Linux Terminal Guide

Table of Contents

How to open it

There's a few ways that you can accomplish that:

- Use the shortcut
- Open from the applications
- Open from the search bar

Using the shortcut

If you've opted to use this method, you're a badass 😎To do that, you just have to use this shortcut `ctrl + alt + t`. This will launch the terminal.

> If this seems a bit mouthful to you, you can always change your shortcut from settings. But this is the default in Ubuntu Linux.

From the apps

You can also open the terminal by going to all the applications and opening it from there. Opening the apps can done by pressing the supper key (the one with Windows' logo on it) and then the `a` key.

From the search bar

Press the super key and then search **terminal** in the search bar.

> There's no shortcut key set for the search bar by default in Ubuntu but you can always change or add it from the settings.

Where & who are you

By default, you'll be in your **root** directory. This is the **base** of all your directories/folders. You will see something like this:

```
~$ |
```

What `~` symbol means is that you are in the root folder. Don't worry if yours is different. The important part is that you will be in your root directory by default.

A way to see your location

If you want to see which folder you are currently in, you can use `pwd` command. This will print working directory. The syntax will look something like:

```
~$ pwd
```

As soon as you hit enter, you'll see:

```
~$ pwd
/home/najmiter
~$ |
```

In your case, it'll be your username, and not **najmiter** 😅

Who are you

If you forget who you are, you can ask the terminal. No jk!

There is a command that tells you your username. This is helpful when you've got more than one users on your OS. You can, then execute this command to know which user you're currently logged in as. That command is `whoami`. Here's a demo

```
~$ whoami
najmiter
~$ |
```

Changing directories

You're already familiar with Windows operating system. There's **local disks** and inside of those, there's folders. Those folders may further contain many folders and files.

These folders are implemented as **trees**. Each folder is a **node**. These nodes are **doubly linked**. Meaning: you may go forward or backward from any folder you're currently on.

Now don't get frustrated by it! You already know that stuff and for a quick revision, you can just open any of your local disk and start browsing the folders. You will come to an understanding that "*WOW! It really is just a tree!*".

And just like you can just wander around your directories in Windows, you may do that from the terminal as well. And to do that, we have the `cd` command. I'm sure you already know that it's a shortened version for **change directory**.

`cd` command

The syntax for it is really simple: you type the letters `cd` followed by the path. Hitting enter would take you to that directory.

> **Path** is the address of any file or folder. There's two kinds of paths; relative and absolute path. Relative path is relative to your **working directory** whereas the absolute path is all the way from your **root** directory also known as the **home directory**.

The argument to the `cd` command is optional. By default, it's you're root directory. Let's change directory to **Desktop** and learn from that!

```
~$ cd Desktop
```

After executing this command, you'll see:

```
~Desktop$ |
```

As you know, the default parameter for the `cd` command is the **root**. So, if you executed the `cd` without specifying the destination, it'll take you to the root. And it doesn't matter how many nodes (directories) deep you've gone, it'll still take you home (to the place, you belong 🎶)

```
~/Desktop$ cd
~$ |
```

Other ways to do the same are `cd ~/` , `cd ~` , and `cd /` . They all have the same effect but, simple `cd` is shorter 😄

Go to the previous folder

You can use `cd -` and this will take you to the folder that you were in. It's like clicking that **arrow back** button in **Windows Explorer**.

Scroll the cursor to top

Running the commands can cluster your terminal and your cursor is eventually going to reach the bottom of the screen and the entire screen is going to be messy with all the commands and their outputs that you shall have had done before.

You might try to scroll the terminal but it won't work!

The solution to that is a shortcut. Use `ctrl + t` and your cursor will reach the top.

`clear` command

You may have also come across this command. But There's a difference between using `ctrl + t` and the `clear` command.

Using `ctrl + t` will only scroll the cursor to the top of the screen. Whereas `clear` means to also remove all of the outputs (text) from the terminal.

Try both of them and then notice that the after the `ctrl + t`, you will be able to scroll up to the previous commands. Whereas, after `clear`, there will be nothing left to scroll up to.

Canceling an ongoing process

You may find yourself in the middle of something that you don't want to continue on anymore. You can cancel any ongoing process like a package being installed or a program running by pressing `ctrl + z`.

