Linux Command Line Basics, Examples, Cheatsheats & Books

SeventhQueen

Linux command line is very famous for its extreme flexibility in doing tasks. One or two commands are enough to do wonders on your system.

Such feature barely exists in other families of operating systems like Windows (PowerShell is very limited in terms of usage), which is why you must learn how to use it if are going to use Linux.

Let's start by introducing you to the topic.

Linux CLI Historical Introduction

Let's start first by introducing you to the term "**Unix Shell**". A shell is a command line interpreter that provides the user with the ability to manage the system using commands he enters. You can think of it as some sort of communication channel between you and the operating system. It's called a *shell* because it sits around the operating system's kernel and covers it just like a crab would take cover by his own shell, hence the name.

UNIX was an operating system released back at 1970s by a company called AT&T. After few years from its release, Unix no longer existed as a single operating system but became a family of different operating systems, each which do follow the Unix specifications and design and philosophy. By time, some other operating systems came to the light as "Unix-like"; which are the operating systems that do follow the Unix family design and are very similar to the Unix specifications and standards, one of them is Linux, which is a Unix-like OS.

In it's design, Unix had what's known as a Unix shell, which is the language the user writes into the console in order to make the system operate in various ways. You can enter these commands directly in the command line prompt, or save them into a file and execute them later, the latter is known as "**shell scripting**".

There are many various Unix shells out there, not just one. Each has its own features and disadvantages. The most famous shell however is called "Bash", and is almost installed by default in all Linux distributions, which is the main topic of our article. When somebody refers to the Linux command line, they are probably referring on how to write Bash commands and Bash shell scripts.

Why Learn Linux Command Line?

Because it's extremely practical and useful. Using Linux command line, you'll be able to do very powerful stuff in a short amount of time on Linux. While most people prefer graphical user interfaces, still, in most of the times, practical and quick user interfaces for everything may not exist. Hence, you'll need the Linux command line to do a lot of tasks.

You can automate many stuff in your life using the command line, like downloading/taking a backup of a set of files you need, getting various important information from all around the web and reporting them to you,

scrap information from certain aspects of your system and store them in a certain way or any other thing.

You'll also be able to do stuff way more quickly: Renaming 10000 different photos to a different name can be done in only 1 command. Converting an `.avi` film into an `.mp4` can be done in less than a minute. Separating audio and video in a file into 2 different files is so easy. Replacing a certain text in 1000 text document is also done in one single command. Searching for a specific photo/text/audio or any other kind of information is also so easy…

See this tweet for some of the CLI magic:

The possibilities are endless, engineers, doctors, university professors, students, IT people and everyone else are using the Linux command line to do basically everything they need in an easier and more quickly way.

Basics of Linux Commands

We will start by learning the very basics of the Bash Shell, and other Linux command line concepts.

What is a Shell Session?

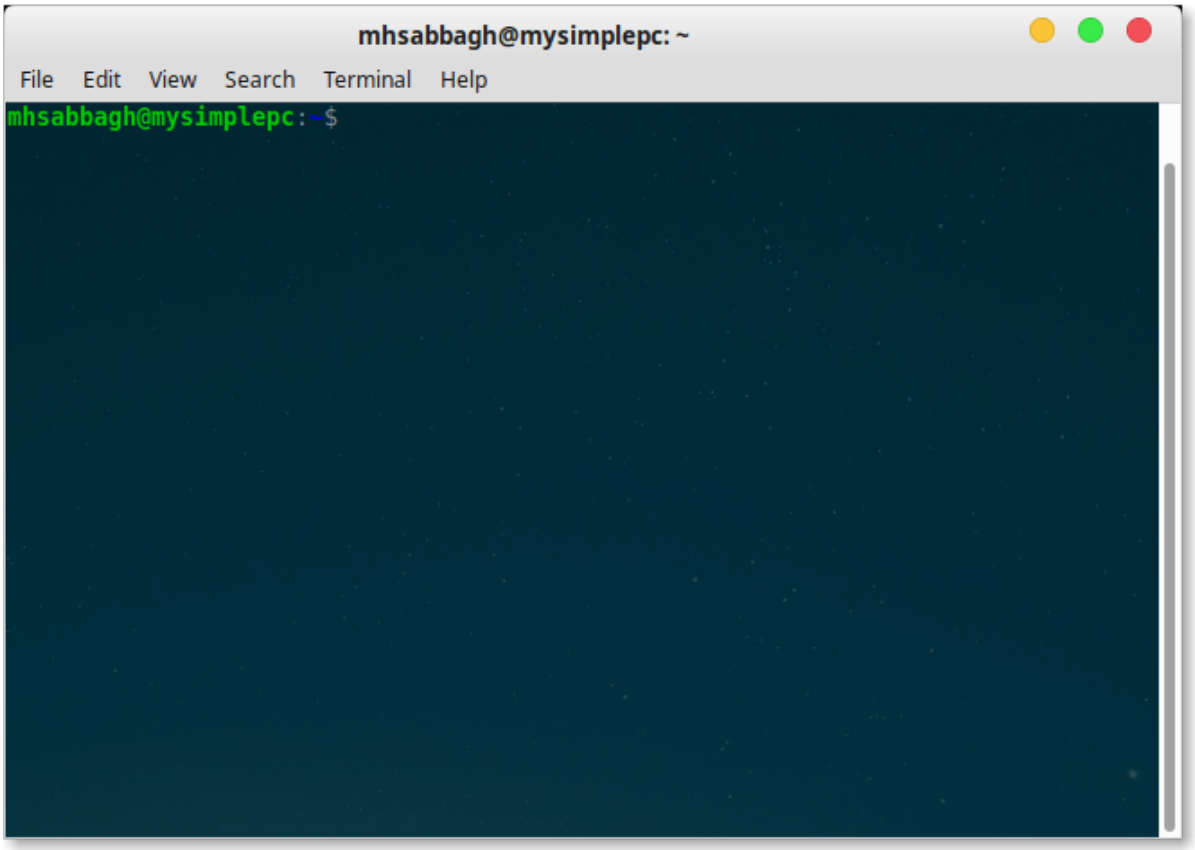
After your system boots up (regardless of the distribution you are using), a new shell session is opened for you during the whole period of your usage.

