An Introduction to BASH Command-Line for the beginner

Koushik “Razor-X” Roy

Disclaimer:

Contents

1 Introduction

Where we see beauty, most see darkness

1.1 History of the Shell

To start of the Command Line guide, I'm going to start off with a short history of the Shell itself, as it evolved in tandem with Linux. As we all (should) know, Linux is just a kernel, nothing more. Everything that modern Linux distributions use today are add-on software. The only thing that can really be considered "Linux" is the kernel. But what does this have to do with the shell? Well, as is apparent, there are many alternate solutions for one problem in the Linux world, thanks to the fact that the only standardized Linux thing is the kernel. But, there are a few things that most people take for granted as having no alternatives in this day, such as the shell: BASH. What most people don't know is that, just like other utilities for Linux, there are alternate shells for Linux. It just so happens that BASH is the most popular. Come with me, and I'll take you on a tour of the evolution of shells from their roots in UNIX, and what led up to BASH.

The first UNIX shell ever was SH, the Bourne SHell. As it was the de facto UNIX standard, many many scripts were written for it by numerous people around the world. Finally, the first deviant shell arose from the basic SH shell. It was called CSH, a shell with a syntax similar to C. But, because of its total incompatibility with the majority of shell scripts existing at the time, CSH never grew popular. People kept on using SH.

Then came the next deviant shell, KSH, or Korn SHell. Named after its creator, David Korn, KSH was fully compatible with existing SH scripts, and had some features that CSH came with. KSH was also much faster than CSH, which drove the little popularity CSH had down. KSH remained popular for quite a while.

Finally, ZSH came on to the scene. ZSH was a shell that combined the best of SH, CSH, and KSH into one shell. ZSH also accorded quite a bit of popularity. ZSH is even included in today's Cygwin distributions.

Still, around this time, a remake of SH was coming up, called BASH or the Bourne Again SHell. This shell quickly exceeded ZSH in popularity, and became one of the de facto shells. When UNIX forked, Linux embraced BASH, and BSD embraced CSH. The current default shell in FreeBSD and OpenBSD is TCSH, a superior version of CSH.

1.2 Format of Commands in this book

Throughout the book, you will be shown Linux commands. This section is designed to make you familiar with the format of the commands that we write about.

```
program -t -e -s <parinput> -T input
```

Take a look at that. Now, we'll take it apart in small bits. To start a certain program in your path (I'll get into that in a second) you must type `program`, and make sure that it matches the case of the program you want to call. All Linux input is case-sensitive. Make sure to type it in correctly. The last word of the command is `input`. This is the file that you are passing to `program` as input, so it can process it according to the function of `program`.

Now, look at the second part. It's a group of letters with attached hyphens separated by spaces: `-t -e -s -T`. These are called the program parameters, and are passed to the program when the program starts running. Each one of these parameters such as `-t` have a special meaning. Make sure to observe that the parameter `-s` is followed by `<parinput>`. This notation is exclusive to the book, as it denotes input that a specific parameter requires, in this case, the parameter `-s` requires any file to be passed with the parameter. It would look like `program -s parinput input`, where `parinput` is the input for the parameter `-s` and `input` is the input for the program `program`. Also, note that `-T` is capitalized, just like all other input, parameters are case sensitive.

Rather than writing about every parameter, this book will focus on the most used ones. If you need to learn about other parameters, read the man page.

1.3 Piping

BASH is a very powerful shell with a multitude of scripting and other input directing options. While it is out of the scope of the document to introduce scripting, this document will introduce the concept of piping, as it is the best way to use many commands.

First, I will go into a simple primer on Linux program output. Almost all programs do something and write their results to the console. Each of these Linux programs is writing their output to the console. When a program writes its input to the console, it is also considered as writing to `stdout` or "Standard Output".

Piping allows you to redirect output to places other than `stdout`. Many examples of piping exist. When asking a command for a list of processes, by default, the command sends its output to `stdout`. But, the list of processes may be large, and you would like another program to sort through the list and find one specific line you are looking for. In this case, you must pipe the output of the first program to the second program. The second program then writes its input to `stdout`. Piping can be performed indefinitely. For an example of piping, you can do `ls -a | more`. In this case, `|` is the piping character, it tells BASH to pipe the output of `ls -a` to the program `more`. `more` will be covered in more detail in the Basic Commands section.

1.4 Linux Directory System Primer

Note: If you already understand the material, feel free to skip this section.