Cancel a command

Other times, you may also be in the middle of typing something on the terminal and then realize that it's not the right command or maybe you don't want to execute the said command. You can cancel that line by pressing `ctrl + c`.

Here's a demo:

```
~$ cd Desktop/Lab/Stuff
```

At this point, if you want to cancel this command, don't delete it by pressing `backspace` multiple times. Instead use `ctrl + c`. And you'll be brought to the next line and that command will be ignored.

```
~$ cd Desktop/Lab/Stuff^C
~$ |
```

Installing stuff

You're on a computer and there's no doubt that you'll be doing computer stuff on there. We usually need applications to do that. Installing softwares from the terminal is very easy in Linux. Let's install **g++** to learn this one:

To install anything, you will have to use the following command:

`sudo apt install $PACKAGE_NAME`

> Terminal has also the ability to make variables (see more later). To indicate a variable, we use the dollar sign `$` and then the name of the variable.

Everything before $PACKAGE_NAME will be the same. We don't have the variable called PACKAGE_NAME but we can simply write the name of the package directly.

```
~$ sudo apt install g++
```

Execute this command and it'll install the **g++** compiler.

> g++ is the compiler for C++ but it also brings the C compiler with it. Because C++ is derived from C language and we are well aware that object of the dervied class brings all the features of its base class 😄

What is `sudo`

The word `sudo` is shortened version of **superuser do**. You may know this guy as **The Administrator**. When you've got many users on Windows, you run the setup files as administrator because it has to be installed on the root directory.

We start our commands with this keyword because we want to give it the permissions to write data on the **root** directory. (For your understanding, you can relate the **root** to the **Local Disk C** of your Windows operating system)

What is `apt`

This stands for **Strong Packaging Tool**. The purpose of this guy is to help us in the actual installation of the package. It's also responsible for updating all the packages installed on the system when new updates are available.

> To update your packages, you can use `sudo apt update` command and it'll update (if any available) the file that containes all the information about what's installed on your system. Then use `sudo apt upgrade` and it'll install all the available updates for all the packages (softwares) installed on your computer.

Uninstalling stuff

To uninstall anything, use `sudo apt purge $PACKAGE_NAME` and like isntalling, it'll ask for your password. Then it'll also ask for confirmation. Enter `y` and it'll uninstall said package.

Opening files & stuff

You can open any file, folder, or even links from the terminal. The syntax is really simple. All you have to do is say `open $NAME`. This $NAME could be the name of a file, folder or even an online link.

This will open the specified target in its default application.

Viewing files of a directory

You can take a peek at what's inside a directory that you specify. To do that, all you need to do is say `ls`. This command will show you a short list of all the files and directories inside you working directory.

```
~$ ls
```

This will show the directories and files in your root directory. You can also indirectly specify a folder by giving the absolute or the relative path to it like `ls ~/Desktop` and it'll show the contents of the Desktop directory.

Using flags

You can also specify how you want that list to look like and how much information do want to see. You can use `ls -l` to show the long listing. What that means is that you'll see all the details of every item.

You can also use the `-a` flag to see the hidden files. You may also combine them and say `ls -al` and this will show all the deatils about all the files including the hidden files.

> To see more flags and what you can do, use `man ls` command and it'll show you the manual for `ls` command and how to use it.

Creating directories & files

Let's see how we can create our own folder from the terminal. To do that, you can use the `mkdir $FOLDER_NAME` command. The FOLDER_NAME can be anything you want. Upon execution, this will create a new folder with you specified name in your working directory.

Creating directories/folders

Let's create a new folder called **OS** in Desktop folder and learn how to use this command.

```
~$ cd Desktop
~Desktop$ mkdir OS
~Desktop$ |
```

Let's list all the files in our Desktop directory by using the `ls` command.

```
~Desktop$ ls
OS
~Desktop$ |
```

Creating files

Let's change directory to the **OS** folder that we just created.

```
~Desktop$ cd OS
~Desktop/OS$ |
```

In here, we're going to create a new file to write our C++ code. Let's create **main.cpp** from the terminal. To do that, use the `touch $FILE_NAME` command.

```
~Desktop/OS$ touch main.cpp
```

Upon listing the files of the working directory, you will see that your file has been created.

