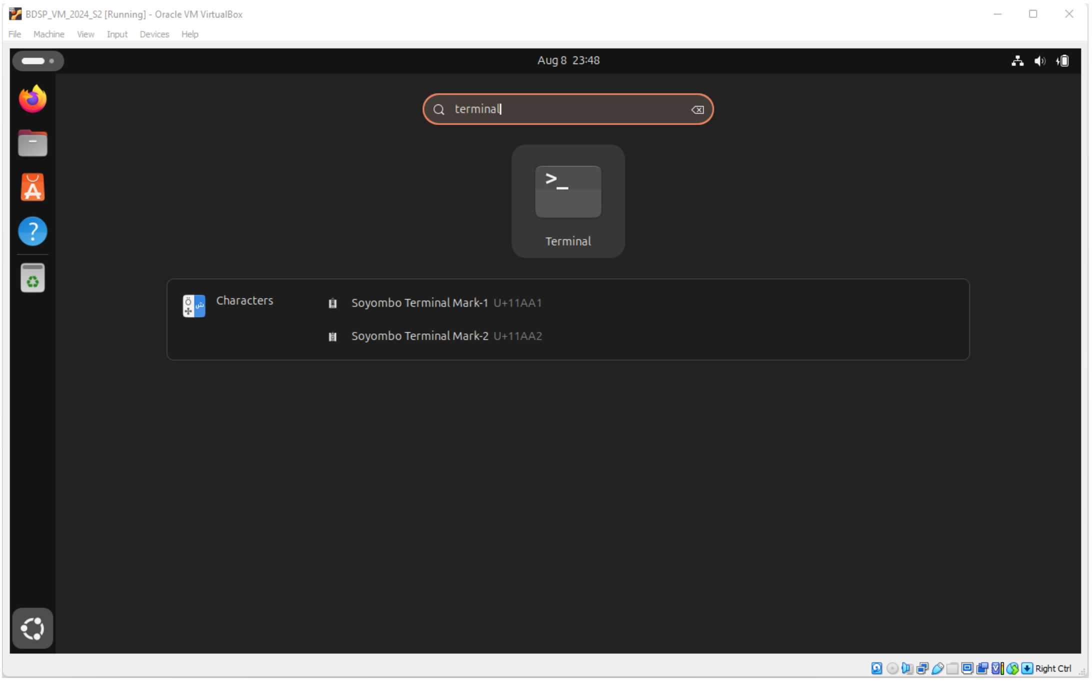
Tutorial 1

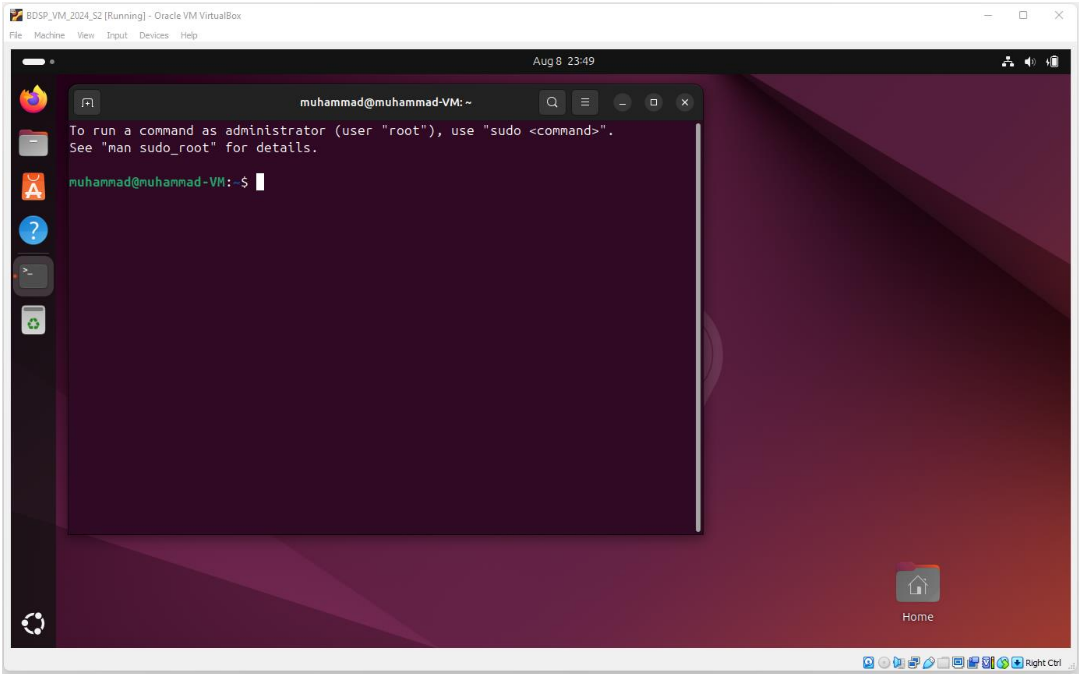
Introduction to Linux Commands

1) On Ubuntu 24.04 LTS system, we can find a launcher for the terminal by clicking on the Activities item at the bottom left of the screen, then typing the first few letters of “terminal”, “command”, “prompt” or “shell”.


