

19 Absolute Simple Things About Linux Terminal Every Ubuntu User Should Know - It's FOSS

You are here: [Home](#) / [Terminal Tools and Tricks](#) / 19 Absolute Simple Things About Linux Terminal Every Ubuntu User Should Know

Terminal often intimidates new users. However, once you get to know it, you gradually start liking it. Well, that happens with most Linux users.

Even if you are using Ubuntu as a desktop system, you may have to enter the terminal at times. New users are often clueless about many things. Some knowledge of basic Linux commands always helps in such cases but this article is not about that.

This article focuses on explaining small, basic and often ignored things about using the terminal. This should help new Ubuntu desktop users to know the terminal and use it with slightly more efficiency.

The terminal you see is just one of the [various terminal applications](#) available. After all terminal is just a GUI tool that gives you access to a shell where you can run the commands.

Different terminal applications (properly called terminal emulators) look different, have slightly different functions and features (like different keyboard shortcuts, color combination, fonts etc).

This article is specifically focused on the default Ubuntu terminal which is an implementation of the GNOME terminal.

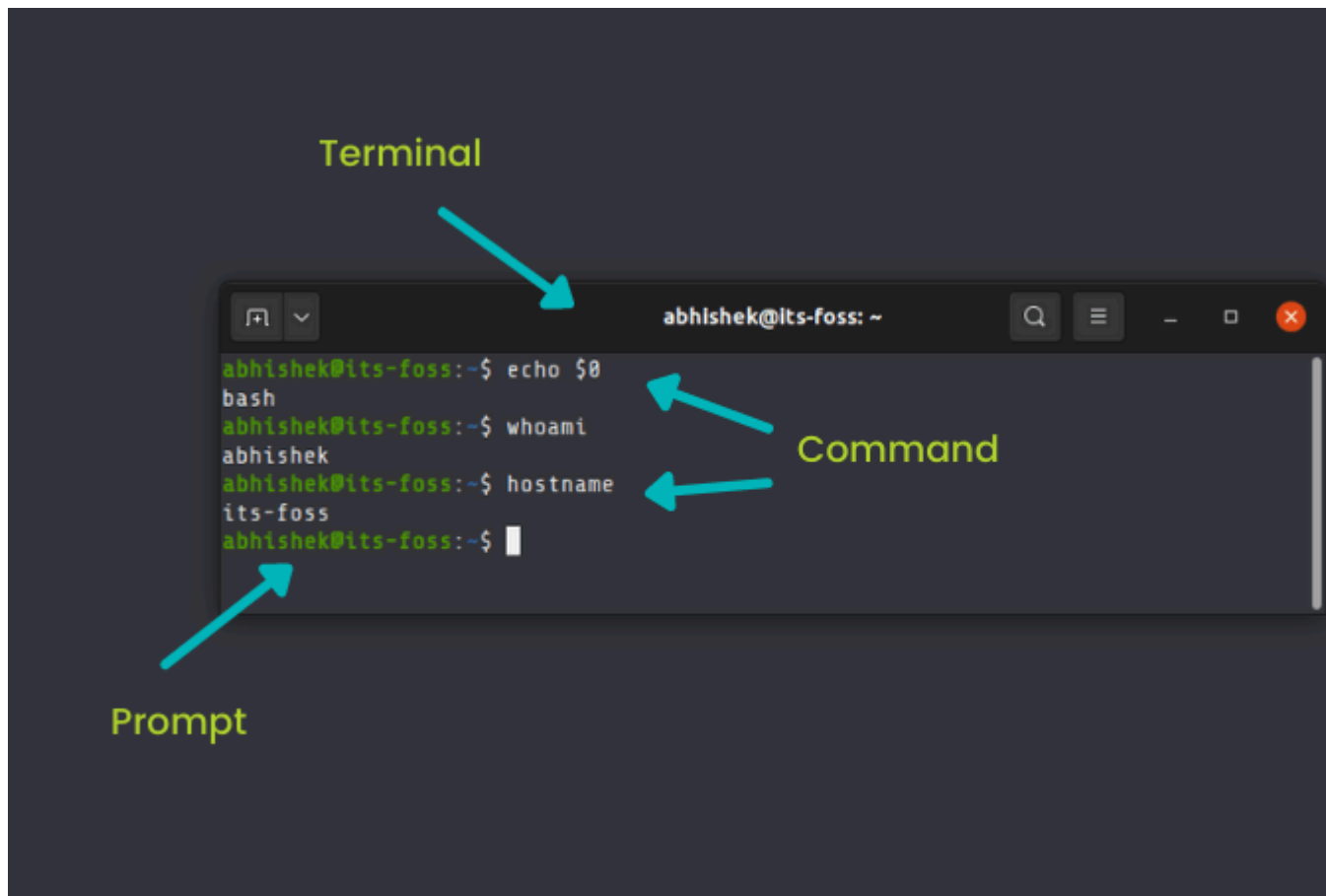
1. Open the terminal with keyboard shortcut

You can [open the terminal in Ubuntu](#) by looking for it in the system menu. However, my favorite way is to use the Ctrl+Alt+T [keyboard shortcut in Ubuntu](#).

Ctrl+Alt+T

2. Terminal vs shell vs prompt vs command line

Before you see anything else, you should know the difference between different terminologies that are often (incorrectly) used interchangeably.



Terminal is the graphical application that runs a shell by default.

Shell is difficult to visualize separately from the terminal. The terminal is running a shell, usually Bash shell by default in Ubuntu. Like terminals, there are various shells as well. Bash is the most popular of them all and default shell on most Linux distributions.

The commands you type are interpreted by the shell. Often people think that screen they see in the terminal is the shell. That's fine for understanding.

Prompt is what you see before the space where you type the commands. There is no set standard for the prompt. In some old terminals, you would just have a blinking cursor to the place where you can type the commands. In Ubuntu terminal, prompt gives you some information which you'll see in detail in the later sections of this article.

Command line is not something specific to Linux. Every operating system has a command line interface. Many programming languages have command line interface. It's a term used for the interface where you can run and execute commands.

3. Understanding the prompt

You know it by now. What you see before the space where you type the command is called prompt. It is configurable and looks different in different distributions, terminal applications and shells.