And one thing I'd like to emphasise here; I've shown you how you can create a `.cpp` file but that DOES NOT mean that's the only one you can do.

You can even create image files or videos even. `touch` has no limits. Touch everything! (That came out the wrong way but I hope you get it 😅)

Now you can use the `open` command like `open main.cpp` and it'll open it in the **text editor** (default). But you're already the **VIM** expert now so you know how to open it with that. Sky's the limit!

> You can also create multiple files in one go. For example, `touch main.cpp array.hpp array.cpp make.txt` will create all four of these files in the working directory.

Deleting files and directories

To delete a file, you can use the `rm $FILE_NAME` command if the file is also in the working directory. Otherwise you will have to provide the path to the file.

To delete a folder, use the same command but use the `-r` flag. If you the directory has children directories, it's going to give you a warning or an error. To overcome that, use the force `-f` flag.

```
~Desktop/OS$ rm main.cpp
```

If you execute this command, make sure to make this file again becuase we need it 😁 Let's create a dummy folder and then delete it.

```
~Desktop/OS$ mkdir dummy
~Desktop/OS$ rm -r dummy
~Desktop/OS$ |
```

If the **dummy** folder had more folders inside it, we'll instead use `rm -rf dummy`.

Writing to files

Let's a few different ways to write to a file from the terminal without actually opening the file.

`echo` command

You can use this command to basically write any value to the terminal and to files. Let's see some examples...

```
~Desktop/OS$ echo "anybody home?"
anybody home?
~Desktop/OS$ |
```

That was just the terminal. Let's now see how we can write to a file using this command.

```
~Desktop/OS$ echo "#include <iostream>" >> main.cpp
```

Let's break it down! The first keyword is the command itself. After that, it's the **string** that we want to write to the file. Then we have two **greate-than** symbols. They indicate where we want this string to be written. Then we have the name of the file, `main.cpp`.

There's another way to do this and that is to use only one **greate-than** symbol. The difference is that it will overwrite the entire file with the new string. Whereas if you use **>>**, it will **append** to the contents of the file.

`cat` command

Another and a better way to write to a file from the terminal is to use the `cat` command. Let's jump into it and see how to use it.

```
~Desktop/OS$ cat main.cpp
#include <iostream>
~Desktop/OS$ |
```

Now, don't get confused by it. It's **NOT reading**! the file. It's actually **writing** the contents of the specified file to the terminal.

Let's now **write** some code using `cat` command.

```
~Desktop/OS$ cat >> main.cpp
|
```

It'll keep taking the input until you press `ctrl + d` from your keyboard. At which point, it'll write whatever you'd written on the terminal to the file in **append** mode. It's the same for this just as was in `echo` command. If you use one **>**, it's going to overwrite the file with the new content but will keep the previous data if you use two **>>**.

```
~Desktop/OS$ cat >> main.cpp
#include <iostream>
int main() {
  std::cout << "Hello World\n";
} |
~Desktop/OS$ |
```

`>` command

You can also do the shorthand for the `cat` command like:

```
~Desktop/OS$ >> main.cpp
|
```

And of course, you can use the > version as well.

Reading from files

Reading files is as easy as writing to them. Let's go over a few ways of a reading file from the terminal.

`less` command

This command will show you the contents of the file in read mode. To jump out of there and back to the terminal, you have to press `q` key.

If the file has more content than there's space on your screen to display, it will give you a scroll. Use **arrow up**, **arrow down**, **page up**, or **page down** keys to scroll the text. And whenever you press the `q` key, it'll close the file and you'll be back to the terminal.

```
~Desktop/OS$ less main.cpp
```

As soon as you hit enter, you'll see your file's content:

```
#include <iostream>
int main() {
  std::cout << "Hello World\n";
}
main.cpp (END)
```

Press `q` and you'll be back into the terminal:

```
~Desktop/OS$ less main.cpp
~Desktop/OS$ |
```

`head` command

This shows you the first ten lines of your file by default.

```
~Desktop/OS$ head main.cpp
#include <iostream>
int main() {
  std::cout << "Hello World\n";
}
~Desktop/OS$ |
```

But you can specify the number of lines that you want to see by using the `-n` flag.

```
~Desktop/OS$ head -n 1 main.cpp
#include <iostream>
~Desktop/OS$ |
```

`tail` command

Same as the `head` command but it shows the last 10 lines of the file by default. You can also use the `-n` flag with this to specify the lines that you want to see of your choice.

