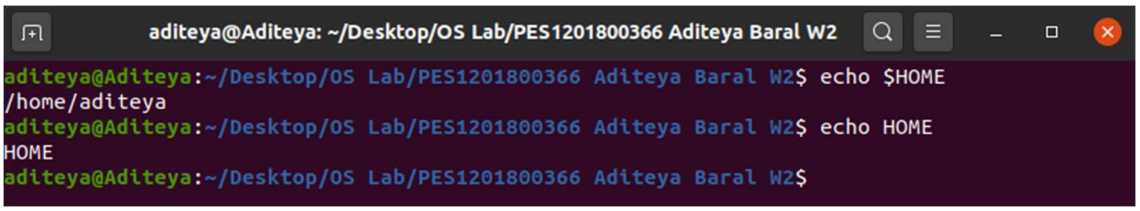

## 5. Exercise 2

### 5.1 Code

```
1. echo $HOME
```

```
1. echo HOME
```

### 5.2 Output




### 5.3 Explanation

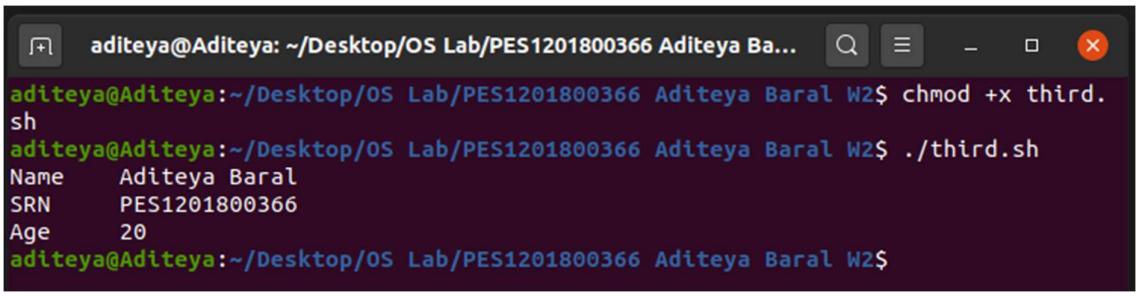
Hence, `echo $HOME` is the right command to display the home directory, since HOME is a system variable, and values stored in variables in a shell script can only be accessed using a `$` symbol as a prefix. Without the `$` symbol, the shell will read it as a normal string and display `HOME`.

## 6. User Defined Variables

### 6.1 Code

```
1. #!/bin/sh
2. name="Aditeya Baral"
3. srn=PES1201800366
4. age=20
5. echo "Name\t$name"
6. echo "SRN\t$srn"
7. echo "Age\t$age"
```