A **shell session** is basically an environment that allows your programs and other software on the system to communicate between each other and the system as a whole.

You can open many shell sessions at the same time, and some of them may be parents of other sessions, which for example will cause your currently running processes to be terminated if you terminate the parent shell session.

Linux Terminal Key Concepts

Let's start by opening the "terminal emulator", which will create a new sub-session for you in a window. Search for "Terminal" program in the applications menu of your system and see if you can find it. For example, GNOME Terminal program:



The first thing you'll see in most terminal emulators is the text put before each line you write in the terminal, which conists of 3 parts:

1. The current username of the session.
2. The current hostname (after the `@` symbol).
3. The current location (after the `:`), in our case here it is `~`, which is a symbol that stands for the home directory of the user.

Let's start by writing a simple command, like the following one:

```
cd /home/
```

The previous command consists of two things: The base command `cd`, and the argument that you pass to it which is `/home/`. After you write the command and you hit the "Enter" button on your keyboard, you will notice that the current folder of your shell session was changed into the `/home/` folder, meaning that `/home/` is becoming your current working directory.

In so many cases, you can pass more than one argument to a command to make it do many things. For example, the following command will list the contents of both the `/etc` and the `/home` folders:

```
ls /etc /home
```

You can also pass options (or multiple options) to the commands. For example, the `-a` option, if added, will display the hidden files (which have their names started with a `.`):

```
ls -a ~
```

Common Bash/Linux Command Line Symbols

There are so many special "hacky" symbols that you can write in your Bash to do stuff more quickly. Here's a quick list of them with details:

Symbol	Explanation	Examples
~	is equal to the current user's home directlry. E.g: /home/someone/	cd ~ ls ~
*	A symbol which stands for "everything". Let's say you want to remove all the .jpg files from your Downloads folder which have their name starting with the "E" character, then you can use this symbol to represent all the other letters except E. See the example.	rm ~/Downloads /E*.jpg ls /etc/*c nano /var/log /nginx/*
&	Run a command in the background. It will return the PID of the newly running process to you and won't show you the output.	sudo apt update &
&&	These symbols written together stand for "and". So if you want to run 2 commands together, you can use it.	sudo apt update && sudo apt upgrade
\	Allows you to continue writing commands/Bash syntax in new line.	sudo \ apt \ update
..	In many cases, especially in navigation, the two dots stand for the parent folder.	cd ..
.	In navigation or referring to files/folders, the dot stands for the current folder.	ls .
#	Everything after this symbol in the same line is considered to be a comment, so it won't be processed by the shell.	cd # This commands moves you somewhere.
|	This is called "Piping", which is the process of redirecting the output of one command to the input of another command. Very useful and common in Linux/Unix-like systems.	cat /etc/profile | grep bash
>	Take the output of a command and redirect it into a file (will overwrite the whole file).	ls ~ > output.txt
<	Read the contents of a file into the input of a command.	grep bash < /etc/profile
>>	Append a text or a command output into the last line of a file.	echo "First Line" > output.txt echo "See this is the last line" >> output.txt