Copying & moving files

This is a little bit tricky. Not really but sometimes, especially for beginners, it can be! But we're here to make the tricky stuff lemon-squeasy stuff, right 😄

Copying a file

Before I tell you how it's done, I want you to imagine a scenario with me. Imagine your mother/sister or anyone at home asks you to put the dress in the almirah. That almirah is in some room. So, you will go to that room. But then you won't just put it there (if you're a good kid that is) but instead you will go the almirah and then put it there.

The room is like a folder or directory. Just like a room has things inside it, a folder can have things (files) in it too. Putting some content into that folder doesn't mean just throw it in there randomly. Sure, we will go that folder but then we also have to find the right spot (the file).

This confuses lots of beginners so I thought we should clear it up beforehand. Now let's talk business 😎

The `cp $SOURCE_FILE $DEST_FILE` command can be used to copy a file. Let's copy our `main.cpp` file to **Downloads** folder and learn along the way.

```
~Desktop/OS$ cp main.cpp ~/Downloads/main_ki_copy.cpp
~Desktop/OS$ |
```

Let's go to the Downloads folder by typing `cd ~/Downloads`. Of course, you could first go to the root and then go the Downloads folder but this is the same.

```
~Downloads$ ls
main_ki_copy.cpp
~Downloads$ |
```

Assuming you didn't have anything in your Downloads folder, you should also see this output. But, at least, among other files/directories, you will see your `main_ki_copy.cpp` as well.

Here, we first went to the room (Downloads folder) and then specified where (main_ki_copy.cpp) we wanted to put our dress (code / text / content). I hope it's clear now 🤩

> This does not at all mean that you can't copy or move without specifying the name of the destination. If you don't specify the name for the file, it will keep the original name. The point of the story was to help you understand that data is not just randomly thrown into folder.

Moving a file

This one is the same as copying except for a few concepts. As you know, copying a file keeps the original file at its place while making a new file at the destination. Whereas **moving** means to move the file itself to the destination.

The `mv $SOURCE_FILE $DEST_FILE` command can be used to move a file. Let's move the `main_ki_copy.cpp` back the OS folder.

```
~Downloads$ mv main_ki_copy.cpp ~/Desktop/OS/kahan_se_i_hai.cpp
~Downloads$ |
```

Notice that, this will move the file to it's specified destination but it will also **rename** the file becuase this time, I use a different name. No doubt, you could've used the same name but I wanted to show you something and that is that you can also use the `mv` command to rename a file or a directory.

Let's rename the **OS** folder to **Politics Class**. And to keep our command short and clean, let's also `cd` to Desktop because that's where the subject folder is located.

```
~Downloads$ cd ~/Desktop
~Desktop$ mv OS "Politics Class"
~Desktop$ |
```

Notice that I've put the name in quotes because there's a space in the name. Make sure to put the quotes if your chosen name has empty spaces. Anyway, now if you `ls`, you'll notice that your folder has now been changed to **Politics Class**.

Universal selector

You have learned a few commands up until now. And you saw how you can target one file or directory at a time. But you can also apply filter (not the Snapchat filter) on your targets.

Let's imagine you had many `cpp` files and you wanted to, maybe, copy them, or delete them, or move them or whatever. Instead of doing it to each one of them, you can use the **universal selector**.

Let's create some files in the Desktop folder and then move them to **Politics Class** folder at once.

```
~Desktop$ touch array.cpp vector.cpp tensor.cpp network.cpp
~Desktop$ mv *.cpp ./Politics Class
~Desktop$ |
```

The `*` is the **universal selector** and it means that any file that ends with **.cpp**, put it on my plate. We got some business to take care! So it will select all of those files and do whatever told.

You can also use this selector when removing file. Using `rm *.cpp` will remove all the `cpp` files from the working directory.