BDSP_VM_2024_S2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Aug 8 23:48
terminal
Terminal
Characters
Soyombo Terminal Mark-1 U+11AA1
Soyombo Terminal Mark-2 U+11AA2
Right Ctrl

If you can’t find a launcher, or if you just want a faster way to bring up the terminal, most Linux systems use the same default keyboard shortcut to start it: **Ctrl-Alt-t**.

Depending on your Linux system, the colours may not be the same, and the text will likely say something different, but the general layout of a window with a large (mostly empty) text area should be similar.


BDSP_VM_2024_S2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Aug 8 23:49
muhammad@muhammad-VM: ~
Home
Right Ctrl

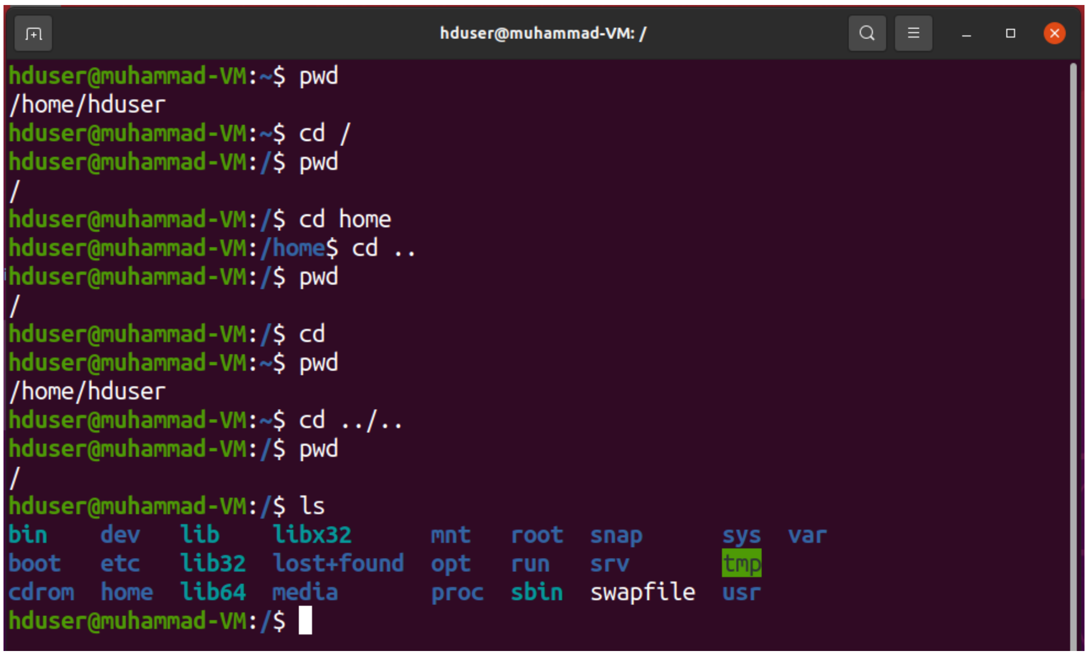
```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

muhammad@muhammad-VM:~$
```

Let's run our first command.

```
pwd
```

You should see a directory path printed out (probably something like **/home/YOUR_USERNAME**), then another copy of that odd bit of text. Following screenshot will give you an idea how to type the commands on the terminal/ shell.



```
hduser@muhammad-VM:~$ pwd
/home/hduser
hduser@muhammad-VM:~$ cd /
hduser@muhammad-VM:/$ pwd
/
hduser@muhammad-VM:/$ cd home
hduser@muhammad-VM:/home$ cd ..
hduser@muhammad-VM:/$ pwd
/
hduser@muhammad-VM:/$ cd
hduser@muhammad-VM:~$ pwd
/home/hduser
hduser@muhammad-VM:~$ cd ../..
hduser@muhammad-VM:/$ pwd
/
hduser@muhammad-VM:/$ ls
bin    dev   lib    libx32      mnt   root  snap      sys  var
boot   etc   lib32  lost+found  opt   run   srv       tmp
cdrom  home  lib64  media       proc  sbin  swapfile  usr
hduser@muhammad-VM:/$
```

Linux commands are **case sensitive**. You can change the working directory using the **cd** command, an abbreviation for 'change directory'. Try typing the following:

```
cd /
pwd
```

You can see a "/" directory, referred to as the *root* directory, is the base of that unified file system. From there everything else branches out to form a tree of directories and subdirectories. From the root directory, the following command will move you into the "home" directory (which is an immediate subdirectory of "/"):

```
cd home
pwd
```

To go up to the parent directory, in this case back to "/", use the special syntax of two dots (..) when changing directory (note the space between cd and .., unlike in DOS you can't just type cd.. as one command):

```
cd ..
pwd
```

Typing cd on its own is a quick shortcut to get back to your home directory:

```
cd
pwd
```

You can also use .. more than once if you have to move up through multiple levels of parent directories:

```
cd ../..
pwd
ls
```

ls means display the list of files and folders in a directory.