The Linux Directory System is based on the original UNIX Directory System. As a consequence, almost all UNIX forks have similar Directory Systems. In Linux, the base directory (the drive itself) is `/`. This is equivalent to `C:\` in Windows and DOS, where `C` is the drive on which the Operating System is installed in. All subfolders of `/` are appended on to the name. To access the subfolder `home` on `/`, the directory would be `/home/` The trailing slash at the end of the line does not need to be included. The Windows or DOS equivalent of this directory would be `C:\home`. In Linux, there are generally some default directories: `/bin`,`/boot`, `/dev`, `/etc`, `/home`, `/lib`, `/mnt`, `/proc`, `/root/`, `/sbin/`, `/sys/`, `/tmp`, `/usr`, and `/var`. Many of these directories hold certain types of files. It is out of the scope of this document to explain the significance of each of these folders. Suffice to say, these folders are important in most modern Linux distributions. Also in Linux, there are hidden files and folders. These files or folders are preceded by a `.` to signify that they are hidden. For example, `/home/.asoundrc` is a hidden file called `.asoundrc` in the `/home` directory.

2 Basic Commands

2.1 Introduction

The Linux command line is managed using multiple programs that perform certain functions. This chapter will explain the basic commands, their uses, and how they can be used together for optimizing tasks. The BASH shell also contains a very sophisticated method of scripting, but BASH scripting is out of the scope of the document. By the end of the chapter, you should be able to perform many common tasks using BASH.

Note: All lesser-used program parameters will be ignored

2.2 man

```
man programname
```

Manual, or `man` is the most useful Linux program by far. The function of the man program is simple. By typing in `man` preceding the `programname` you view the program's manual. Most commands have manual pages, and most manual pages have important information about the command in question. A `man` page contains the program syntax and possible parameters, plus the explanation of the parameters. Man itself has quite a few parameters, but they are very rarely used. Why not find them out yourself and try out your first command by typing `man man`?

2.3 cp

```
cp -R -v input destination
```

Copy, or `cp` is one of the most basic commands. This command is one of the fundamental commands in most Command Line interfaces. Simply, `cp` copies a file from `input` and places it in `destination`. In many Linux texts, you will see the word `foo` as a moniker for `input` and the word `bar` as the moniker for `destination` or `output`. As these are nondescriptive names, this book does not use the words `foo` or `bar`. Also,

In `cp`, the most common parameters are `-R` and `-v`. When copying directories, in many cases, you want to copy the contents of the directories as well. The `-R` parameter enables you to do this. In many Linux programs `-R` performs the same function, as it means "recursive". The next parameter `-v` also is very common in other Linux programs. This command stands for "verbose", or verbose destination.

2.4 mv

```
mv -R -v -i input destination
```

Move, or `mv` is another basic command. This command is used almost as often as the `cp` command. The Move command moves a file from `input` to `destination`.

Common parameters of `mv` include `-R`, `-v`, and `-i`. As above, `-R` stands for "recursive", and `-v` stands for "verbose". The parameter `-i` stands for "Interactive Mode". In "Interactive Mode" `mv` will prompt you for every moved file.

2.5 rm

```
rm -R -i -v input
```

Remove, or `rm` is the last basic command. This command is used often, but not as often as `cp` or `mv`. The Remove command does exactly what its name implies, it removes a file or a directory. from the drive. If any file or folder is deleted using `rm` then it is unrecoverable, unless special software is used.

The parameters of `rm` are pretty straightforward. From above sections, `-R`, `-i`, and `-v` should be known as "Recursive", "Interactive Mode", and "Verbose" respectively. Only `-R` can be used to delete directories. When using `-R` all files and folders in a folder specified will be deleted.

2.6 ls

```
ls -a -aa --author -C -d -x
```

List, or `ls` is a very basic command with a very length set of parameters. The function `ls` has is very simple. `ls` lists the contents of a directory on to the console. The parameters are used to manipulate the format of the output of `ls`.

The first parameter `-a` stands for "all". When using `ls` and calling `-a`, everything in the directory will be shown including hidden files, . and .. (. is the current directory and .. is the directory directly above it.). The second parameter `-aa` is "almost all". This parameter shows everything that `-a` does without the . and .. included.

2.7 grep

2.8 ln

2.9 chown

2.10 chmod

2.11 sudo

2.12 alias

2.13 cat

3 Text Editors

I would like to think of myself above the man who would forsake ketchup over mustard if only to avoid the possibility of choice

3.1 Introduction

The world of Linux is filled with choice. So much so, that there is almost always a program that suits everyone's taste. The trouble with Linux is that, this gargantuan amount of choice often complicates the user environment, because these choices are not thoroughly explored. I firmly believe that the true power of a Linux system is the power to configure every portion of the system to tastes tailored to your own person. Therefore, knowing the array of choices, and understanding what each choice means is essential to customizing a system.

Another thing in Linux is ease-of-use. Some users prefer ease-of-use over anything else. While I strongly suggest sampling every choice and picking through your preference, if you absolutely must pick the easiest text editor to use, then you should choose nano.