Ubuntu terminal has configured the prompt to show you a few things. You can get the following information at a glance:

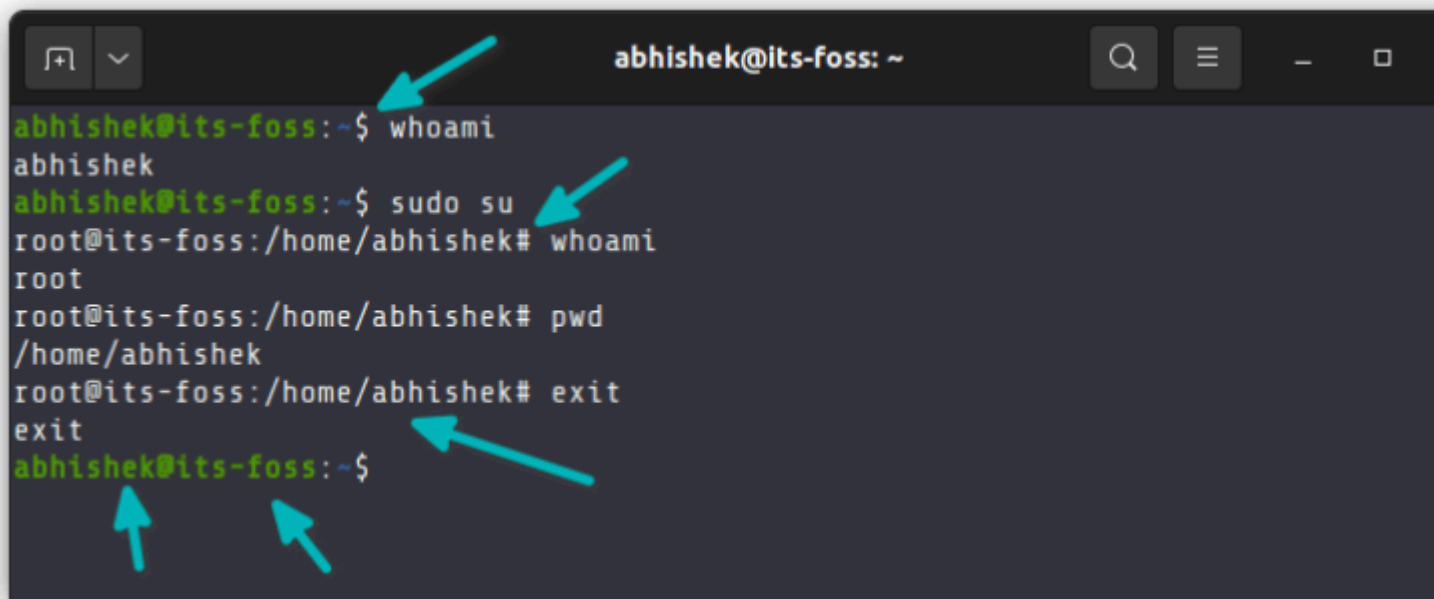
- User name
- Hostname (name of the computer)
- Current working directory

A few more things you may wonder about.

The colon (:) in the prompt is a separator to distinguish between hostname and the current location.

Tilde (~) means the home directory of the present user.

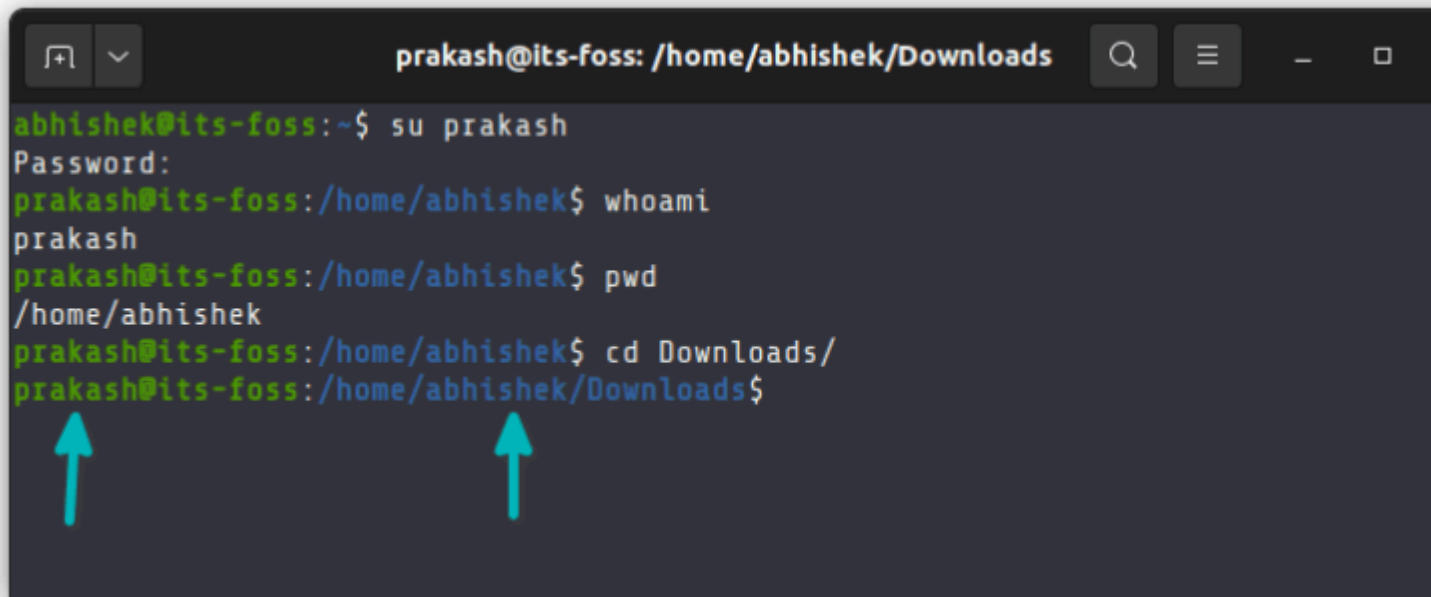
For normal users, the prompt ends with dollar (\$) symbol. For the root user, it ends with pound or hash (#) symbol. And hence the joke that pound is stronger than dollar.

A terminal window titled 'abhishek@its-foss: ~' with standard Ubuntu window controls. The terminal shows a sequence of commands and prompts. Red arrows point to specific parts of the prompt: one to the username 'abhishek' in the first prompt, one to the hostname 'its-foss' in the second prompt, one to the root prompt 'root@its-foss: /home/abhishek#', and two to the 'abhishek@its-foss: ~\$' prompt at the bottom. The commands executed are 'whoami', 'sudo su', 'whoami', 'pwd', and 'exit'.

```
abhishek@its-foss:~$ whoami
abhishek
abhishek@its-foss:~$ sudo su
root@its-foss:/home/abhishek# whoami
root
root@its-foss:/home/abhishek# pwd
/home/abhishek
root@its-foss:/home/abhishek# exit
exit
abhishek@its-foss:~$
```

Did you notice that when I switched to the root user, the command prompt looked different without any colors? This is another reminder that prompt is not a standard and is configured explicitly. For normal users, Ubuntu has a different configuration of the prompt than the root.

Simple information like this helps indirectly. In a multi-user environment, you can easily figure out which user you are using right now and if it is a root user. The displayed location is also helpful.