Creating folders and files

We are going to create some real files to work with. To avoid accidentally trampling over any of your real files, we're going to start by creating a new directory, well away from your home folder, which will serve as a safer environment in which to experiment:

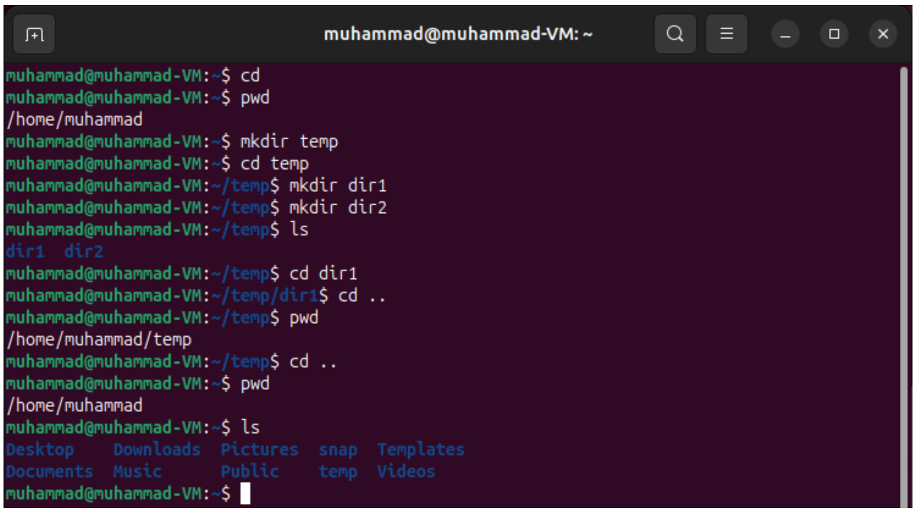
cd Hit the Enter Key
pwd
The directory should be /home/muhammad
mkdir temp
cd temp
mkdir dir1
mkdir dir2

There's something a little different about that command. So far we have only seen commands that work on their own (cd, pwd) or that have a single item afterwards (cd /, cd ~/Desktop).

List (ls) command:

ls

If you've followed the last few commands, your terminal should be looking something like this:



```
muhammad@muhammad-VM:~$ cd
muhammad@muhammad-VM:~$ pwd
/home/muhammad
muhammad@muhammad-VM:~$ mkdir temp
muhammad@muhammad-VM:~$ cd temp
muhammad@muhammad-VM:~/temp$ mkdir dir1
muhammad@muhammad-VM:~/temp$ mkdir dir2
muhammad@muhammad-VM:~/temp$ ls
dir1  dir2
muhammad@muhammad-VM:~/temp$ cd dir1
muhammad@muhammad-VM:~/temp/dir1$ cd ..
muhammad@muhammad-VM:~/temp$ pwd
/home/muhammad/temp
muhammad@muhammad-VM:~/temp$ cd ..
muhammad@muhammad-VM:~$ pwd
/home/muhammad
muhammad@muhammad-VM:~$ ls
Desktop    Downloads  Pictures  snap  Templates
Documents  Music      Public    temp  Videos
muhammad@muhammad-VM:~$
```

mkdir another folder
ls

Creating files using redirection

Our demonstration folder is starting to look rather full of directories but is somewhat lacking in files. Let's remedy that by redirecting the output from a command so that, instead of being printed to the screen, it ends up in a new file. First, remind yourself what the **ls** command is currently showing:

ls

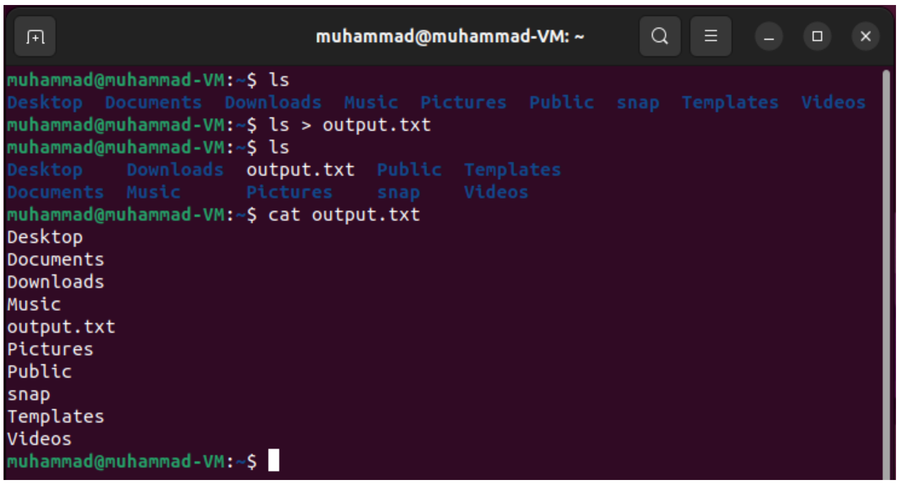
Suppose we wanted to capture the output of that command as a text file that we can look at or manipulate further. All we need to do is to add the greater-than character (">") to the end of our command line, followed by the name of the file to write to:

