Commands I learnt:

1. cd – its basically used to change directory. If I give cd home/USERNAME/Desktop, lets say for example, the command will help me move to the Desktop in the username I'm currently logged into. If I give cd alone, it will automatically move to the home directory, regardless of which directory I'm at, hence its called as "absolute directory".

2. pwd – it will show which directory I'm currently at, in the given Username.

3. whoami – it will show the Username I'm currently logged into.

4. mkdir – it stands for "make directory". Its used to create a folder in which I can store/ create multiple other files/ folders. To create a folder inside a folder, I did something like mkdir -p dir4/dir5/dir6 – basically create the parent directories too. And to access these folders, as mentioned earlier, I did cd dir4/dir5/dir6. To make a directory with its name having space, I enclosed its name within quotes while creating it: like mkdir "folder 1".

5. ls – its a command used to list all the folders and file in the directory I am currently at.

6. echo – its a command used to repeat whatever command I say. A usecase of this is to "echo" the texts or commands that I give into a given file, say test_1.txt. All that I have to do is: echo "Hello World!" > test_1.txt. Lets say that I have another file test_2.txt in which theres "Happy New year 2024!". If I want to concatenate these 2 texts, into a file called test_3.txt, what I'd do is: cat test_* > test_3.txt. The command cat test_* echoes the data in all the files whose names start with test_. Here theres only test_1 and test_2. So only they are echoed, and into test_3.txt as I've mentioned it. Now, to check test_3.txt, I say ls cat test_3.txt. It would show "Hello World! \n Happy New year 2024!". Similarly, if I want to append the text rather than replace, Id use cat test_* >> test_3.txt. Append basically doesn't erase whats already existing. It only adds onto whats already there.

7. less – this command is used to display the file one page at a time for convenience. I personally didn't use this command much.

8. mv – I used this command to move files around different directories. The way I did this is: mv test_4.txt dir1 – where mv is the move command, test_4.txt is the file I want to move, and dir1 is the directory where I want it to be moved. If by chance I want to move everything in a particular directory to another, I'd do: mv dir1/* dir2, where dir1 is the directory from which I will move the files and dir2 is the directory where I'll move all my files. A cool thing is, we can use the same logic as the echo command – if i want to move all the files named test_ initially to lets say, dir1, I can do: mv test_* dir1. I can also move multiple files/folders at once like: mv test_* output.txt dir3 dir2 where test_, output.txt and dir3 are the files/folders to be moved to dir2. If I now want to move output.txt in dir2 to dir6 which is in dir5, which is in dir4, which is in dir1(so sorry ;-;), I'd do: mv dir2/output.txt dir4/dir5/dir6. Lastly, I used the mv command to rename a file like: mv combined.txt combined2.txt . Cool stuff.

9. cp – I used this command to make copies of files just in case I lost anyone of them. Backups, eh? mv combined2.txt combined2_backup.txt

10. rm – I used this command to delete some files/ folders. I simply had to do: rm output.txt if directory: rmdir dir1 and if that directory had files and folders: rm -r dir1. If i wanted to delete a directory in which there are many files and folders, and I dont want to delete it without knowing if there are contents, i can do rmdir -p dir1/dir2/dir3 which would first delete dir3, then 2 and then 1,

so it would only delete empty directories. If dir1 had other files, only 2 and 3 would get deleted. Its the exact opposite of what we did while creating files/folders.

11. wc – I used this command to find out the numeber of lines, words and characters used in a text file. Didn't use it much tho

12. piping – this is probably one of the most useful things ive seen. I used it to perform multiple commands at once – something like getting an output from one command, and using it as an input for another command. Like: ls ~ | wc -l. What this does is to first list the items present in home, and echoes this data directly to the word count command, where ur only going to view the number of lines, hence the number of items. So cool. While doing this i learnt about the sort command, which sorts according to alphanumericals and length of the string, and uniq which basically only echoes the unique lines. What we can do is: sort output.txt | uniq | wc -l for example, where it basically sorts output.txt, only echoes the unique lines into the word count command, such that only number of lines is visible.

13. sudo – ok until now i thought whatever was done was cool, but omg this is fun. Sudo is basically super user do (sound like srmthfg). We can install stuff, ask it to display shady stuff(not that shady), given that we provide the correct password coz we dont want non super users to access it? For example i did sudo apt install tree to implement hierarchy of files in directories.

14. hidden files – shady stuff, eh? Basically to convert a file into a hidden one, do
mv output.txt .output.txt. And to access it, cat .output.txt. To see all hidden files, do ls -a.
basically u can do the same operations on the hidden files, just get the name right.

SHELL SCRIPTING

1. installed vim – sudo apt-get install vim
2. created file: vim test
3. capital A to start typing
4. esc to escape from typing
5. :q to exit to the main terminal
6. :set number to set numbers for each line
7. :w to save file
8. :wq to save and quit
9. :q! To discard changes and quit
10. w to jump thru forward, b to go backward (word by word)
11. 0 to go to start of line
12. G to go to end of file
13. $ to go to end of line
14. :syntax on to highlight syntax
15. :set tabstop=x to change tab space
16. :set autoindent to indent automatically
all these are saved in home directory, in a file called .vimrc
to check which vimrc – do :echo $MYVIMRC
17. once ive finished typing and saving, id exit and execute the vim file. If it doesnt run, in the terminal id say: chmod 755 filename, which would create the executable. Now it would run if i gave the same command 755 means the owner can do anything with the file or directory, and other users can read and execute it, but not change it. 777 means everyone can do anything to it.