### 6.2 Output



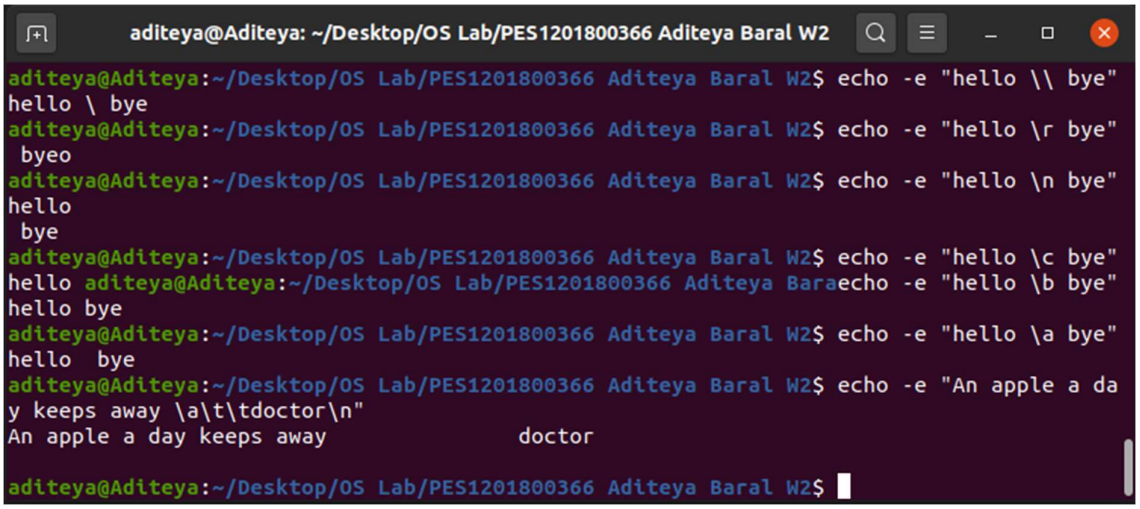

## 7. Echo

### 7.1 Code

```
1. #!/bin/sh
2. echo -e "hello \\ bye"
3. echo -e "hello \r bye"
4. echo -e "hello \n bye"
5. echo -e "hello \c bye"
6. echo -e "hello \b bye"
```

```
7. echo -e "hello \a bye"
8. echo -e "An apple a day keeps away \a\t\tdoctor\n"
```

### 7.2 Output

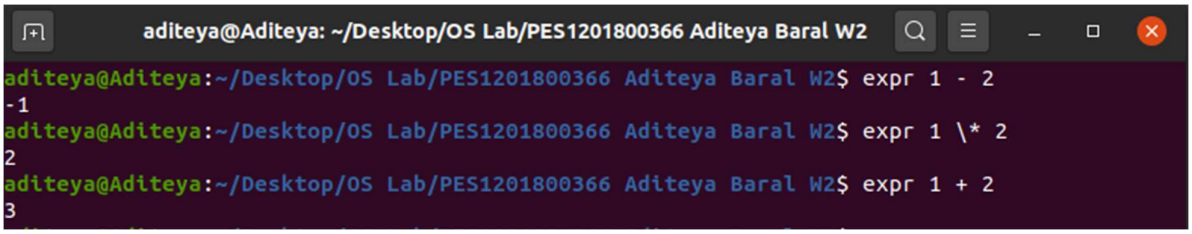



## 8. Shell Arithmetic

### 8.1 Code

```
1. expr 1 - 2
2. expr 1 \* 2
3. expr 1 + 2
```

### 8.2 Output

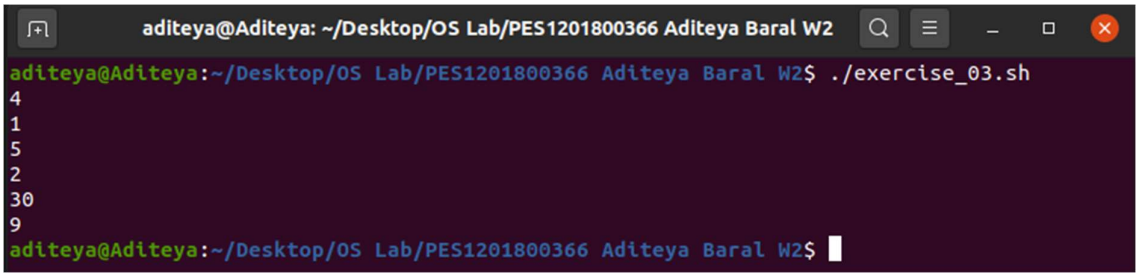



## 9. Exercise 3

### 9.1 Code

```
1. #1/bin/sh
2. expr 1 + 3
3. expr 2 - 1
4. expr 10 / 2
5. expr 20 % 3
6. expr 10 \* 3
7. echo `expr 6 + 3`
```