ls > output.txt

This time there's nothing printed to the screen, because the output is being redirected to our file instead. If you just run **ls** on its own you should see that the *output.txt* file has been created. We can use the **cat** command to look at its content:

cat output.txt

Now it is not *exactly* what was displayed on the screen previously, but it contains all the same data, and it's in a more useful format for further processing.



```
muhammad@muhammad-VM:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
muhammad@muhammad-VM:~$ ls > output.txt
muhammad@muhammad-VM:~$ ls
Desktop    Downloads  output.txt  Public  Templates
Documents  Music      Pictures    snap    Videos
muhammad@muhammad-VM:~$ cat output.txt
Desktop
Documents
Downloads
Music
output.txt
Pictures
Public
snap
Templates
Videos
muhammad@muhammad-VM:~$
```

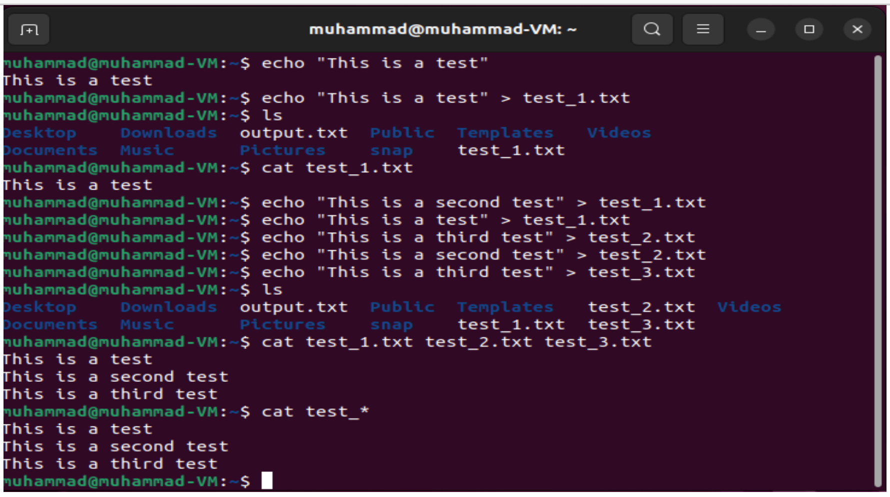
All folder/ file names are copies into output.txt using > command.

Let's look at another command, echo:

```
echo "This is a test"
```

Yes, echo prints its arguments back out again (hence the name). But combine it with a redirect, and you've got a way to easily create small test files:

```
echo "This is a test" > test_1.txt
echo "This is a second test" > test_2.txt
echo "This is a third test" > test_3.txt
ls
```



```
muhammad@muhammad-VM:~$ echo "This is a test"
This is a test
muhammad@muhammad-VM:~$ echo "This is a test" > test_1.txt
muhammad@muhammad-VM:~$ ls
Desktop    Downloads  output.txt  Public  Templates   Videos
Documents  Music      Pictures    snap    test_1.txt
muhammad@muhammad-VM:~$ cat test_1.txt
This is a test
muhammad@muhammad-VM:~$ echo "This is a second test" > test_1.txt
muhammad@muhammad-VM:~$ echo "This is a test" > test_1.txt
muhammad@muhammad-VM:~$ echo "This is a third test" > test_2.txt
muhammad@muhammad-VM:~$ echo "This is a second test" > test_2.txt
muhammad@muhammad-VM:~$ echo "This is a third test" > test_3.txt
muhammad@muhammad-VM:~$ ls
Desktop    Downloads  output.txt  Public  Templates   test_2.txt  Videos
Documents  Music      Pictures    snap    test_1.txt  test_3.txt
muhammad@muhammad-VM:~$ cat test_1.txt test_2.txt test_3.txt
This is a test
This is a second test
This is a third test
muhammad@muhammad-VM:~$ cat test_*
This is a test
This is a second test
This is a third test
muhammad@muhammad-VM:~$
```

You should cat each of these files to check their contents. But cat is more than just a file viewer - its name comes from 'concatenate', meaning "to link together". If you pass more than one filename to cat it will output each of them, one after the other, as a single block of text:

```
cat test_1.txt test_2.txt test_3.txt
```

Where you want to pass multiple file names to a single command, there are some useful shortcuts that can save you a lot of typing if the files have similar names. A question mark ("?") can be used to indicate "any single character" within the file name. An asterisk ("*") can be used to indicate "zero or more characters". These are sometimes referred to as "wildcard" characters. A couple of examples might help, the following commands all do the same thing:

```
cat test_1.txt test_2.txt test_3.txt
cat test_?.txt
cat test_*
```

More escaping required

As you might have guessed, this capability also means that you need to escape file names with ? or * characters in them, too. It's usually better to avoid any punctuation in file names if you want to manipulate them from the command line.

