

Linux

Nguyễn Quân Bá Hồng¹

October 31, 2022

¹Independent Researcher, Ben Tre City, Vietnam
e-mail: nguyenquanbahong@gmail.com; website: <https://nqbh.github.io>.

Contents

1	Shotts, 2019. The Linux Command Line: A Complete Introduction. 2e	3
1.1	What Is the Shell?	6
1.1.1	Terminal Emulators	6
1.1.2	Making Your 1st Keystrokes	6
1.1.3	Command History	6
1.1.4	Cursor Movement	6
1.1.5	Try Some Simple Commands	7
1.1.6	Ending a Terminal Session	7
1.2	Navigation	7
1.2.1	Understanding the File System Tree	7
1.2.2	The Current Working Directory	8
1.2.3	Listing the Contents of a Directory	8
1.2.4	Changing the Current Working Directory	8
1.2.5	Absolute Pathnames	8
1.2.6	Relative Pathnames	8
1.2.7	Some Helpful Shortcuts	9
1.3	Exploring the System	10
1.3.1	More Fun with <code>ls</code>	10
1.3.2	Options & Arguments	10
1.3.3	A Longer Look at Long Format	11
1.3.4	Determining a File's Type with <code>file</code>	11
1.3.5	Viewing File Contents with <code>less</code>	12
1.3.6	Taking a Guided Tour	13
1.3.7	Symbolic Links	14
1.3.8	Hard Links	14
1.3.9	Summing Up	15
1.4	Manipulating Files & Directories	15
1.5	Working with Commands	15
1.6	Redirection	15
1.7	Seeing the World as the Shell Sees It	15
1.8	Advanced Keyboard Tricks	15
1.9	Permissions	15
1.10	Processes	15
1.11	The Environment	15
1.12	A Gentle Introduction to <code>vi</code>	15
1.13	Customizing the Prompt	15
1.14	Package Management	16
1.15	Storage Media	16
1.16	Networking	16
1.17	Searching for Files	16
1.18	Archiving & Backup	16
1.19	Regular Expressions	16
1.20	Text Processing	16
1.21	Formatting Output	16
1.22	Printing	16
1.23	Compiling Programs	16
1.24	Writing Your 1st Script	16
1.25	Starting a Project	16

1.26	Top-Down Design	16
1.27	Flow Control: Branching with <code>if</code>	16
1.28	Reading Keyboard Input	16
1.29	Flow Control: Looping with <code>while/until</code>	16
1.30	Troubleshooting	16
1.31	Flow Control: Branching with <code>case</code>	16
1.32	Positional Parameters	16
1.33	Flow Control: Looping with <code>for</code>	16
1.34	Strings & Numbers	16
1.35	Arrays	16
1.36	Exotica	16
2	Miscellaneous	17
2.1	Calibre	17

Bibliography 18

I have used the following Linux distributions: UBUNTU, KUBUNTU, SUSE, OPENSUSE, where the 1st one is install in my personal notebooks/laptops, the 2nd one is install in my WIAS Dell XPS 15 (KDE provides a lot of good stuffs), while the other ones are install in computer servers at WIAS Berlin. Note that SUSE & OPENSUSE are produced by Germans¹.

Question 0.1. *Which resources I should use to learn deeply Linux in a fast but complete way?*

William Shotts. *The Linux Command Line: A Complete Introduction* (2e) seems a good start.

Question 0.2. *Which e-book readers should I use on Linux?*

¹In my own experiences, I believe that Germans & Frenches prefer building their own facilities over using those provided by the rest of the world.

Chapter 1

Shotts, 2019. The Linux Command Line: A Complete Introduction. 2e

About the Author. “William Shotts has been a software professional for > 30 years & an avid Linux user for > 20 years. He has an extensive background in software development, including technical support, quality assurance, & documentation. He is also the creator of [LinuxCommand.org](https://linuxcommand.org), a Linux education & advocacy site featuring news, reviews, & extensive support for using the Linux command line.” – Shotts, 2019, p. 6.

About the Technical Reviewer. “Jordi Gutiérrez Hermoso is a coder, mathematician, & hacker-errant. He runs Debian GNU/Linux exclusively since 2022, both at home & at work. Jordi has been involved with GNU Octave, a free numerical computing environment largely compatible with MATLAB, & with Mercurial, a distributed version control system. He enjoys pure & applied mathematics, skating, swimming, & knitting. Nowadays he thinks a lot about environmental mapping, greenhouse gas emissions, & rhino conservation efforts.” – Shotts, 2019, p. 7

Introduction

“I want to tell you a story. No, not the story of how, in 1991, Linus Torvalds wrote the 1st version of the Linux kernel. You can read that story in lots of Linux books. Nor am I going to tell you the story of how, some years earlier, Richard Stallman began the GNU Project to create a free Unix-like operating system. That’s an important story too, but most other Linux books have that one, as well.