Shell scripting is basically like c

```
#!/bin/bash
# Comment
echo "Hello World"
myName="Bhadresh"
declare -r NUM1=5
NUM2=4
num3=$((NUM1+NUM2))
num4=$((NUM1-NUM2))
num5=$((NUM1*NUM2))
num6=$((NUM1/NUM2))
echo "5+4=$num3"
echo "5-4=$num4"
echo "5*4=$num5"
echo "5/4=$num6"
echo $((5**2))
echo $((5%4))
rand=5
let rand+=4
echo "$rand"
echo "rand++ = $((rand++))"
echo "rand-- = $((rand--))"
echo "--rand = $((--rand))"
echo "++rand = $((++rand))"
~
```

```
badbud@BVLL-ASUS:~/working$ ./test1
Hello World
5+4=9
5-4=1
5*4=20
5/4=1
25
1
9
rand++ = 9
rand-- = 10
--rand = 8
++rand = 9
```

```
#!/bin/bash
# Comment
echo "Hello World"
myName="Bhadresh"
declare -r NUM1=5
NUM2=4
num3=$((NUM1+NUM2))
num4=$((NUM1-NUM2))
num5=$((NUM1*NUM2))
num6=$((NUM1/NUM2))
echo "5+4=$num3"
echo "5-4=$num4"
echo "5*4=$num5"
echo "5/4=$num6"
echo $((5**2))
echo $((5%4))
rand=5
let rand+=4
echo "$rand"
echo "rand++ = $((rand++))"
echo "rand-- = $((rand--))"
echo "--rand = $((--rand))"
echo "++rand = $((++rand))"
cat<< END
This test
prints on
many lines
END
```

```
badbud@BVLL-ASUS:~/working$ ./test1
Hello World
5+4=9
5-4=1
5*4=20
5/4=1
25
1
9
rand++ = 9
rand-- = 10
--rand = 8
++rand = 9
This test
prints on
many lines
```

```bash
#!/bin/bash

read -p "Whats your name?" name
echo "Hello $name"

```

```
badbud@BVLL-ASUS:~/working$ ./test4
Whats your name?Bhadresh
Hello Bhadresh
badbud@BVLL-ASUS:~/working$ S
```

```bash
#!/bin/bash

getDate()
{
        date
        return
}
getDate

name="Bhadresh"
demLocal()
{
        local name="Vaageesh"
        return
}
demLocal
echo "$name"

getSum()
{
        local num3=$1
        local num4=$2
        local sum=$((num3+num4))
        echo $sum
}
num1=5
num2=6
sum=$(getSum num1 num2)
echo "the sum is $sum"
```

```
badbud@BVLL-ASUS:~/working$ ./test3
Monday 01 April 2024 11:33:58 AM IST
Bhadresh
the sum is 11
```

```bash
#!/bin/bash

read -p "How old are you? " age

if [ $age -ge 16 ]
then
  echo "You can drive"
elif [ $age -eq 15 ]
then
  echo "You can drive next year"
else
  echo "You can't drive"
fi

```

```
$ ./hello_world
How old are you? 15
You can drive next year
$ ./hello_world
How old are you? 16
You can drive
$ ./hello_world
How old are you? 2
You can't drive
$
```

LIGHT DOSE – SHELL SCRIPTING

```
badbud@BVLL-ASUS:~/working$ gedit dir1/p1.txt dir2/p2.txt dir3/p3.txt p4.txt
badbud@BVLL-ASUS:~/working$ ls dir1
p1.txt
badbud@BVLL-ASUS:~/working$ tree
.
|-- dir1
|   `-- p1.txt
|-- dir2
|   `-- p2.txt
|-- dir3
|   `-- p3.txt
|-- find_text
|-- modified
|-- p4.txt
|-- test1
|-- test2
|-- test3
`-- test4

4 directories, 9 files
badbud@BVLL-ASUS:~/working$ ./find_text
mv: './modified/p4.txt' and './modified/p4.txt' are the same file
mv: './modified/p3.txt' and './modified/p3.txt' are the same file
DONE!!!
badbud@BVLL-ASUS:~/working$ ls modified
p1.txt.bak  p2.txt.bak  p3.txt.bak  p4.txt.bak
badbud@BVLL-ASUS:~/working$ tree
.
|-- dir1
|-- dir2
|-- dir3
|-- find_text
|-- modified
|   |-- p1.txt.bak
|   |-- p2.txt.bak
|   |-- p3.txt.bak
|   `-- p4.txt.bak
|-- test1
|-- test2
|-- test3
`-- test4

4 directories, 9 files
badbud@BVLL-ASUS:~/working$
```

```
#!/bin/bash
text_files="./modified"
mkdir -p "$text_files"
find . -type f -name "*.txt" -exec mv {} "$text_files"  \;
find "$text_files" -type f -name "*.txt" -exec mv {} {}.bak \;
echo "DONE!!!"
~
~
```