Jumpstart Linux
Bo Waggoner
Updated: 2014-09-15

Abstract

A basic, rapid tutorial on Linux and its command line for the absolute beginner. Prerequisites: a computer on which to install, a DVD and/or USB to use for installation. This document focuses on the basics of *using* Linux and the command line rather than installing it; installation guides abound elsewhere on the web.

1 Installation (briefly)

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 `http://www.linuxmint.com/`.
These instructions should also work well with Ubuntu, whose website is `http://www.ubuntu.com/`.

Installation. Go to the Downloads page on your chosen distribution's website and follow the instructions to download. (If in doubt, usually you can just pick the top link on the page. If your computer was purchased after 2009, it very likely is 64-bit.)
You will need to burn the installer image to a DVD or use it to create a bootable USB, then boot the computer from that and install. You can find detailed instructions online, *e.g.* at `http://www.linuxmint.com/documentation.php`. 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).
It is very helpful to have an Internet-enabled device available in case you need to troubleshoot during installation. Hopefully, everything goes well (but don't be surprised if it takes a little tweaking to work with your wireless or so on).

2 The Command Line, Part 1

Beginner-friendly distributions like Linux Mint let you 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.
To start, open the command line application from the Menu. 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 write anything that we type in blue:

```
$ pwd
```

Type pwd into the command line and press enter; this executes the pwd command.[1] 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 you should see

```
username@machinename ~ $ pwd
/home/username
username@machinename ~ $
```

with a blinking cursor ready to type the next command. From now on, I'll just write this operation as

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

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

```
$ ls
Desktop    Download  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 "/home" with

```
$ cd /home
```

(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.)
Now, since we are one directory up, if you execute "ls", you should see your own home folder:

```
$ ls
username
```

(Where, again, username is your user name. If there were other users of the computer, you would see their home folders listed here as well.) Let's go back to it:

```
$ cd username
```

[1] Note: In general, you should be careful about executing commands you don't trust if you don't know what they do; they might mess up your computer! But I promise these are all ok.

3 The Directory Structure

As a preliminary note, when we are navigating on the command line, there are some "special" directories. First, "." always means the current folder, and ".." always means the folder than contains this one. So

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

We haven't gone anywhere. But

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

We've gone up one level. Also, ~ is a shorthand for your home folder, /home/username. So you can quickly say

```
$ cd ~
```

to go back to your documents from wherever you are.
Now, the Linux filesystem is organized so that, hopefully, you know where everything on your computer belongs and everything is actually located where it belongs. For now, what you need to know is that the "root" or top-level directory is just called "/". Everything on the computer is stored there.

```
$ 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'm only worried about three of the folders listed: bin, usr, and home. home we've seen before: It's where *your* personal home folder goes. Now we know why it's called "/home": It's a folder named "home", and it's located in the directory "/". All of your documents, music, pictures, anything should be stored in /home/username.
/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! If you want some tips on navigating the terminal — for instance, use the "up" arrow key to browse through recently used commands, and use "tab" while typing to see a list of the current directory's contents — you can search online for tips on using bash.)

4 Sudo and "root" access

Normally, when you are navigating around your computer with commands like cd, you are acting as yourself; that is, as "username" (whatever your user name may be). But in addition to you, there is someone else who has more power to use your machine: root. You should have set up a root password when installing Linux (or perhaps you made it the same as your own password; probably fine if you're the only user of the computer). The idea of root is that some operations are too important to be left to common users of the machine; only a privileged administrator is allowed to make certain changes. For instance, if you try to edit an existing file in /bin/ (which I do *not* recommend), you will find that you must be acting as root in order to do so.
On the command line, we can execute a command as root (rather than as username) by prefacing it with sudo. For example, suppose we wanted to add a new file to /bin. We could try opening it with a text editor, like Pluma[2]:

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

This will open Pluma, a text editing program, editing a new file located at /bin/mynewfile.txt. But if we try to save the file we're editing, 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.
As a helpful feature, if you execute a command as root, the Terminal will "remember" that you have root privileges for a short time (say, 5 minutes or so), and apply them to any additional commands you run if necessary.

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

This is one of the most ridiculously 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 Ubuntu, the package manager is

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

accessed through apt-get. 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-get install gcc
$ sudo apt-get install g++
```

(I have to use sudo because installation requires root.) Then I'm done![3]
If I want to find out if "progname" is available for installation, then in addition to Googling, I can run

```
$ aptitude search progname
```

(though let's not get into the difference between aptitude and apt-get).
Any programs mentioned in this document are certainly available via the package manager for just about any distribution of Linux. (Alternatives to apt-get, which are used for certain other distributions, include yum and pacman, 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

One of the important tenets of Linux, going hand-in-hand with the command line, is the use of plain text as much as possible. Plain text, as most programming code is written in, has many advantages from the Unix philosophy perspective. For instance, when settings are stored in plain text in an agreed-upon format, it is easy both for programs to automatically modify them and for humans to edit them by hand. Thus, in maintaining Linux, you will find yourself editing text files containing settings. If you also code, you may find it well worth learning Emacs or Vim. In general, however, Pluma and its cousins work fine. In general, prefer plain text unless you have a good reason not to. Plain text is easily searchable and sortable. For instance, to write this document I am using LaTeX, a document typesetting framework. It is basically a programming language, so the file is written in plain text with typesetting commands, then compiled to produce a pdf.
Since most of my documents are written in LaTeX, I can search and sort and spell-check them from the command line. The benefits of plain text stack. Here is a quick example — of course, these programs come installed by default or else are available with apt-get. (Note: The # sign followed by text signifies a "comment". You can just skip typing it in. But if you do type it in, anything that follows # will be ignored by bash.)

```
$ apt-get install texlive  # installs LaTeX
$ pluma myfile.tex         # followed by much editing of the file
$ pdflatex myfile.tex      # pdflatex is installed with texlive
$ spell myfile.tex         # a spellchecker
$ grep "Linux" myfile.tex  # search for the word "Linux" in the file
$ wc myfile.tex            # "wordcount" program
```

[3] 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.

7 Command Line 2: Flags, Man Pages, Pipes

Plain text works well with many key tools in Linux. For instance, "grep" searches through text for the given text (or regular expression). If we want to find the 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" *
```

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
```

Pipes. Because the universal language of Linux commands is plain text, they often play nicely with each other: We can send the (plain text) output of one command to another command, which will take it as input.
For instance, let's say we want to find all files in our documents with "hello" in their name. We know that grep can search through text for the string "hello". So we just need to feed grep a list of all files in our documents.
Luckily, we know that the command ls is used to list the contents of a folder; and by looking up its man page, we can see that it has a -R flag for listing all subdirectories recursively. So executing

```
$ cd ~
$ ls -R
```

would give us a list of all files stored in our home folder (remember that ~ is shorthand for /home/username).
Sidenote: If we pass a folder name to ls as an argument, then it lists the contents of that folder. So we

can instead simply execute "ls -R ∼".
Now, we just need to feed that text to grep. This is very easy using *pipes*, which (as expected) pipe the output of one program into another. The pipe symbol, "|", is what we need here:

```
$ ls -R | grep "hello"
```

This runs the command *grep "hello"* on the output of *ls -R*, which is exactly what we want.
(Of course, we could have just used the "find" command! But that doesn't illustrate pipes....)