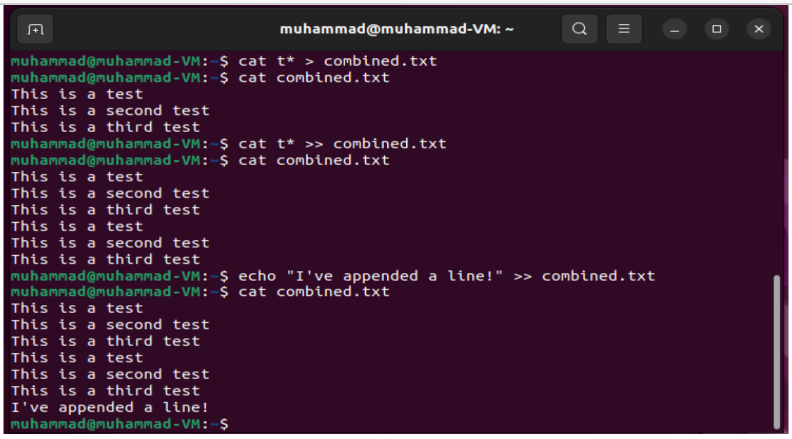
If you look at the output of ls you will notice that the only files or folders that start with "t" are the three test files we have created, so you could even simplify that last command even further to cat t*, meaning "concatenate all the files whose names start with a *t* and are followed by zero or more other characters". Let's use this capability to join all our files together into a single new file, then view it:

```
cat t* > combined.txt
cat combined.txt
```

What do you think will happen if we run those two commands a second time? Will the computer complain, because the file already exists? Will it append the text to the file, so it contains two copies? Or will it replace it entirely? Give it a try to see what happens, but to avoid typing the commands again you can use the Up Arrow and Down Arrow keys to move back and forth through the history of commands you've used. Press the Up Arrow a couple of times to get to the first cat and press Enter to run it, then do the same again to get to the second.

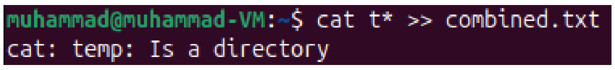
As you can see, the file looks the same. That's not because it's been left untouched, but because the shell clears out all the content of the file before it writes the output of your **cat** command into it. Because of this, you should be extra careful when using redirection to make sure that you don't accidentally overwrite a file you need. If you do want to append to, rather than replace, the content of the files, double up on the greater-than character:

```
cat t* >> combined.txt
echo "I've appended a line!" >> combined.txt
cat combined.txt
```



```
muhammad@muhammad-VM:~$ cat t* > combined.txt
muhammad@muhammad-VM:~$ cat combined.txt
This is a test
This is a second test
This is a third test
muhammad@muhammad-VM:~$ cat t* >> combined.txt
muhammad@muhammad-VM:~$ cat combined.txt
This is a test
This is a second test
This is a third test
This is a test
This is a second test
This is a third test
muhammad@muhammad-VM:~$ echo "I've appended a line!" >> combined.txt
muhammad@muhammad-VM:~$ cat combined.txt
This is a test
This is a second test
This is a third test
This is a test
This is a second test
This is a third test
I've appended a line!
muhammad@muhammad-VM:~$
```

If you faced error after cat t* >> combined.txt after this command as mentioned below



```
muhammad@muhammad-VM:~$ cat t* >> combined.txt
cat: temp: Is a directory
```

This happened due to creation of temp folder at the start. You can update this command by introducing two more letters

$cat tes* >> combined.txt

This command (cat t* >> combined.txt) will work after this update. We discuss also in the class in the case of any questions.

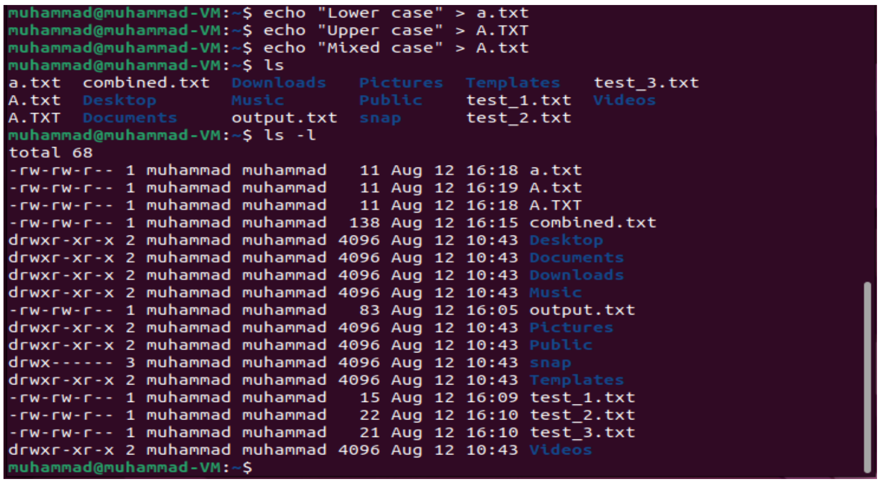
A note about case

Unix systems are case-sensitive, that is, they consider “A.txt” and “a.txt” to be two different files. If you were to run the following lines you would end up with three files:

```
echo "Lower case" > a.txt
echo "Upper case" > A.TXT
echo "Mixed case" > A.txt
```

Generally, you should try to avoid creating files and folders whose name only varies by case. Not only will it help to avoid confusion, but it will also prevent problems when working with different operating systems. Windows, for example, is case-*insensitive*, so it would treat all three of the file names above as being a single file, potentially causing data loss or other problems.