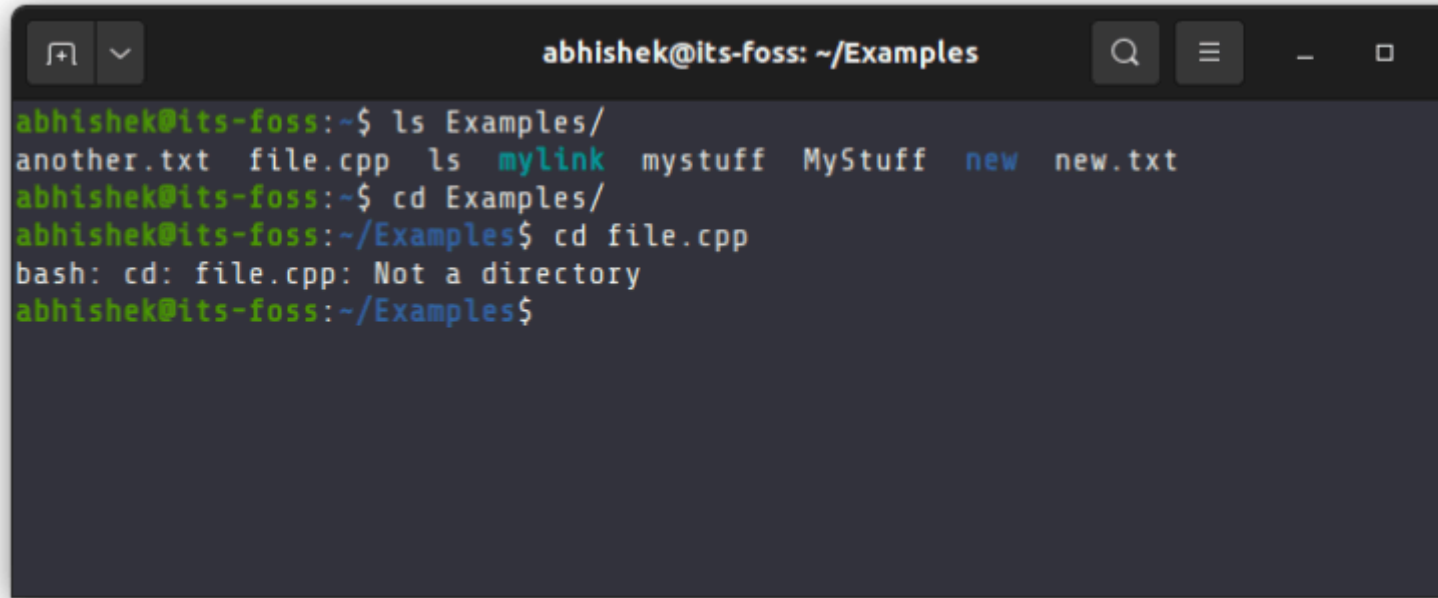
A terminal window with a dark background. The title bar shows 'prakash@its-foss: /home/abhishek/Downloads'. The terminal content shows a sequence of commands: 'su prakash' (prompt changes to 'prakash@its-foss:~\$'), 'whoami' (output: 'prakash'), 'pwd' (output: '/home/abhishek'), and 'cd Downloads/' (prompt changes to 'prakash@its-foss:/home/abhishek/Downloads\$'). Two red arrows point upwards to the first and second parts of the final prompt: 'prakash@its-foss' and '/home/abhishek/Downloads\$'.

4. Directory and files

Two terms you hear the most in Linux are directory and files.

You probably know what a file is but you may get confused with the term 'directory'. Directory is nothing but folder. It keeps files and folders inside it.

You can go inside the directories but you cannot enter files. You can read files of course.

A terminal window with a dark background. The title bar shows 'abhishek@its-foss: ~/Examples'. The terminal content shows a series of commands and their outputs: 'ls Examples/' lists files including 'mylink' and 'new.txt'; 'cd Examples/' changes the directory; 'cd file.cpp' results in an error 'bash: cd: file.cpp: Not a directory'.

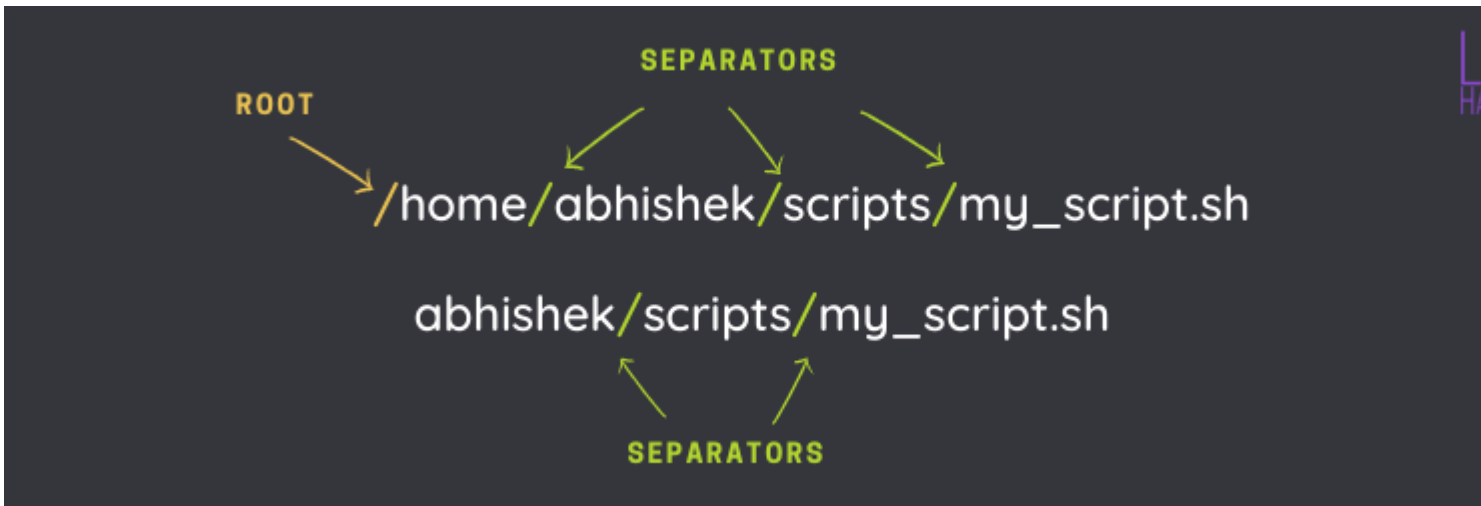
You can use the term 'folder' for directory and it should be fine. However, it is better to use 'directory' because this is what you'll see referenced in various tutorials, documents etc. You'll even find commands like `rmdir`, `mkdir` hinting that they deal with directories.

Additional note: Everything is a file in Linux. Even directory is a special kind of file that has the memory address of files and directories inside it. I have explained it in my [article on hard links](#). You may refer to that if you want to know more on this topic.