9.2 Output

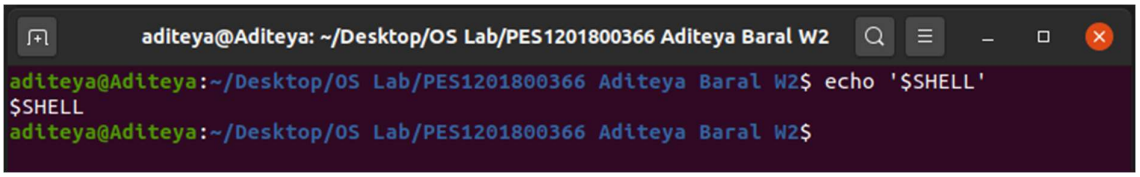



# 10. Exercise 4

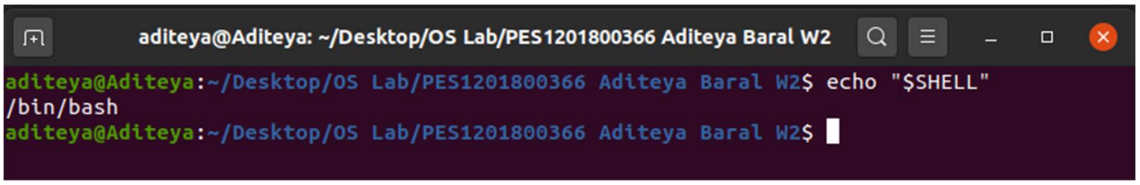
**Question - What is the meaning of Single quote (‘), Double quote (“) and Back quote (`) in shell?**

**Answer** – The shell understands special characters (such as escape sequences) with special meanings. For example, `$variable` is used to expand and obtain the value stored in variable. It also expands wildcards (such as `*` and `?`). However, sometimes, we need to display them as is. In such cases, we can use the various quoting methods.

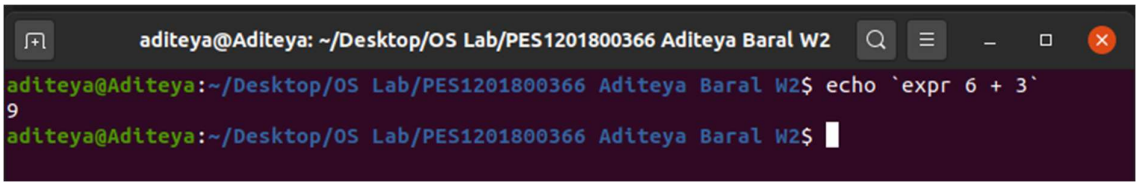
- **Single Quotes (‘):**
  - The single quote is used to remove the special meaning of **any character** enclosed within them i.e. the special characters are treated as ordinary strings
  - Variables, wildcards as well as command substitutions are disabled
  - Example: `echo ‘$SHELL’` will display `$SHELL`




- **Double Quotes (“):**
  - The double quote is used to remove the special meaning of **most characters** enclosed within them i.e. most special characters are treated as ordinary strings
  - Only wildcards are disabled
  - This does not apply to `$`, `‘`, `“`, `` ` ``, `\$`, and `\`
  - Example: `echo ‘$SHELL’` will display `/bin/bash`




- **Back Quotes (`):**
  - The back quote is used to **execute any command** enclosed within back quotes
  - Example: `echo `expr 6 + 3`` will display `9`