No, I want to tell you the story of how you take back control of your computer.

When I began working with computers as a college student in the late 1970s, there was a revolution going on. The invention of the microprocessor¹ had made it possible for ordinary people like you & me to actually own a computer. It’s hard for many people today to imagine what the world was like when only big business & big government ran all the computers. Let’s just say, you couldn’t get much done.

Today, the world is very different. Computers are everywhere, from tiny wristwatches to giant data centers to everything in between. In addition to ubiquitous² computers, we also have a ubiquitous network connecting them together. This has created a wondrous³ new age of personal empowerment⁴ & creative freedom, but over the last couple of decades something else has been happening. A few giant corporations have been imposing their control over most of the world’s computers & deciding what you can & cannot do with them. Fortunately, people from all over the world are doing something about it. They are fighting to maintain control of their computers by writing their own software. They are building Linux.

Many people speak of “freedom” with regard to Linux, but I don’t think most people know what this freedom really means. Freedom is the power to decide what your computer does, & the only way to have this freedom is to know what your computer is doing. Freedom is a computer that is without secrets, one where everything can be known if you care enough to find out.” – Shotts, 2019, pp. 30–31

Why Use the Command Line?

“Have you ever noticed in the movies when the “superhacker” – you know, the guy who can break into the ultra-secure military computer in < 30 seconds – sits down at the computer, he never touches a mouse? It’s because filmmakers realize

¹**microprocessor** [n] (*computing*) a small unit of a computer that contains all the functions of the central processing unit.

²**ubiquitous** [a] present, appearing or found everywhere.

³**wondrous** [a] (*literary*) strange, beautiful, & impressive, SYNONYM: **wonderful**.

⁴**empowerment** [n] [uncountable] a positive feeling that you have some control over your life or the situation you are in.

that we, as human beings, instinctively know the only way to really get anything done on a computer is by typing on a keyboard!

Most computer users today are familiar only with the *graphical user interface* (GUI) & have been taught by vendors⁵ & pundits⁶ that the *command line interface* (CLI) is a terrifying thing of the past. This is unfortunate because a good command line interface is a marvelously⁷ expressive way of communicating with a computer in much the same way the written word is for human beings. It's been said that "graphical user interfaces make easy tasks easy, while command line interfaces make difficult tasks possible," & this is still very true today.

Since Linux is modeled after the Unix family of operating systems, it shares the same rich heritage of command line tools as Unix. Unix came into prominence⁸ during the early 1980s (although it was 1st developed a decade earlier), before the widespread adoption of the graphical user interface &, as a result, developed an extensive command line interface instead. In fact, 1 of the strongest reasons early adopters of Linux chose it over, say, Windows NT was the powerful command line interface that made the "difficult tasks possible." – Shotts, 2019, p. 31

What This Book Is About

"This book is a broad overview of "living" on the Linux command line. Unlike some books that concentrate on just a single program, such as the shell program `bash`, this book will try to convey how to get along with the command line interface in a larger sense. How does it all work? What can it do? What's the best way to use it?

This is not a book about Linux system administration. While any serious discussion of the command line will invariably lead to system administration topics, this book touches on only a few administration issues. It will, however, prepare the reader for additional study by providing a solid foundation in the use of the command line, an essential tool for any serious system administration task.

This book is Linux-centric. Many other books try to broaden their appeal by including other platforms such as generic Unix & macOS. In doing so, they "water down" their content to feature only general topics. This book, on the other hand, covers only contemporary Linux distributions. 95% of the content is useful for users of other Unix-like systems, but this book is highly targeted at the modern Linux command line user." – Shotts, 2019, p. 32

Who Should Read This Book

"This book is for new Linux users who have migrated from other platforms. Most likely you are a "power user" of some version of Microsoft Windows. Perhaps your boss has told you to administer a Linux server, or you're entering the exciting new world of single board computers (SBC) such as the Raspberry Pi. You may just be a desktop user who is tired of all the security problems & wants too give Linux a try. That's fine. All are welcome here.