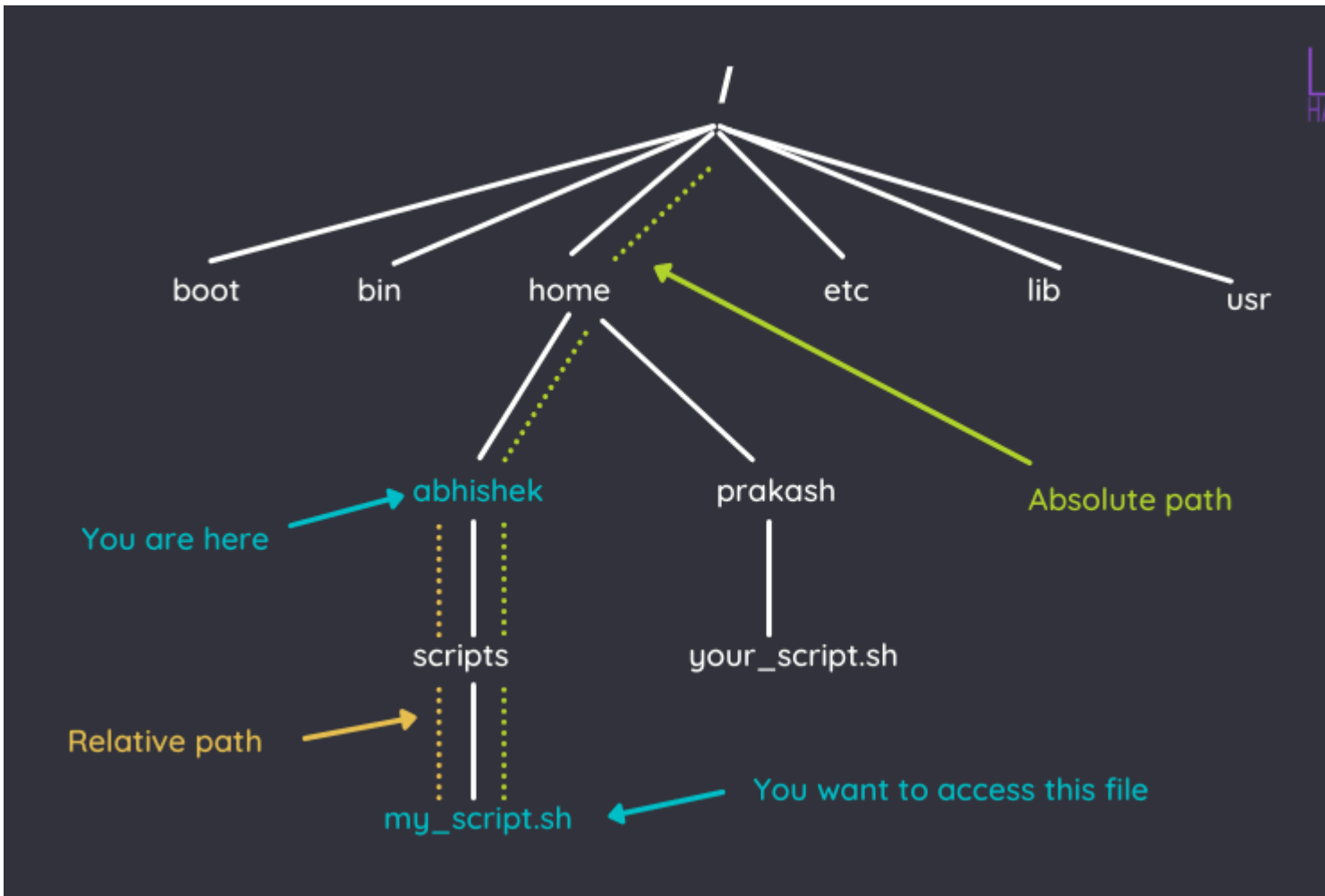
5. Path: Absolute and relative

The [directory structure in Linux resembles](#) the root of a tree. Everything starts at root and spreads out from there.

If you have to access a file or directory, you need to tell how to reach its location by providing its 'path'. This path that is composed of directory names and separators (`/`). If a path starts with `/` (i.e., root), it is an absolute path otherwise it is a relative path.



The absolute path starts from the root and can be easily referenced from anywhere in the system. The relative path depends upon your current location in the directory structure.



If you are in the location `/home/abhishek` which has a directory named `scripts` containing a file `my_script.sh` and you want the path for this file, its absolute path will be:

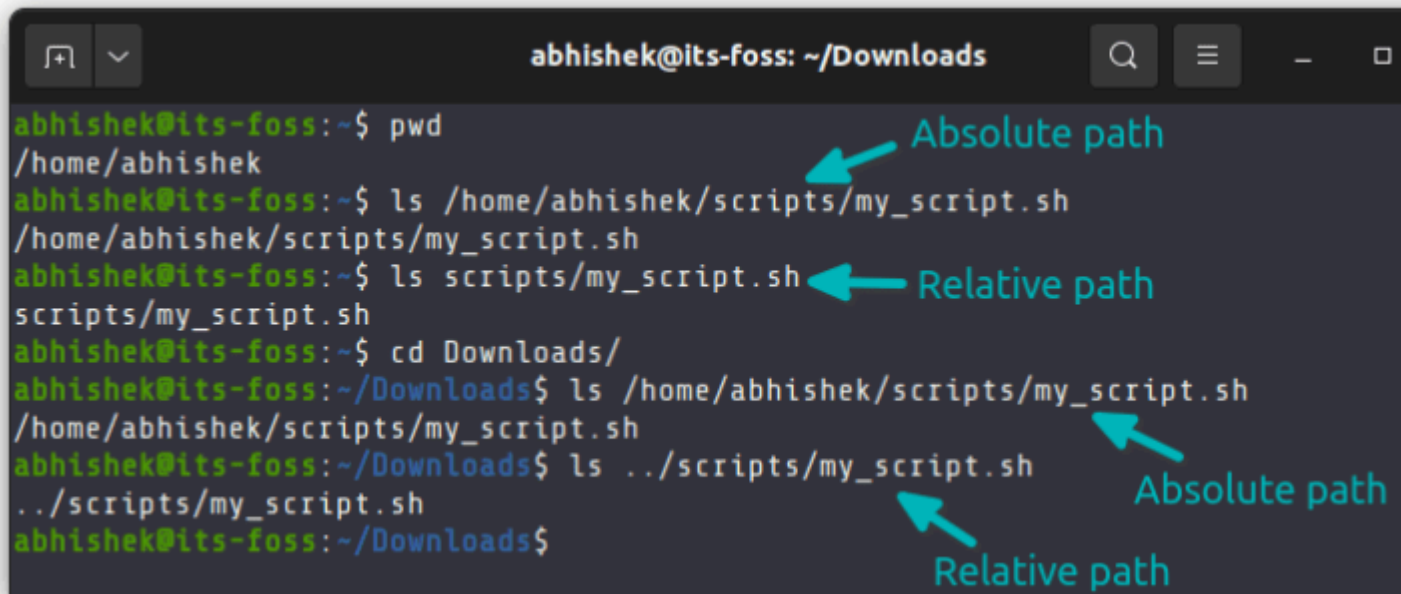
/home/abhishek/scripts/my_script.sh

Its relative path will be:

scripts/my_script.sh

If you change the location, the absolute path of the file remains the same. However, the relative path changes because it is relative to your current path.