You might be tempted to hit the Caps Lock key and use upper case for all your file names. But most shell commands are lower case, so you would end up frequently having to turn it on and off as you type. Most seasoned command line users tend to stick primarily to lower case names for their files and directories so that they rarely must worry about file name clashes, or which case to use for each letter in the name.



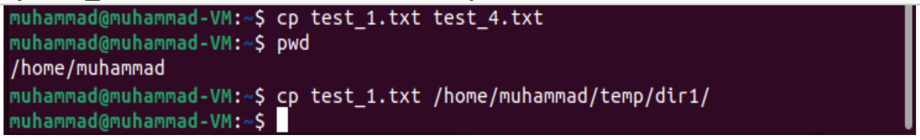
```
muhammad@muhammad-VM:~$ echo "Lower case" > a.txt
muhammad@muhammad-VM:~$ echo "Upper case" > A.TXT
muhammad@muhammad-VM:~$ echo "Mixed case" > A.txt
muhammad@muhammad-VM:~$ ls
a.txt  combined.txt  Downloads   Pictures  Templates   test_3.txt
A.txt  Desktop       Music       Public    test_1.txt  Videos
A.TXT  Documents     output.txt  snap      test_2.txt
muhammad@muhammad-VM:~$ ls -l
total 68
-rw-rw-r-- 1 muhammad muhammad   11 Aug 12 16:18 a.txt
-rw-rw-r-- 1 muhammad muhammad   11 Aug 12 16:19 A.txt
-rw-rw-r-- 1 muhammad muhammad   11 Aug 12 16:18 A.TXT
-rw-rw-r-- 1 muhammad muhammad  138 Aug 12 16:15 combined.txt
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Desktop
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Documents
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Downloads
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Music
-rw-rw-r-- 1 muhammad muhammad   83 Aug 12 16:05 output.txt
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Pictures
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Public
drwx------ 3 muhammad muhammad 4096 Aug 12 10:43 snap
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Templates
-rw-rw-r-- 1 muhammad muhammad   15 Aug 12 16:09 test_1.txt
-rw-rw-r-- 1 muhammad muhammad   22 Aug 12 16:10 test_2.txt
-rw-rw-r-- 1 muhammad muhammad   21 Aug 12 16:10 test_3.txt
drwxr-xr-x 2 muhammad muhammad 4096 Aug 12 10:43 Videos
muhammad@muhammad-VM:~$
```

ls -l showed all details and privileges of the files and folders

Copy (cp) Files and Directories

The **cp** command will copy files and directories or copy multiple sources to a destination directory. The basic syntax of the cp command is:

```
cp test_1.txt test_4.txt
cp test_1.txt /home/muhammad/temp/dir1/
```



```
muhammad@muhammad-VM:~$ cp test_1.txt test_4.txt
muhammad@muhammad-VM:~$ pwd
/home/muhammad
muhammad@muhammad-VM:~$ cp test_1.txt /home/muhammad/temp/dir1/
muhammad@muhammad-VM:~$ 
```

move (mv) Files and Directories

The **cp** command will copy files and directories or copy multiple sources to a destination directory. The basic syntax of the **cp** command is:

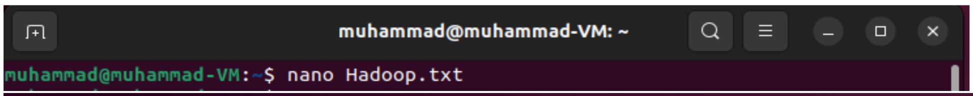
```
mv test_1.txt /home/muhammad/temp/dir2/
```

nano editor to create a new file

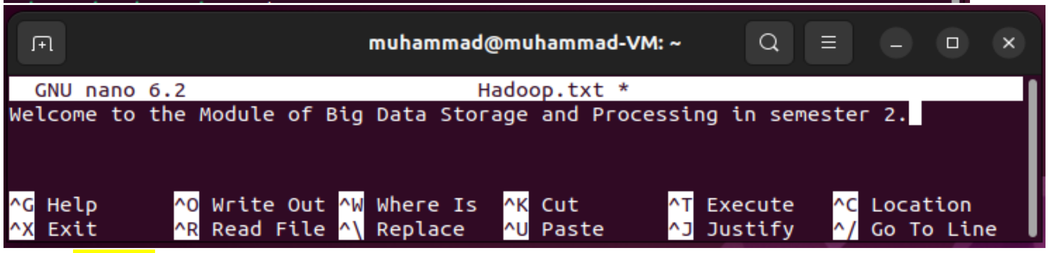
The **nano** command will be used for editing a file and the syntax is mentioned below

```
nano Hadoop.txt
```

An editor will be opened and write a message inside the editor, “We are going to start working on Hadoop in the next week”. To save text message in a file using nano editor, press **ctrl + x**, you will get a message, “save modified buffer”, press a key “**y**”, you will get a message that you like to save the in the same file. Just press the “Enter” key. Your text is saved in the file named as Hadoop.txt. If you would like to check again, write **nano Hadoop.txt** again on the terminal. If you do not perform any changes in the text file (Hadoop.txt), press ctrl + x to exit from the nano editor.