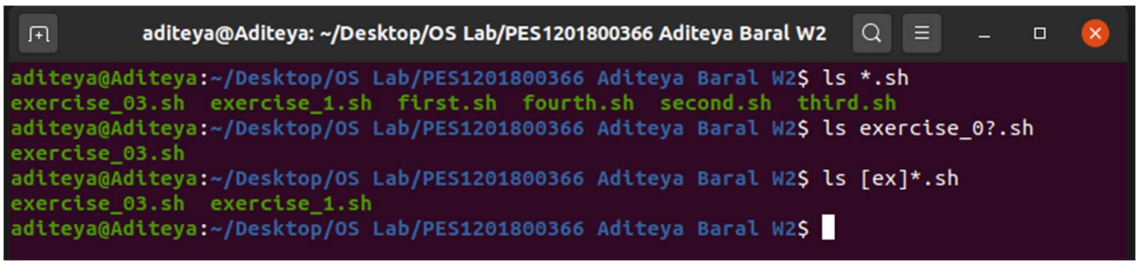



# 11. Wildcards

## 11.1 Code

```
1. #!/bin/sh
2. ls *.sh
3. ls exercise_0?.sh
4. ls [ex]*.sh
```

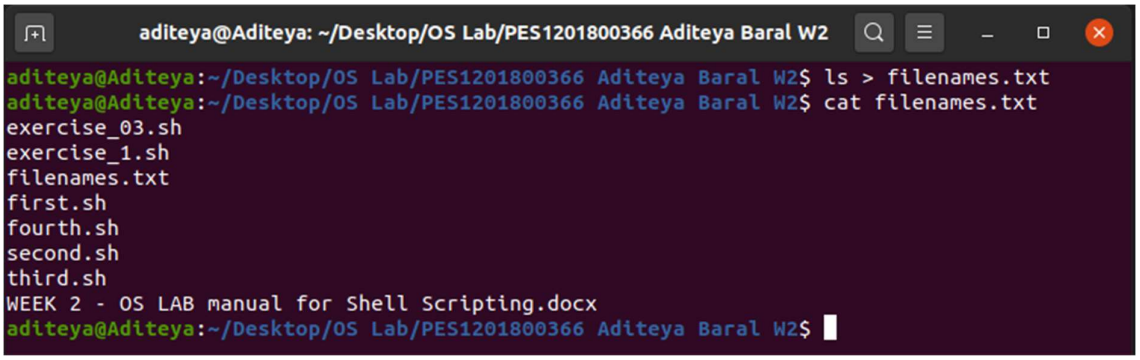
## 11.2 Output




# 12. Redirection of Standard IO

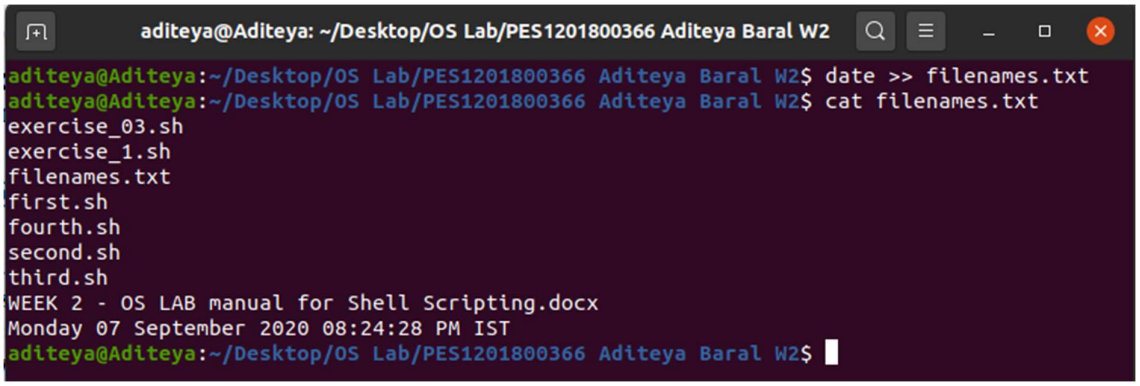
## 12.1 Output Redirection

```
1. ls > filenames.txt
2. cat filenames.txt
```



