Jumpstart Linux
Bo Waggoner
Updated: 2019-01-22

Abstract

A basic, rapid tutorial on Linux and its command line for the absolute beginner. Focuses on the basics of *using* Linux and the command line (not installing it).

1 Getting Started

There are many installation guides online, but here is a quick idea of what to expect.

Selecting a distribution. There are many “versions” or “brands” of Linux, known as distributions. If you aren’t sure which one to get, just pick Linux Mint for now. It isn’t necessarily any better than any other distribution, but it isn’t any worse, and it’s easier to follow instructions that focus on one particular distribution. Its website is `https://www.linuxmint.com/`.

Installation. You will probably need a recent USB drive. Follow instructions online to download an installation image and place it on the USB. Then you will boot the computer from the USB drive and install it to the computer’s hard drive according to the instructions. There are also guides available on dual-booting (installing Linux alongside your current operating system, so you can choose which system to use on startup).

What you should know about Linux.

- You can use regular GUI programs to change your settings, browse files, edit documents, surf the web, etc. just like Mac and Windows.
- All of these things and more can be also be done by typing commands.
- All files are organized into a single hierarchy of folders.
- Everything you want to access – evey file, program, setting, configuration, USB drive, sensor reading – is represented as a file and located somewhere in that hierarchy.
- You can mostly get away with using regular GUI programs that will change these settings and access everything “under the hood”, but it is helpful to know how to use the command line and what the file hierarchy looks like.

2 The Command Line, Part 1

In beginner-friendly distributions like Linux Mint, you can do most things you need without ever touching the command line. You can open the Menu and find your Settings, your web browser (probably Firefox by default), your Office Applications (probably LibreOffice, which serves the same niche as Microsoft Office), etc. You can navigate through your folders and documents.

But the heart of Linux is the command line, where we can type commands (instructions). Everything you do through the menu — everything you can click on — can be done on the command line as well; but the command line can do much more.

In the menu you can find the command line application (usually called a shell or terminal). In Mint, it's called the Terminal. If it looks like mine, you'll see

```
username@machinename ~ $
```

with a blinking cursor after the $. Here "username" is the user name you've set yourself and "machinename" is the name you gave your computer in the installation process.
Usually, for shorthand, we write a command to be typed into the terminal by just putting $ sign before it, and I'll put things we type in blue. Everything else is what's printed in response to our commands. For example, let's run the "pwd" command.

```
$ pwd
/home/username
```

pwd is a program; it stands for "print working directory". Print means print. *Directory* means folder. Working just means the folder we are currently in. By default, the Terminal should place you in your *home* folder: /home/username. So that's what it printed.

You can see a list of items in this folder with the "ls" command ("list") (again, type and press enter):

```
$ ls
Desktop    Downloads  Pictures  Templates
Documents  Music      Public    Videos
```

Of course, if you have other files or folders in your home folder (or don't have one of the above), you'll see a somewhat different output.
To change folders, use the "cd" command (for "change directory"). For example, we can go to the folder "Documents" with

```
$ cd Documents
```

(If you like, you can open up the file manager — that is, the GUI one with icons and clicking on folders — and follow along there as you execute these commands in the terminal.)
In the command line, we use "." to mean "the current folder" and ".." to mean the folder above. So we can first check where we are, then go back to the home folder by executing

```
$ pwd
/home/username/Documents
$ cd ..
$ pwd
/home/username
```

3 The Directory (Folder) Structure

The tilde symbol ~ is a shorthand for your personal home directory. So

```
$ cd ~
$ pwd
/home/username
```

Looking at this again, it says that we are in the folder “username” which is in the folder “home” which is at the top level. We just call that top-level directory “/”. Everything on the computer is stored somewhere under it. Let’s see by going to that folder and listing what’s in it:

```
$ cd /
$ ls
bin   dev  home  lib64       media  opt   root  sbin     srv  tmp  var
boot  etc  lib   lost+found  mnt    proc  run   selinux  sys  usr
```

(Again, you may have slightly different output, but most of the contents will be the same.)
For now, I’ll only explain three of the folders listed: bin, usr, and home. We’ve seen home before: It’s where *your* personal home folder goes (same for other users of the computer).

/bin contains the essential programs that you can run. For example, the commands we’ve been executing, like “ls” and “pwd”, are simply programs, and most of them so far are located in /bin. If you type something in the command line and hit enter, Linux will look in /bin to see if that’s the name of a program; if so, it’ll run that program.

Finally, /usr also contains a folder named bin. The programs stored in /usr/bin/ are usually the programs you install and use, rather than the ones that are essential. For instance, if you have firefox installed, then you probably have the file /usr/bin/firefox, and if you execute it:

```
$ /usr/bin/firefox
```

then Firefox will open and run. Just like /bin, Linux will automatically look in here whenever you type a command, so

```
$ firefox
```

works just fine.
(Sidenote: the Terminal running on most versions of Linux is called *bash*. That is, “bash” is the name of the program you are using when you’re using the command line, and yes, it’s located in /bin! You can search online for tips on using bash.)