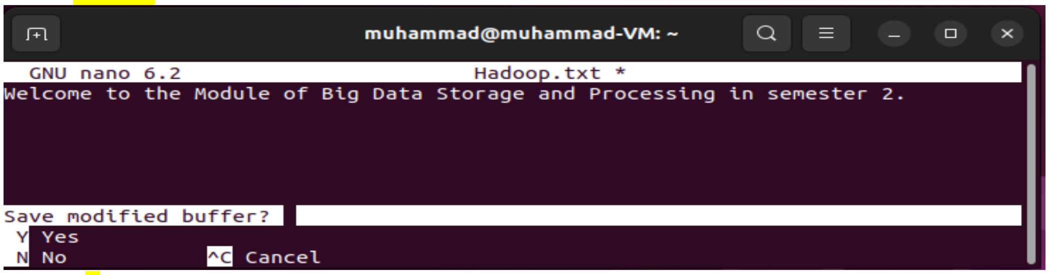
```
muhammad@muhammad-VM:~$ nano Hadoop.txt
```



```
  GNU nano 6.2                       Hadoop.txt *
Welcome to the Module of Big Data Storage and Processing in semester 2.


^G Help         ^O Write Out ^W Where Is  ^K Cut         ^T Execute   ^C Location
^X Exit         ^R Read File ^\ Replace   ^U Paste       ^J Justify   ^/ Go To Line
```

Press **ctrl + x** to exit from the nano editor.

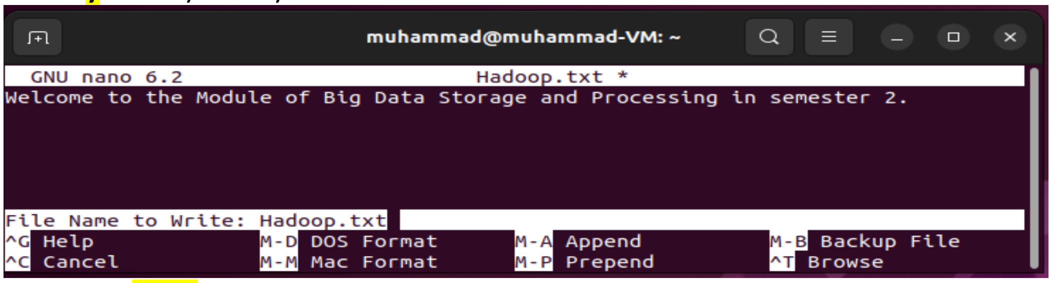


```
  GNU nano 6.2                       Hadoop.txt *
Welcome to the Module of Big Data Storage and Processing in semester 2.




Save modified buffer?
 Y Yes
 N No                ^C Cancel
```

Press “**y**” from your key board.

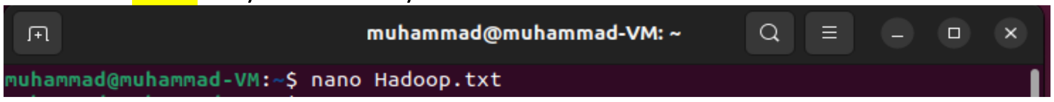


```
  GNU nano 6.2                       Hadoop.txt *
Welcome to the Module of Big Data Storage and Processing in semester 2.




File Name to Write: Hadoop.txt
^G Help               M-D DOS Format      M-A Append          M-B Backup File
^C Cancel             M-M Mac Format      M-P Prepend         ^T Browse
```

Press the “**Enter**” key from the keyboard to save the contents of the file.



```
muhammad@muhammad-VM:~$ nano Hadoop.txt
```

chmod command for shell scripts

nano script.sh
A nano editor will be opened and write the following command in the file as
echo "Welcome to CCT College."
Press **ctrl + x** keys simultaneously and press **y** to save the command in the file.
ls -l
check the privileges of script.sh file and the file does not have executable privileges. Use the following command to make the file executable as
chmod 700 script.sh
The output will be displayed by executing the command as mentioned below
./script.sh