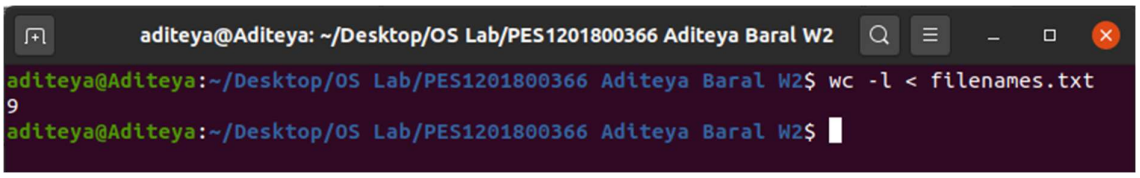

### 12.2 Output Redirection with Redirector

```
1. date >> filenames.txt
2. cat filenames.txt
```




### 12.3 Input Redirection

```
1. wc -l filenames.txt
```

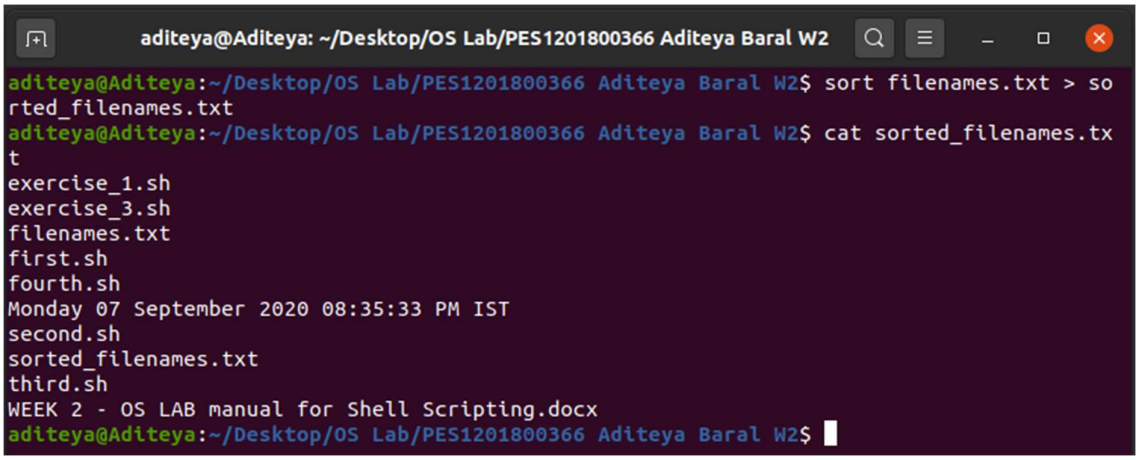



## 13. Exercise 5

**Question – What does the following command do?**

```
1. sort < myfile > sorted_file
```

**Answer** – The given command will first use input redirection to obtain the contents (lines) of myfile, which is sent to the sort command. After sorting the lines in alphabetical order, the output is redirected to sorted_file and stored.

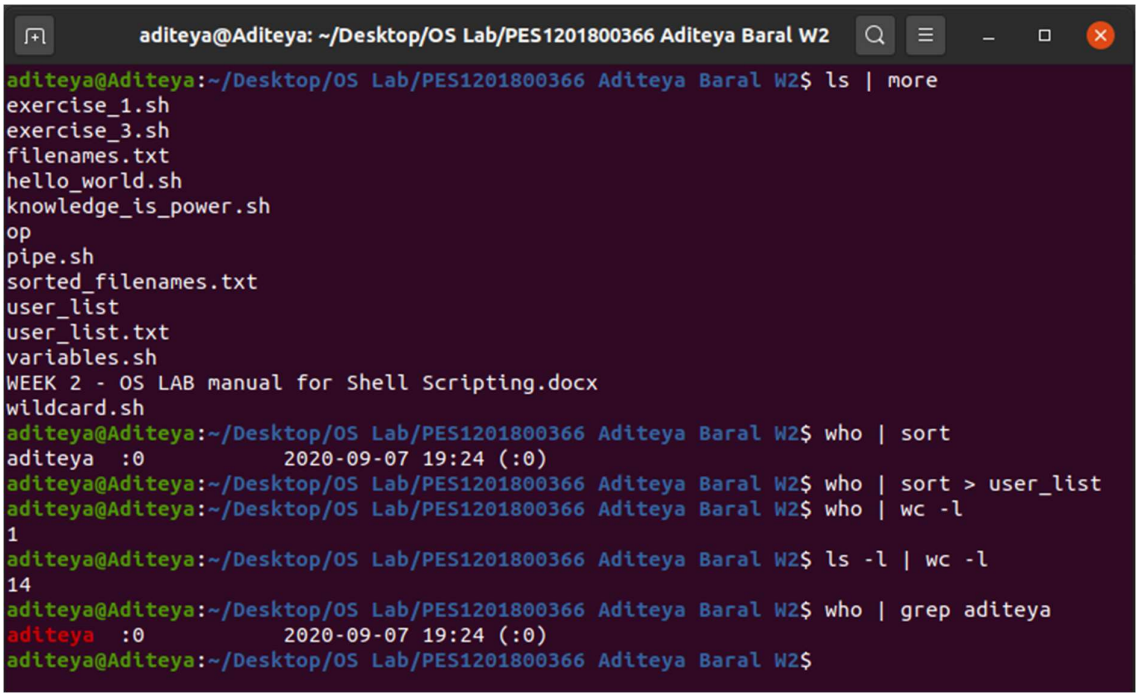



# 14. Pipes

## 14.1 Code

```
1. ls | more
2. who | sort
3. who | sort > user_list.txt
4. who | wc -l
5. ls -l | wc -l
6. who | grep aditeya
```