Image: Show relative path and absolute path by changing the location



The image shows a terminal window with the title 'abhishek@its-foss: ~/Downloads'. The terminal output is as follows:

```
abhishek@its-foss:~$ pwd
/home/abhishek
abhishek@its-foss:~$ ls /home/abhishek/scripts/my_script.sh
/home/abhishek/scripts/my_script.sh
abhishek@its-foss:~$ ls scripts/my_script.sh
scripts/my_script.sh
abhishek@its-foss:~$ cd Downloads/
abhishek@its-foss:~/Downloads$ ls /home/abhishek/scripts/my_script.sh
/home/abhishek/scripts/my_script.sh
abhishek@its-foss:~/Downloads$ ls ../scripts/my_script.sh
../scripts/my_script.sh
abhishek@its-foss:~/Downloads$
```

Annotations in the image:

- A red arrow points to the path `/home/abhishek/scripts/my_script.sh` in the second command, labeled "Absolute path".
- A red arrow points to the path `scripts/my_script.sh` in the third command, labeled "Relative path".
- A red arrow points to the path `/home/abhishek/scripts/my_script.sh` in the fifth command, labeled "Absolute path".
- A red arrow points to the path `../scripts/my_script.sh` in the sixth command, labeled "Relative path".

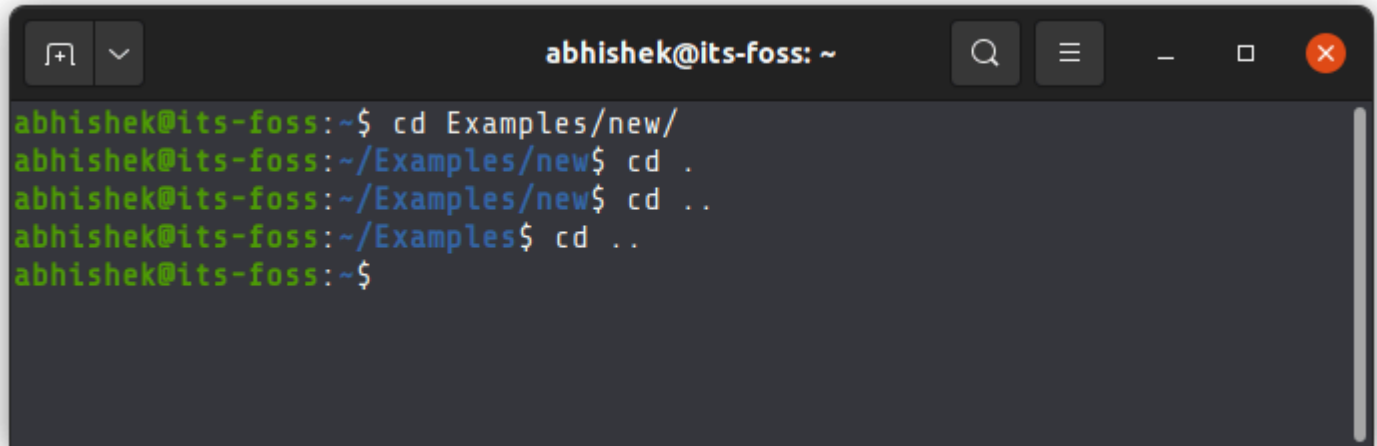
6. . and ..

You may often come across . and .. notation while using the Linux terminal.

Single dot (.) means the current directory.

Double dots (..) mean the parent directory (one directory above the current location).

You'll often use the double dot (..) in relative path or for changing directory. Single dot (.) is also used in relative path but more importantly, you can use it in the commands for specifying the current locations.

A terminal window titled 'abhishek@its-foss: ~' with standard window controls. The terminal shows a sequence of 'cd' commands navigating through a directory structure: from the home directory to 'Examples/new/', then to '.', then to '..', and finally back to '..' from the 'Examples' directory, returning to the home directory.

```
abhishek@its-foss:~$ cd Examples/new/  
abhishek@its-foss:~/Examples/new$ cd .  
abhishek@its-foss:~/Examples/new$ cd ..  
abhishek@its-foss:~/Examples$ cd ..  
abhishek@its-foss:~$
```

7. Understand the command structure

A typical Linux command consists of a command name followed by options and arguments.

command [options] argument

Option, as the name suggests, are optional. When used, they might change the output based on their properties.

For example, the cat command is used for viewing files. You can add the option -n and it will display line numbers as well.

Options are not standardized. Usually, they are used as single letter with single dash (-). They may also have two dashes (--) and a word.

Same options may have different meaning in a different command. If you use -n with head command, you specify the number of lines you want to see, not the lines with numbers.


```
abhishek@its-foss: ~  
abhishek@its-foss:~$ cat sample.txt  
This is a sample text  
Created by Abhishek  
For It's FOSS  
To explain various Linux concepts  
In a simple manner  
abhishek@its-foss:~$ cat -n sample.txt  
  1 This is a sample text  
  2 Created by Abhishek  
  3 For It's FOSS  
  4 To explain various Linux concepts  
  5 In a simple manner  
abhishek@its-foss:~$ head -n 3 sample.txt  
This is a sample text  
Created by Abhishek  
For It's FOSS  
abhishek@its-foss:~$
```

In command documentations, *if you see something between brackets ([])*, it indicates that the contents of the bracket are optional.

Similarly, arguments are also not standardized. Some commands expect filenames as argument and some may expect directory name or a regular expression.

8. Getting help

As you start using commands, you may remember some of the options of frequently used commands but it is simply not possible for you to remember all the options of any command.