4 Sudo and “root” access

When you are logged in as “username”, you typically only have limited permissions to access certain files or make certain changes. Everything in you home directory (remember that’s “/home/username”) is fair game, but things like /bin/ may be restricted, and you can’t install or remove software.
However, hopefully you made your account an administrator. On Linux this is also called “having root” for reasons we won’t get into. If your account has root access, then you can still do those restricted things, you just have to enter your account’s password first. You might see this when changing some of your computer’s settings.
On the command line, we can execute a command as root (meaning with our administrator privileges) by prefacing the command with “sudo”. For example, suppose we wanted to add a new file to the folder /bin. We could try opening it with a text editor, like Pluma[1]:

[1] If you don’t have pluma installed, substitute with the text editor you do have, like gedit or kwrite or so on.

```
$ pluma /bin/mynewfile.txt
```

If we run this and try to save the file, we'll run into a problem: "You do not have the necessary permissions." For this to succeed, we'd have to run Pluma as root:

```
$ sudo pluma /bin/mynewfile.txt
```

The command line would then prompt us to enter our root password before launching Pluma, and we would be able to save the file successfully.

5 Package Managers: Install (Free) Software Really Easily (For Free)!

This is one of the most awesome parts of Linux. Package managers handle the installation of most software you'll ever need. You tell them what software you want, and they connect to a repository, download, and install the software for you.

You can find a GUI (graphical user interface) version of the package manager in the Menu, but we'll focus on accessing it through the command line. On Linux Mint (and other distributions based on Debian, which is a fundamental Linux distro), the package manager is accessed through the "apt" program. For instance, suppose I'd like to install the GNU compiler for the C and C++ programming languages; then I can simply open a command line and execute

```
$ sudo apt install gcc
$ sudo apt install g++
```

I have to use sudo here because installing software requires root. Then I'm done![2]

If I want to find out if "myprogram" is available for installation, then in addition to searching online, I can run

```
$ apt search myprogram
```

Any programs mentioned in this document are available via the package manager for just about any distribution of Linux. (They have different names, like dnf on Fedora and pacman on Arch, but they work pretty much the same way.) Any time you need to install some sort of software on Linux, it's worth checking whether the package manager has it before trying a manual installation. Most of the time, it will be there.

6 Plain Text and Command-Line 2

One of the important tenets of Linux, along with the command line, is the use of plain text files as much as possible. For example, all of your system configuration and settings are stored in text files somewhere. For the most part, just editing those files is a completely normal way to change your settings. When you use a GUI, it will just write those changes to those files under the hood. Plain text allows for both.

Many command-line tools in Linux are designed to process plain text, search it, manipulate it, etc. For instance, "grep" searches through text for the given text (or regular expression). If we want to find the

[2] Occasionally the installation will pause to ask for confirmation of certain permissions, like using a large amount of disk space, and you have to type 'y' or 'n' for "yes" or "no" – think of these like the little confirmation popups, but easier to deal with.

phrase “hello” in the file “hiworld.txt”, then we would execute

```
$ grep "hello" hiworld.txt
```

This prints all the lines in the file hiworld.txt that contain “hello”, with “hello” colored red. If we wanted to find it in every file in the working directory (that is, every file in the current folder), then we could use a special character, “*”, which means “every file in the directory”:

```
$ grep "hello" *
```

Arguments. When we type a command such as “cd /home”, the first full word (before a space) is the command or program we are actually running. The rest of the words, such as /home, are the *arguments* or *parameters* we give that command. In this case, we tell the cd command to go to /home. If the argument has spaces in it, we have to use quotes, otherwise it is treated as multiple commands!

```
$ cd My Stuff     # Wrong!
bash:  cd:  too many arguments
$ cd "My Stuff"   # Right!
```

Flags and man pages. Most programs have various options that can be set by what are called “flags”. These are usually of the form “-r” (a single dash followed by a single letter) or “–recursive” (a double dash followed by a word or short phrase). These control the behavior of the program.
For instance, we can ask grep to search, not just a the current directory, but all subdirectories as well (and all of their subdirectories, etc.). To do this, we use the -r flag:

```
$ grep -r "hello" *
```

It often doesn’t matter whether the flag goes before or after the arguments (an argument is an input like “hello” in the above example).
To find out about a command’s possible flags and foibles, we can check its “man page”, where man is short for manual. Almost every command has a man page, accessible by executing, for instance (try it out!),

```
$ man grep
```

Additionally, many/most commands support the –help flag, and will give you information if you run

```
$ grep --help
```

7 Wrapping Up

This intro has only scratched the surface. But you only need to know how to scratch the surface in order to get up and running with Linux. You might barely need to use the command line at all, or you might start enjoying it and becoming a command-line wizard(ess). In any case, good luck!