If this is your first introduction into the command-line world, you may want to know the point of using text editors. The UNIX world thrives on text files. Unlike the behavior of Windows which stores settings in a central repository called the registry. This registry stores settings using different variables, many of which are hexadecimal. The convoluted system of the Windows registry is a very effective detractor away from configuring it in Windows. On the other hand, in UNIX, everything is stored in accessible plain-text, with features such as comments that severely increase the readability of the settings you are reading. But, in order to edit these text files, it's almost a given that you must have a nice text editor. The chore is finding a text editor that you are comfortable with. The hope is that by the end of this chapter, you will have found a comfortable text editor, or at the very least, be on the lookout for new text editors on the internet, as the pool of text editors is almost limitless. Note also that the list in this guide is restricted to free text editors.

$\boxed{\text{Note: All program choices are presented alphabetically with no partiality.}}$

3.2 emacs

3.2.1 Overview

Emacs is a powerful command-line text editor. Emacs is one of the text-editor powerhouses, and a legacy editor. Emacs is an editor that defines the extent of the power of text-editing, if a feature can exist, it has most likely been added. The way features are added to emacs, is through a programming language called emacs-lisp. This programming language allows users to add numerous features to emacs.

The power of emacs lies not just in its text editing capabilities, but also its multipurpose nature. A special feature of emacs, is that, emacs can do much more than a text editor should. Emacs can be used to run terminals, it can be used to chat on IRC, and surf the internet. Emacs (for better or for worse) has been called it's own distro. Still, this power comes with its own crippling duality. The size of emacs is enormous, much more than leaner editors such as vim. Still, if you prefer to edit a file with a text editor that can support any possible feature that you could want, then emacs is the right editor for you.

3.2.2 Movement Controls

The emacs movement controls are heavily based around the concept of the Control and Meta characters. The Control key is your comon Control key found on most keyboards. (Note: In the original UNIX keyboards, the Control key was located where the current Caps Lock key is located on the US-101 keyboards. This is one of the reasons that emacs is based around the Control character, as it was in a convenient location. Microsoft's classic disregard for standards pushed the Control key to occupy a grim fate in a relatively untouched corner of the keyboard.) On most current Linux distributions, the Meta key is the Left-Alt key. In Linux, this behavior maybe changed using the `setxkbmap` command (this can also be used to switch the Caps Lock and Control keys to retain the layout of the original UNIX keyboards). In this guide, I will give a very rudimentary rundown of the movement controls in emacs. Emacs is explained in much more exhaustive detail in the tutorial included with emacs, if you want to try and experience emacs, I strongly urge you to use this tutorial.

The notation for emacs controls goes thusly: `C-a` represents the key combination of Control and 'a' with Control being depressed before 'a'. `M-a` represents the same key combination as `C-a` except that the Meta key is being depressed instead of the Control key.

Key Binding	Description
C-f	Forward One Character
C-b	Backward One Character
C-p	Previous One Line
M-f	Forward One Word
M-b	Backward One Word
M-p	Backward One Sentence
M-f	Forward One Sentence
C-v	Forward One Page
M-v	Backward One Page

Something to note in the limited emacs bindings I have presented here, is that, for each of the commands that the `C-` key performs, the Meta key equivalent performs something that has to do with something tangible. In this example, `C-f` scrolls forward one character, which is a computer term, whereas, `M-f` scrolls one word forward, which is more tangible than a computer term.

Of course, just as in any other legacy text editors, emacs supports scrolling using the arrow keys, but, it is suggested that arrow keys are not used for scrolling in emacs because certain terminals (more on that later) do not support arrow keys.

3.2.3 Muscle Memory

If a text editor is used enough, eventually the most used commands (and, if you use text-editor's movement commands (as you're supposed to be doing) they will be the first to be embedded) will embed themselves into your muscle memory. If you use a text editor for almost all operations (such as I) then these muscle bindings will dictate everything you do. Therefore, I spend a subsection in each chapter explaining the muscle-binding effects of the text editor on your hands, so that if you find any one of these muscle momery's undesireable or harmful in some way, you will stay away from the program. (Remember, ergonomics is a major part of the usage of a text editor, I myself use the text editor I use for something specific to ergonomics. If, for some reason, the binds of a text editor are not to your liking, or painful to reach, it is strongly advised not to use the editor, because RSI may develop later.)

Emacs muscle memory is dependant on the control keys. If you are set on using emacs, but the distance of the control keys offstancesz you, you can do what many others have done - switch the left Control key and the Caps Lock key. This will make the Control key much closer to your fingers, and make emacs editing a breeze. If you're willing to go the extra mile to use normal Control keys (as many people do nowadays) then be sure it is not too taxing on your wrist. And, while some control based commands are centered around common QWERTY positions, not everything is, so emacs is a nice, powerful editor for those using non-standard keyboard layots like Dvorak.

3.3 joe

3.4 mcedit

4 Package Managers