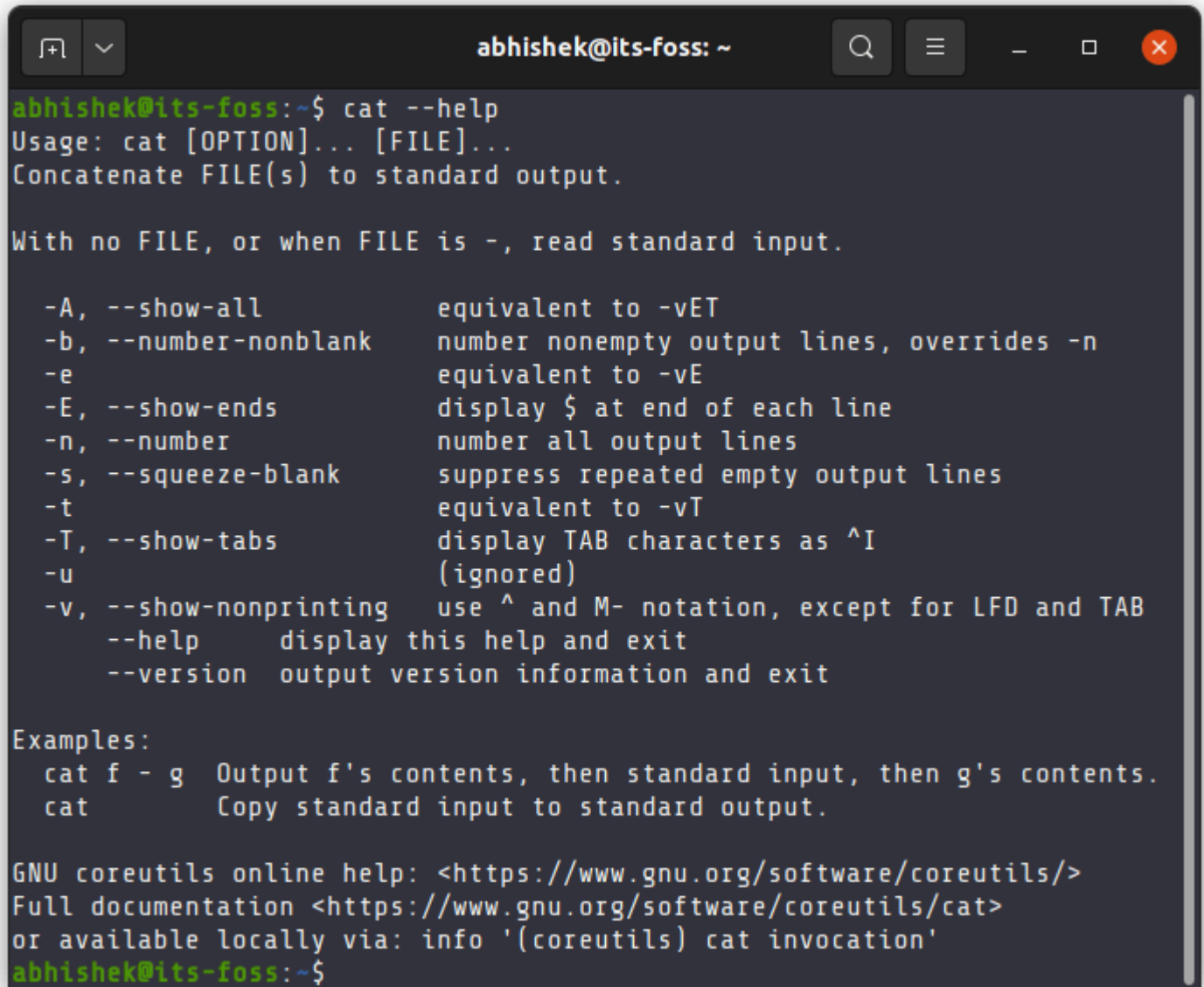
Why? Because a single command may have more than ten or twenty options.

So, what do you do when you cannot recall all the options? You take help. And with help, I do not mean asking a question in It's FOSS [Linux forum](#). I ask to use the help option of the command.

Every standard Linux command has a quick help page that can be accessed with -h or --help or both.

command_name -h

It gives you a quick glimpse of the command syntax, common options with their meaning and in some cases, command examples.

A terminal window titled 'abhishek@its-foss: ~' with standard window controls. The terminal shows the command 'cat --help' and its output. The output includes the usage, a description of concatenating files, a list of options with their descriptions, and examples of how to use the command. The prompt 'abhishek@its-foss:~\$' is visible at the bottom.

```
abhishek@its-foss:~$ cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s) to standard output.

With no FILE, or when FILE is -, read standard input.

-A, --show-all           equivalent to -vET
-b, --number-nonblank     number nonempty output lines, overrides -n
-e                        equivalent to -vE
-E, --show-ends           display $ at end of each line
-n, --number              number all output lines
-s, --squeeze-blank       suppress repeated empty output lines
-t                        equivalent to -vT
-T, --show-tabs           display TAB characters as ^I
-u                        (ignored)
-v, --show-nonprinting    use ^ and M- notation, except for LFD and TAB
--help                   display this help and exit
--version                 output version information and exit

Examples:
  cat f - g  Output f's contents, then standard input, then g's contents.
  cat        Copy standard input to standard output.

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/cat>
or available locally via: info '(coreutils) cat invocation'
abhishek@its-foss:~$
```

If you need more help, you can refer to the [manpage](#) i.e. manual of a command:

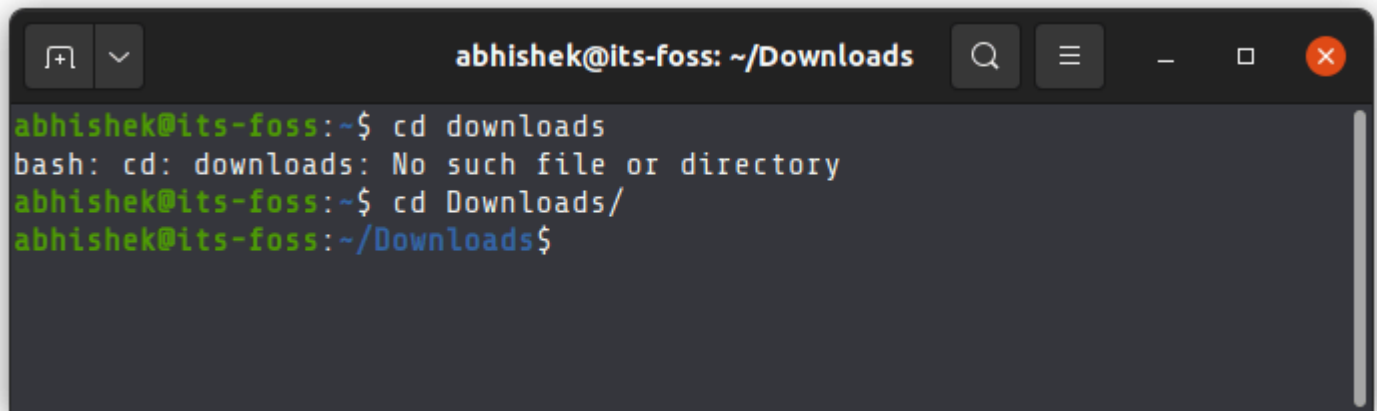
`man command_name`

It goes in all of details and could be overwhelming to read and understand. Alternatively, you can always search on the internet for 'examples of xyz commands in Linux'.

9. Linux is case-sensitive

Linux is case-sensitive. Everything you type in the terminal is case-sensitive. If you do not take that into account, you'll often run into [bash: command not found](#) or file not found errors.

In the home directory, you have all the folders name starting with the upper case. If you have to switch to the Documents directory, you have to keep the first letter as D and not d. Otherwise, the terminal will complain.

A terminal window titled 'abhishek@its-foss: ~/Downloads' with standard window controls. The terminal shows the following commands and output:

```
abhishek@its-foss:~$ cd downloads
bash: cd: downloads: No such file or directory
abhishek@its-foss:~$ cd Downloads/
abhishek@its-foss:~/Downloads$
```

You can have two separate files named file.txt and File.txt because for Linux, file and File are not the same.