## 14.2 Output




# 15. if-else-fi Construct

### 15.1 Code

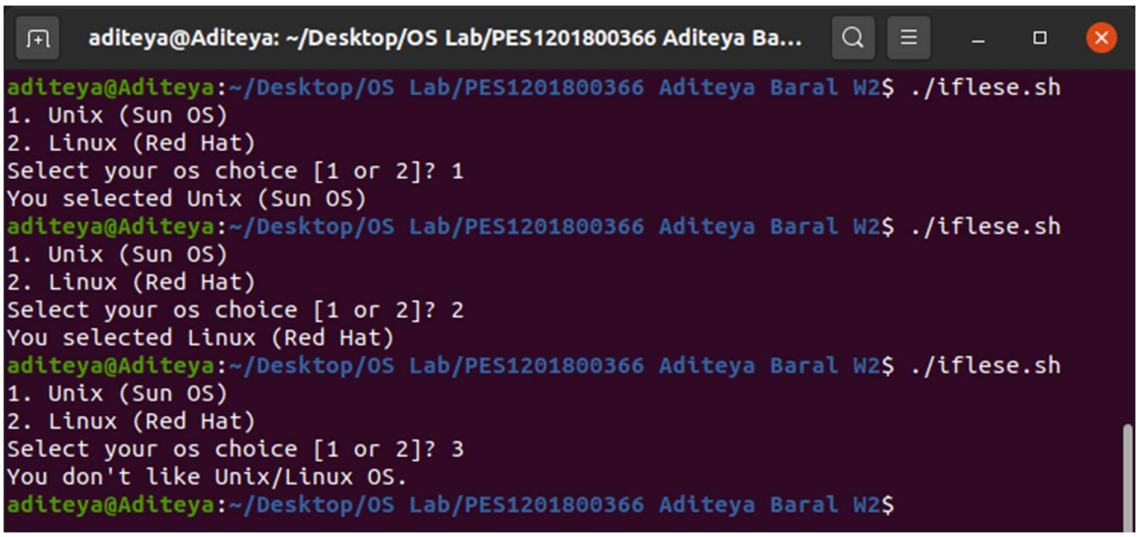
```
#!/bin/sh
osch=0

echo "1. Unix (Sun OS)"
echo "2. Linux (Red Hat)"
echo -n "Select your os choice [1 or 2]? "
read osch

if [ $osch -eq 1 ]
 then
     echo "You selected Unix (Sun OS)"

 else
     if [ $osch -eq 2 ]
     then
         echo "You selected Linux (Red Hat)"
     else
         echo "You don't like Unix/Linux OS."
     fi
 fi
```