The following table shows you the basic commands you can use for navigation in Bash:

Base Command	Explanation	Famous Arguments & Options	Examples
cd	This command allows you to move into a different directory on your Linux	. Stands for the current directory. .. Stands for the	cd /etc/

Base Command	Explanation	Famous Arguments & Options	Examples
	system, which will make it the current folder of where your shell is running. It's just like as if you open a specific folder in any graphical file manager.	parent directory. ../.. the parent of the parent directory.	
ls	Lists the current files and folders in a specific directory.	-a shows the hidden files too. -l shows metdata about files and folders, such as permissions, last modification date, size and so on.	ls ~
pwd	Gives you the current location	-	-

And this tables shows other commands for different tasks:

Base Command	Explanation	Famous Arguments & Options	Examples
touch	Make a new file.	-	touch text.txt
mkdir	Make a new folder	-p Make the requested path regardless if the sub-directories exist or not (because normally, mkdir would return an error if you try to make a 3rd-level directory while the 2nd-level directory doesn't exist).	mkdir newfolder mkdir something\ with\ spaces mkdir -p newfolder/subfolder /subsubfolder
rm	Remove a file or a directory.	-rf Adds the ability to remove folders and their contents (because normal rm can't).	rm file.txt rm -rf foldername
head	Get the first 10 lines of a text file (or the first n lines of a file)	-n Specify the number of lines to output from the beginning.	head /etc/profile head -n 19 /etc/profile
tail	Get the last 10 lines of a text file (or the last n lines of a file).	-n Specify the number of lines to output from the end.	tail /etc/profile tail -n 18 /etc/profile
cat	Output all the contents of a file	-	cat /etc/profile

Base Command	Explanation	Famous Arguments & Options	Examples
grep	Output the lines contain a certain string from a file only.	-	cat /etc/profile | grep "Bash"
chmod	Change the permissions of a file.	+x Make the file executable 777 Allow the file to accessed, written and executed by everyone (very dangerous!). 755 Allow everyone to read the file, but only the owners to edit and execute it.	chmod +x test.sh chmod 755 test.sh

How To Know What a Linux Command Does?

Since you are a new user, you may see many people passing different Linux commands on the Internet that may do different tasks. But how can you understand what these commands will do on your system if you apply them?

ExplainShell.com is the answer.

Let's say that you encountered a very long command while browsing an online article or a book, and you didn't know what does it do and how? Just paste it into the website and it will tell you what each part of it does. It's an amazing online website to explain Linux commands.

Also, there exists a legendary command called `man` that would show you all the available documentation about a specific command. For example, you can run the following commands to see the documents for `ls` and `rm` commands:

```
man ls
man rm
```

You can browse the documents using the `Up` and `Down` arrow keys on the keyboard. To exit, you can just hit `Q`.

Bash Configuration Files

The Unix design philosophy includes a rule that says that "**everything is a file**". This means everything in your PC, whether it was your keyboard, mouse, graphics card, USB sticks, monitors, shell sessions, input/output operations, processes, programs and everything else whether it was software/hardware are all represented as files somewhere on the filesystem. This means that you can access/manipulate any part you would like via just accessing these files in a specific way.

This also applies to Bash, there are many configuration files located all around your system which you can modify in order to do certain things on your OS. Here's a table of these files and what they do:

One famous trick to shorten the long commands that you usually write into smaller ones is using the alias command. For example like this:

```
alias update="sudo apt update && sudo apt upgrade"
```

The next time you enter the `update` command in your terminal emulator and hit Enter, your Ubuntu-based system will update its package information. This is very needed if you usually write long commands every day to do certain tasks. Put the previous command as it is to the end of the `~/.bashrc` file and it will be permanent (because if you don't the shell will forget your alias once you close it):

Bash alias command

Basics of Bash Scripting

We will go now in more advanced ideas and concepts about Bash Shell and its commands.

Introduction

A shell script is nothing more than just a file that contains a set of Bash syntax/commands in an organized way to do a certain task. You literally just open your text editor, write your commands there and save them with the `.sh` suffix (like `test.sh`) and run them later anytime you want from the terminal emulator.

You can put the `#` symbol anywhere you want in Bash so that anything after it in the same line is counted as a comment, not code. An exception from that is if the `#` symbol was followed by `!` in the first line of the file, which means that you are telling the shell which shell you want to use to interpret the shell script (maybe you don't want to use the current one?).