10. Running shell scripts

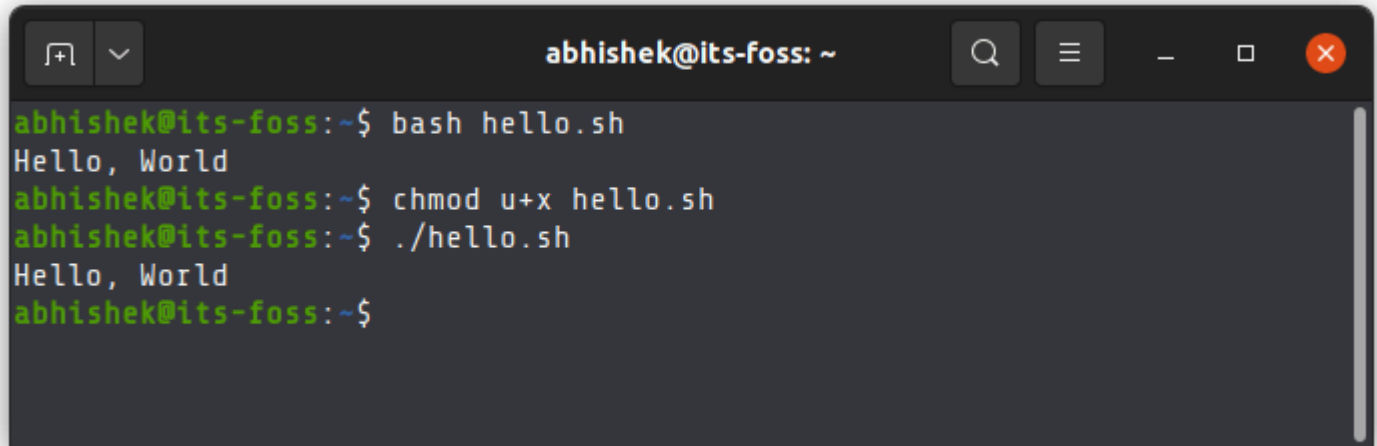
You can [run a shell script](#) by specifying the shell:

```
bash script.sh
```

Or you can execute the shell script like this:

```
./script.sh
```

The second one will only work when the file has execute permission. More on [Linux file permission here](#).

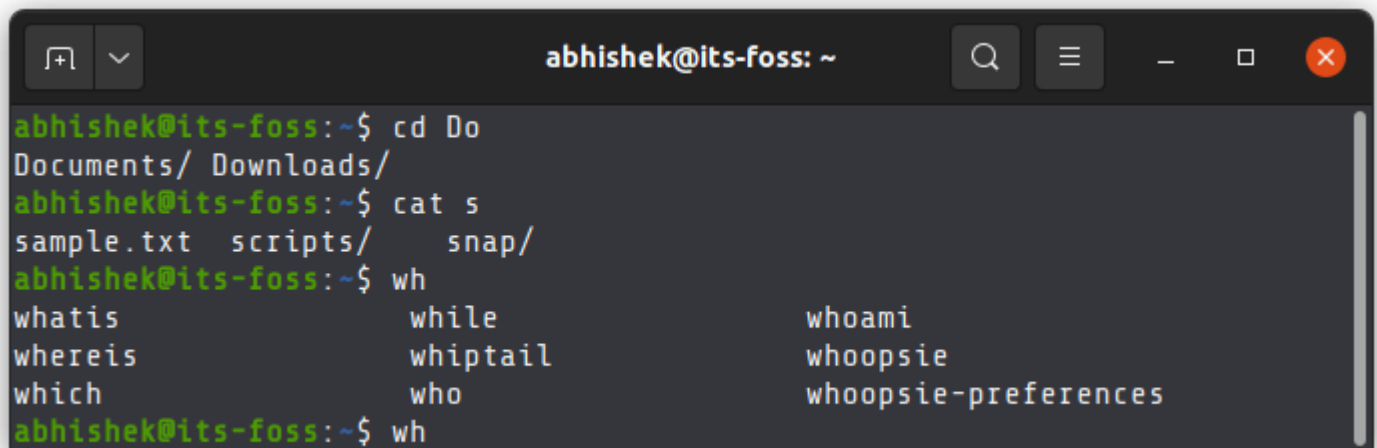
A terminal window titled 'abhishek@its-foss: ~' with standard window controls. The terminal shows a user running 'bash hello.sh', which outputs 'Hello, World'. Then, the user runs 'chmod u+x hello.sh' and './hello.sh', which also outputs 'Hello, World'.

```
abhishek@its-foss:~$ bash hello.sh
Hello, World
abhishek@its-foss:~$ chmod u+x hello.sh
abhishek@its-foss:~$ ./hello.sh
Hello, World
abhishek@its-foss:~$
```

11. Use tab completion instead of typing it all

The Ubuntu terminal is pre-configured with tab completion. This means that if you start writing something in the terminal and then hit tab, it tries to automatically complete it or provide options if there are more than one possible matches.

It works for both commands as well arguments and filenames.

A terminal window titled 'abhishek@its-foss: ~' showing tab completion in action. Typing 'cd Do' followed by a tab key results in 'Documents/' and 'Downloads/'. Typing 'cat s' followed by a tab key results in a list of files: 'sample.txt', 'scripts/', and 'snap/'. Typing 'wh' followed by a tab key results in a list of commands: 'whatis', 'while', 'whoami', 'whereis', 'whiptail', 'whoopsie', 'which', 'who', and 'whoopsie-preferences'.

```
abhishek@its-foss:~$ cd Do
Documents/ Downloads/
abhishek@its-foss:~$ cat s
sample.txt  scripts/    snap/
abhishek@its-foss:~$ wh
whatis      while       whoami
whereis     whiptail   whoopsie
which       who        whoopsie-preferences
abhishek@its-foss:~$ wh
```

This saves a great ton of time because you don't have to write everything completely.

12. Ctrl+C and Ctrl+V is not for copy pasting in terminal

Ctrl+C, Ctrl+V might be the 'universal' keyboard shortcuts for copy paste but it doesn't work in the Linux terminal.

Linux inherits a lot of stuff from UNIX and in UNIX, Ctrl+C was used for stopping a running process.

Since the Ctrl+C was already taken for stopping a command or process, it cannot be used for copy-paste anymore.

13. You can surely copy paste in the terminal

Don't worry. You can still [copy paste in the terminal](#). Again, there is no fixed rule for the copy-paste keyboard shortcuts because it depends on the terminal application you are using or the configuration you have in place.

In Ubuntu terminal, the default keyboard shortcut for copy is Ctrl+Shift+C and for paste, it is Ctrl+Shift+V.

You can use Ctrl+C to copy text and commands from outside the terminal (like a web browser) and paste it using Ctrl+Shift+V. Similarly, you can highlight the text and use Ctrl+Shift+C to copy the text from the terminal and paste it to an editor or other applications using Ctrl+V.

14. Avoid using Ctrl+S in the terminal

Another common mistake beginners do is to use the 'universal' Ctrl+S keyboard shortcut for save. If you use Ctrl+S in the terminal, your terminal 'freezes'.

This is coming from the legacy computing where there was no scope of scrolling back up. Hence, if there were lots of output lines, Ctrl+S was used for stopping the screen so that text on the screen could be read.

You can unfreeze your terminal with Ctrl+Q. But again, avoid using Ctrl+S in the terminal.

15. Pay attention to \$ and <> in command examples

If you are referring to some online tutorial or documentation, you'll see some command examples with text inside <>. This indicates that you need to replace the content along with < and > with a suitable value.