That being said, there is no shortcut to Linux enlightenment. Learning the command line is challenging & takes real effort. It's not that it's so hard, but rather it's so *vast*. The average Linux system has literally *thousands* of programs you can employ on the command line. Consider yourself warned; learning the command line is not a casual⁹ endeavor¹⁰.

On the other hand, learning the Linux command line is extremely rewarding. If you think you're a "power user" now, just wait. You don't know what real power is – yet. &, unlike many other computer skills, knowledge of the command line is long-lasting. The skills learned today will still be useful 10 years from now. The command line has survived the test of time.

It is also assumed that you have no programming experience, but don't worry, we'll start you down that path as well." – Shotts, 2019, pp. 32–33

What's in This Book

"This material is presented in a carefully chosen sequence, much like a tutor sitting next to you guiding you along. Many authors treat this material in a "systematic" fashion, exhaustively¹¹ covering each topic in order. This makes sense from a writer's perspective but can be very confusing to new users.

⁵**vendor** [n] **1.** a person or company that sells things, especially outside on the street; **2.** (*formal*) a company that sells a particular product; **3.** (*law*) a person who is selling something, especially a house.

⁶**pandit** [n] (also **pundit**) **1.** a Hindu priest or wise man; **2.** (*Indian English*) a teacher; **3.** (*Indian English*) a musician with a lot of skill.

⁷**marvellously** [adv] (US English **marvelously**) very; very well, SYNONYM: **wonderfully**.

⁸**prominence** [n] **1.** [uncountable, singular] the state of being important, well known or easy to notice; **2.** [countable, uncountable] (*medical* or *formal*) a thing that sticks out from something; the fact or state of sticking out from something.

⁹**casual** [a] **1.** [usually before noun] without paying attention to detail; **2.** [usually before noun] not showing much care or thought; **3.** [usually before noun] (of a relationship) lasting only a short time & without deep affection; **4.** [usually before noun] (BE) (of work) not permanent; not regular; **5.** not formal; **6.** [only before noun] happening by chance; doing something by chance.

¹⁰**endeavour** [n] (US **endeavor**) (*formal*) **1.** [uncountable, countable] serious effort to achieve something; an attempt to do something, especially something new or difficult; **2.** [countable, usually plural] something that somebody does; [v] **endeavor to do something** (*formal*) to try hard to do or achieve something, SYNONYM: **strive**.

¹¹**exhaustive** [a] including everything possible; very thorough or complete.

Another goal is to acquaint¹² you with the Unix way of thinking, which is different from the Windows way of thinking. Along the way, we'll go on a few side trips to help you understand why certain things work the way they do & how they got that way. Linux is not just a piece of software; it's also a small part of the larger Unix culture, which has its own language & history. I might throw in a rant or 2, as well.

This book is divided into 4 parts, each covering some aspect of the command line experience.

- **Part 1, “Learning the Shell,”** starts our exploration of the basic language of the command line including such things as the structure of commands, file system navigation, command line editing, & finding help & documentation for commands.
- **Part 2, “Configuration & the Environment,”** covers editing configuration files that control the computer's operation from the command line.
- **Part 3, “Common Tasks & Essential Tools,”** explores many of the ordinary tasks that are commonly performed from the command line. Unix-like operating systems, such as Linux, contain many “classic” command line programs that are used to perform powerful operations on data.
- **Part 4, “Writing Shell Scripts,”** introduces shell programming, an admittedly rudimentary but easy-to-learn technique for automating many common computing tasks. By learning shell programming, you will become familiar with concepts that can be applied to many other programming languages.” – Shotts, 2019, pp. 33–34

How to Read This Book

“Start at the beginning of the book & follow it to the end. It isn't written as a reference work; it's really more like a story with a beginning, middle, & end.” – Shotts, 2019, p. 34

Prerequisites

“To use this book, all you will need is a working Linux installation. You can get this in 1 of 2 ways:

- **Install Linux on a (not so new) computer.** It doesn't matter which distribution you choose, though most people today start out with either Ubuntu, Fedora, or OpenSUSE. If in doubt, try Ubuntu 1st. Installing a modern Linux distribution can be ridiculously easy or ridiculously difficult depending on your hardware. I suggest a desktop computer that is a couple of years gold & has at least 2GB of RAM & 6GB of free hard disk space. Avoid laptops & wireless networks if at all possible, as these are often more difficult to get working.
- **Use a “live CD” or USB flash drive.** 1 of the cool things you can do with many Linux distributions is run them directly from a CD-ROM or USB flash drive without installing them at all. Just go into your BIOS setup & set your computer to boot from a CD-ROM drive or USB device & reboot. Using this method is a great way to test a computer for Linux compatibility prior to installation. The disadvantage is that it may be slow compared to having Linux installed on your hard drive. Both Ubuntu & Fedora (among others) have live versions.