A Bash script starts with the `#!/bin/bash` line. This is important so that you specify the path for the shell that you want to use to execute the script, because for example, you may want to use a different shell like `/bin/zsh` or

`/bin/dash` to execute the script if you need that (but of course they may have a different writing syntax, so you should check for compatibility first). But almost in all cases, you won't need to use anything other than Bash throughout all your Linux career. Also, almost in all Linux distributions and some other Unix-like operating systems, the Bash default location is in `/bin/bash` (So you don't need to do any modification).

Example Shell Script

Open your favorite text editor, and write the following commands:

```
#!/bin/bash
cd ~ # Go to the current home directory.
ls # List the current files.
echo "Done!" # Print "Done!"
```

And save the file as `test.sh` in your home folder. Then open your terminal emulator and run the following command to run your Bash script:

```
source ~/test.sh
```

The `source` command executes a specific file that you want in the current shell you are running. Here you are just telling your current Bash shell to execute the file for you. An abbreviation for the `source` command is the dot (`.`). Meaning that you can write the previous command like this:

```
. ~/test.sh
```

You can also execute a script like `./test.sh`, but in this way, the shell will run the script in a new shell, not in the current one. Also make sure that have navigated to the current directory of the shell script before writing `./test.sh`. (Or, you could write the full path: `~/test.sh`).

Variables in Bash

You can also define/access variables in Bash, and you can print them using the `echo` command. Here are some examples:

```
myvar="Test"
echo $myvar
echo "This is a: $myvar"
echo myvar
```

Notice how some of these will give you a correct output, while the last of them will not. You need to be very careful in the way you write Bash commands and scripts; each space and each symbol is very important. For variables, we use the `$` symbol to refer to a variable. The string quotes for example are important, look the following example:

```
test="Something"
echo "This is: $test"
echo 'This is: $test'
```

In the last case, Bash won't process the variable, but it will print the sentence exactly as it's inside the quotes.

Some variables are very helpful in Bash. For example, the `$HOME` variable is the path of the current user's home folder. So you can use it this way to access it:

```
cd $HOME
ls $HOME
```

You can print all the currently defined variables in your current Bash session via the following command:

```
env
```

You now have a very good understanding for the Linux terminal and the Linux command line in general. Of course, there is still so much to learn beyond the scope of this article, but you are in a good place to start learning more things according to your own needs.

Additional Resources

Here are some additional resources so that you can learn the command line of Linux.

Linux Command Line PDF Books & Guides

There are so many books, videos, courses and online tutorials to guide you through learning the Linux command line. Here's a small list of them to get you started:

1. The Linux Command Line: A Complete Introduction: A famous book to start learning the topic. Made in 544 pages that would explain everything you need about writing Bash commands and scripts. Very recommended to read. (It's available PDF free from the website).
2. Linux Command Line Tutorial for Beginners: If you are someone who prefers video content over written one, then this Youtube set of videos is for you. Made in 80 different videos averaging around 10 minutes each, this series takes your hand in explaining various Linux commands beside more advanced topics in writing shell scripts.
3. LearnShell.org: A free online interactive website to learn the shell. You basically write everything and learn everything from inside your browser.

Linux Command Line Cheat Sheets

Cheat sheets are simple single page/multiple page images or PDF documents explaining the most important aspects that a person may need to remember any topic. Linux command line cheat sheets exist so that you can understand what are the most commands to use, or what are the available symbols that you can employ in your usage.

Different people can make different cheat sheets, and hence, they vary in quality.

There are few Linux command line cheat sheets that you can get for free:

- Linux command line cheat sheet by Cheatography: Simple in 2 PDF pages, explains default keyboard shortcuts and I/O operations in Bash.
- Linux command line cheat sheet by Loggly: Small 1-page cheat sheet about most of the basic commands.
- Another one by FossWire: Although dates back to 2007, it is still up-to-date and can be used. Focuses on system, network and file management aspects.
- You may find others simply by searching online.

If you have reached this far, then you should have a good experience about Linux commands, what are they and how they work. Read the books that we previously recommended so that you learn more. Additionally, you may follow the following Twitter accounts for day-to-day learning and interesting tips and tricks about Linux command line:

- Command Line Magic: Cool Unix/Linux Command Line tricks you can use

in 140 characters or less. Here mostly to inspire all to try more. Read docs first, run later.
- Daily Linux Commands: Daily tweets of useful Linux commands which will take your terminal skills to the next level.

If you have anything unclear in your mind about the topic, or you have any note/comment about what's mentioned in this post.. Let us know below!