For example, if you see a command like this:

```
grep -i <search_term> <file_name>
```

You should replace the <search_term> and <file_name> with the respective actual values.

It is an indication that the command is just an example and you have to complete it with actual values.

Another thing to note here is that some tutorials show command examples that start with \$ like this:

Next, install the public key using ssh-copy-id command:

```
$ ssh-copy-id -i /path/to/public-key-file user@host  
$ ssh-copy-id user@remote-server-ip-or-dns-name  
$ ssh-copy-id vivek@rhel7-aws-server
```

Not part of command, avoid copying it with the commands

This is a way for them to indicate that it is command (not command output). But many new Linux users copy the preceding \$ along with the actual command and when they paste it in the terminal, it throws error obviously.

So, when you are copying some command, don't copy the \$ if it is there at the beginning. You should also avoid copying random commands for random websites specially when you do not understand what it does.

Since you are reading about copying commands, when you see commands in multiple lines together, you should copy one line at a time and run them on by one:

```
sudo apt update  
sudo apt install wget
```

Copy the commands one by one,
not together

The next step now is to download the GPG key of AnyDesk repository and add it to your system's trusted keys. This way, your system will trust the software coming from this [ext](#)

The next section tells you how to run multiple commands in one go.

16. You can run multiple commands at once

You can [run multiple commands at once](#) without user intervention. You might have seen it already as an Ubuntu user in the form of this command:

```
sudo apt update && sudo apt upgrade
```

There are three different ways to combine commands in the terminal:

- ;
Command 1 ; Command 2 Run command 1 first and then command 2
- &&
Command 1 && Command 2 Run command 2 only if command 1 ends
2 successfully
- ||
Command 1 || Command 2 Run command 2 only if command 1 fails

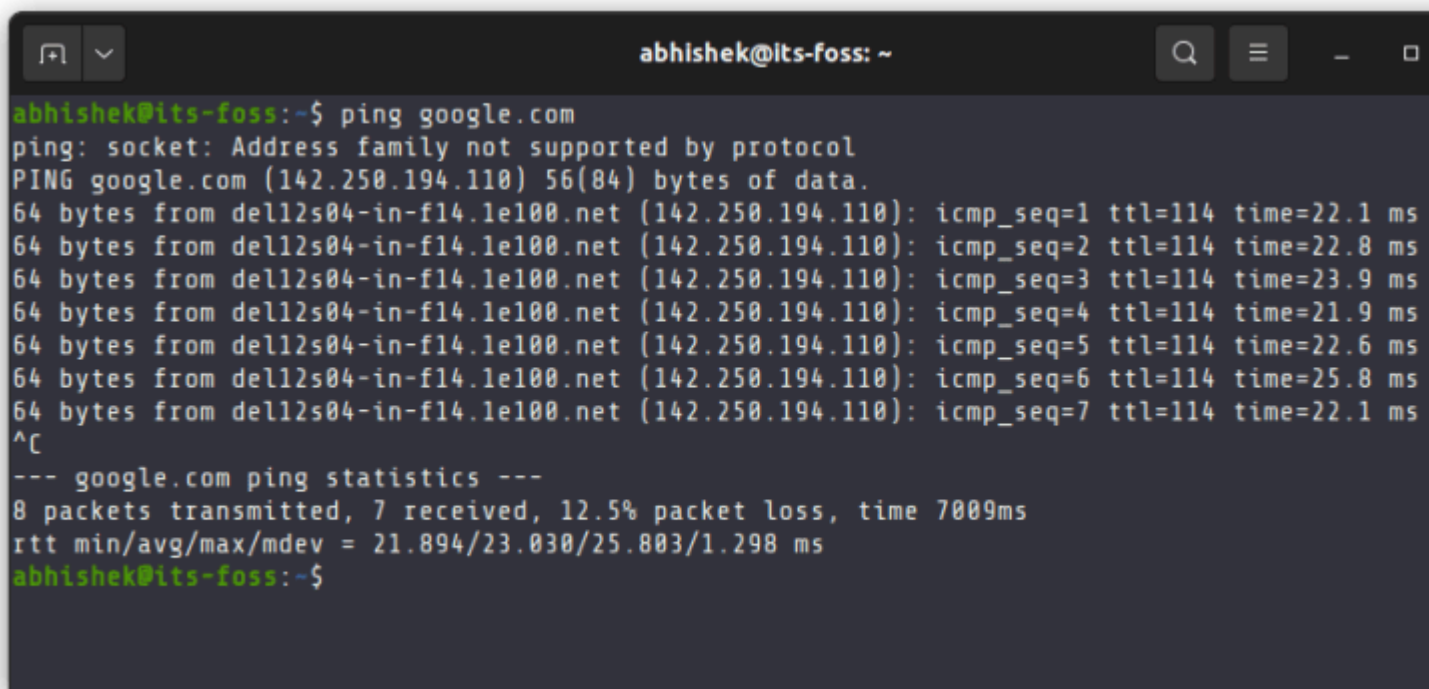
17. Stop a running Linux command

If a Linux command is running in the foreground, i.e., it is showing output or you cannot enter any other command, you can stop it using the Ctrl+C keys.

I discussed it previously. It comes from the legacy computing days of UNIX.

So, the next time you see a command like top or ping running continuously and you want the terminal control back, just use these two keys:

Ctrl+C

A terminal window titled 'abhishek@lts-foss: ~' showing a ping command to google.com. The output shows 7 successful pings with varying times. The user presses Ctrl+C, indicated by '^C' on the line. The terminal then displays '--- google.com ping statistics ---' followed by summary statistics: '8 packets transmitted, 7 received, 12.5% packet loss, time 7009ms' and 'rtt min/avg/max/mdev = 21.894/23.030/25.803/1.298 ms'. The prompt returns to 'abhishek@lts-foss:~\$'.

18. Clear the terminal

When I find that my screen is too cluttered with different kind of output, I clear the terminal screen before starting some other work. It just a habit but I find it helpful.

To clear the terminal, use the command

clear

You may also use Ctrl+L [terminal shortcut](#).

19. Exiting the terminal

In a few cases, I have seen people closing the terminal application to exit the session. You could do that but the proper way to exit a terminal is to use the exit command:

```
exit
```

You may also use the keyboard shortcut Ctrl+D for Ubuntu terminal.

Conclusion

There are so many additional stuff you can do in the terminal even if you are new to the entire terminal thing. You can:

- [Run funny Linux commands](#)
- [Browse the internet in terminal](#)
- [Play games in terminal](#)

And if you are looking for more, [take a look at these Linux command tips and use the terminal like a pro](#).

Honestly speaking, there is way too much to talk about. It is difficult to determine what should be considered as absolute basics and what should be left out. For example, I wanted to avoid including the information about paths because it needs detailed explanation but going too much in detail on a single could be overwhelming.

I have passed the stage where small things used to baffle me in the terminal. If you are new to the Linux terminal or if you remember the struggle from your initial Linux days, feel free to suggest any additions to the list. I might update the list with your input.

And if you learned something new, please do mention it in the comments. I would like to see if this article was worth the effort :)