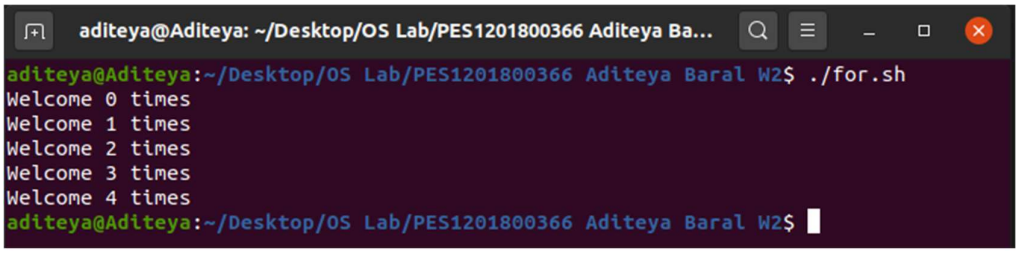
### 15.2 Output




## 16. Loops

### 16.1 For Loop

```
#!/bin/sh
for ((i = 0;i <= 4;i++))
do
    echo "Welcome $i times"
done
```

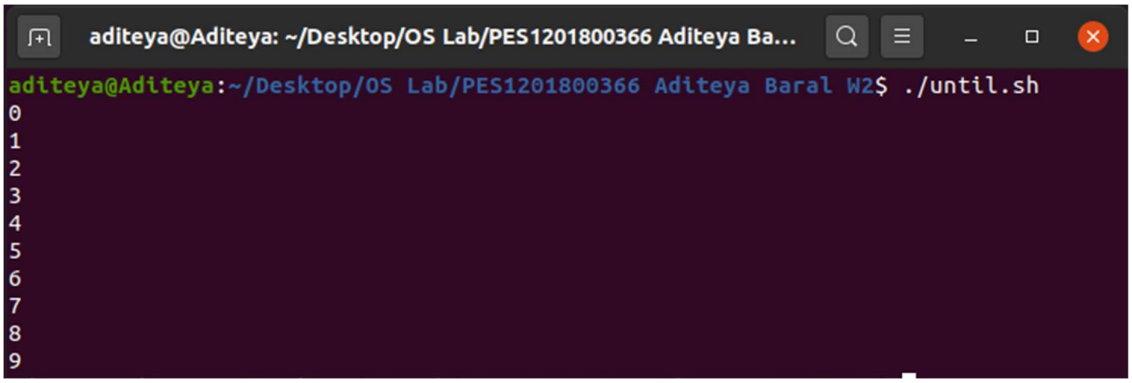



## 16.2 While Loop

```
1. #!/bin/sh
2. a=0
3. while [ $a -lt 10 ]
4. do
5.    echo $a
6.    a=`expr $a + 1`
7. done
```


## 16.3 Until Loop

```
1. #!/bin/sh
2. a=0
3. until [ ! $a -lt 10 ]
4. do
5.    echo $a
6.    a=`expr $a + 1`
7. done
```

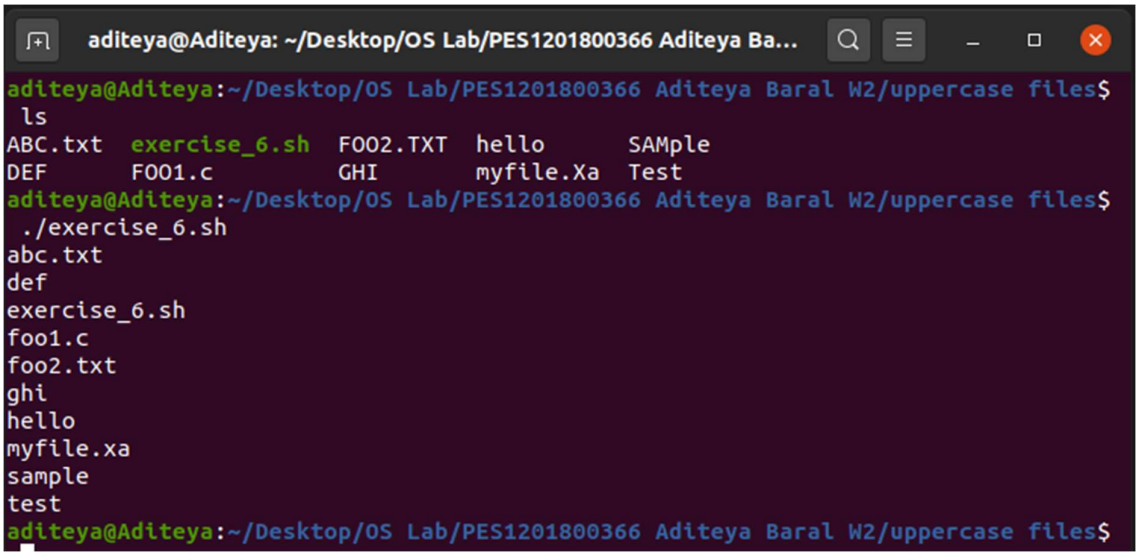


# 17. Exercise 6

## 17.1 Code

```
1. #!/bin/sh
2. ls | tr '[:upper:]' '[:lower:]'
```

## 17.2 Output




***