Table of commands (commands are case sensitive):

pwd	Display path of working directory name.
cd	Change directory.
cd ..	Move to previous folder/ directory
ls	List contents of directory.
ls -l	Long list of contents of directory (shows more information for owner of files).
ls -lht	What changes observed than the previous command?
info pwd	Display information on the pwd command.
mkdir folder/ directory	Make a new directory called folder/ directory.
echo Welcome	Output Welcome.
> file_1.txt	Create a new empty file called **file_1.txt** in the current working directory.
echo file_2 > file_1.txt	Redirect the output from the **echo** command to a file called **file_1.txt.**
cat file_1.txt	Output the contents of the **file_1.txt** file.
cp file_1.txt file_2.txt	Create a copy of the file named **file_1.txt**. The copied file will be called **file_2.txt**.
mv file_1.txt file_11.txt	Move the file named **file_1.txt** to a file named as **file_11.txt.**
rm file_1.txt	Delete the file named **file_1.txt.**
rm -rf folder_name	Delete the folder name.
chmod	**chmod** is the command and system call used to change the access permissions of file system objects. Setting 777 permissions to a file or directory means that it will be readable, writable and executable by all users and may pose a huge security risk.
chown	The **chown** command allows you to change the user and/or group ownership of a given file, directory, or symbolic link

Unix File/ Directory Permissions

Unix permissions are granted to three different entities:

- Owner (---)
- Group members (---)
- Others (i.e. The World) (---)

There are three permission attributes: r, w and x.

Read (r): Allows reading the content of the file. For directories, it allows listing the contents of the directory.

Write (w): Allows modifying the content of the file. For directories, it allows adding, removing, and modifying files within the directory.

Execute (x): For regular files, this permission allows executing the file if it's a script or a binary executable. For directories, it allows entering the directory and accessing its contents.

File permissions are typically represented as a combination of these symbols for the owner, group, and others. For example:

-rw-r--r--: The owner has read and write permissions, while the group and others have read-only permissions. This is a common permission setting for regular files.

-rwxr-x---: The owner has read, write, and execute permissions, the group has read and execute permissions, and others have no access. This is often used for executable files.

drwxrwxr-x: This is the permission set for a directory. The owner and group have read, write, and execute permissions, while others have read and execute permissions.

Each of these symbols corresponds to a specific position in the permission string. If you were referring to a specific context or command related to "xrw," please provide more details so that I can give you accurate information.

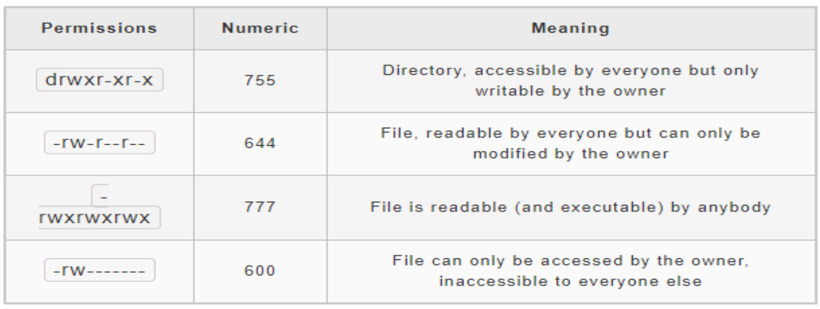
The meaning of the numeric values for file or directory permissions are mentioned below

- 4: Read permission (r)
- 2: Write permission (w)
- 1: Execute permission (x)

To represent a combination of permissions, you simply add up the numeric values:

- 0: No permissions (---)
- 1: Execute permission (--x)
- 2: Write permission (-w-)
- 3: Write and execute permissions (-wx)
- 4: Read permission (r--)
- 5: Read and execute permissions (r-x)
- 6: Read and write permissions (rw-)
- 7: Read, write, and execute permissions (rwx)
- For example, if you see a permission set as "750" for a directory, it means:

The owner has **read**, **write**, and **execute** permissions (4 + 2 + 1 = 7).
The group has **read** and **execute** permissions (4 + 1 = 5).
Others have no permissions (0).



Permissions	Numeric	Meaning
drwxr-xr-x	755	Directory, accessible by everyone but only writable by the owner
-rw-r--r--	644	File, readable by everyone but can only be modified by the owner
-rwxrwxrwx	777	File is readable (and executable) by anybody
-rw-------	600	File can only be accessed by the owner, inaccessible to everyone else

```
$chmod filename 700
```

Answer the following questions.

a) List the contents of a directory so that the files that have been modified most recently are displayed first.

b) List the contents of a directory so that the files that have been modified most recently are displayed last.

c) Change the current working directory to **/usr/bin**

d) Find information for the **cut** command (man grep, press **q** to quit).

e) Output the message **Welcome back to second semester!**

f) Create a new file named as **BDSP.txt** using nano or gedit editor.

g) Investigate how to create a new directory named as **BDSP**.

h) Investigate how to delete a directory.

i) Create a new file called **welcome.txt** containing some text data using **cat** command.

j) How you display the contents of the **welcome.txt** file normally and backwards?

k) How to write a shell script file and provides the executable privileges using **chmod** command?

l) Write a shell script to display the first 10 numbers in the forward and reverse order.

m) Write a shell script to display the even numbers from 0 to 10.

References:

https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

Ubuntu Linux Unleashed 2021 Edition, 14th Edition, Matthew Helmke, Addison-Wesley Professional, 2022.