Regardless of how you install Linux, you'll need to have occasional superuser (i.e., administrative) privileges to carry out the lessons in this book.

After you have a working installation, start reading & follow along with your own computer. Most of the material in this book is “hands on,” so sit down & get typing!

Why I don't call it “GNU/LINUX”. In some quarters, it's politically correct to call the Linux operating system the “GNU/Linux operating system.” The problem with “Linux” is that there is no completely correct way to name it because it was written by many different people in a vast, distributed development effort. Technically speaking, Linux is the name of the operating system's kernel, nothing more. The kernel is very important, of course, since it makes the operating system go, but it's not enough to form a complete operating system.

Enter Richard Stallman, the genius-philosopher who founded the Free Software movement, started the Free Software Foundation, formed the GNU Project, wrote the 1st version of the GNU C Compiler (`gcc`), created the GNU General Public License (the GPL), etc., etc., etc. He *insists* that you call it “GNU/Linux” to properly reflect the contributions of the GNU Project. While the GNU Project predates the Linux kernel & the project's contributions are extremely deserving of recognition, placing them in the name is unfair to everyone else who made significant contributions. Besides, I think “Linux/GNU” would be more technically accurate since the kernel boots 1st & everything else runs on top of it.

In popular usage, Linux refers to the kernel & all the other free & open source software found in the typical Linux distribution, i.e., the entire Linux ecosystem, not just the GNU components. The operating system marketplace seems to prefer 1-word names such as DOS, Windows, macOS, Solaris, Irix, & AIX. I have chosen to use the popular format. If, however, you prefer to use “GNU/Linux” instead, perform a mental search-&-replace while reading this book. I won't mind.” – Shotts, 2019, pp. 34–36

¹²**acquaint** [v] (*formal*) **acquaint somebody/yourself with something** to make somebody/yourself familiar with or aware of something.

What's New in the 2nd Edition

“While the basic structure & content remain the same, this edition of *The Linux Command Line* is peppered with various refinements, classifications, & modernizations, many of which are based on reader feedback. In addition, 2 particular improvements stand out. 1st, the book now assumes `bash` version 4.x, which was not in wide use at the time of the original manuscript. This 4th major version of `bash` added several useful new features now covered in this edition. 2nd, Part 4, “Shell Scripting,” has been improved to provide better examples of good scripting practice. The scripts included in Part 4 have been revised to make them more robust, & I also fixed a few bugs.” “This book is an ongoing project, like many open source software projects.” – Shotts, 2019, p. 36

Part I: Learning The Shell

1.1 What Is the Shell?

“When we speak of the command line, we are really referring to the *shell*. The shell is a program that takes keyboard commands & passes them to the operating system to carry out. Almost all Linux distributions supply a shell program from the GNU Project called `bash`. The name is acronym for *bourne-again shell*, a reference to the fact that `bash` is an enhanced replacement for `sh`, the original Unix shell program written by Steve Bourne.” – Shotts, 2019, p. 38

1.1.1 Terminal Emulators

“When using a graphical user interface (GUI), we need another program called a *terminal emulator* to interact with the shell. If we look through our desktop menus, we will probably find one. KDE uses `konsole`, & GNOME uses `gnome-terminal`, though it's likely called simply Terminal on your menu. A number of other terminal emulators are available for Linux, but they all basically do the same thing: give us access to the shell. You will probably develop a preference for 1 or another terminal emulator based on the number of bells & whistles it has.” – Shotts, 2019, p. 38

1.1.2 Making Your 1st Keystrokes

“Launch the terminal emulator. Once it comes up, we should see something like this: `[me@linuxbox ~]$`. This is called a *shell prompt*, & it will appear whenever the shell is ready to accept input. While it might vary in appearance somewhat depending on the distribution, it will typically include your `username@machinename`, followed by the current working directory (more about that in a little bit) & a dollar sign.

If the last character of the prompt is a hash mark `#` rather than a dollar sign, the terminal session has superuser privileges. This means either we are logged in as the root user or we selected a terminal emulator that provides superuser (administrative) privileges.” – Shotts, 2019, p. 39

1.1.3 Command History

“If we press the up arrow, we will see that the previous command entered reappears after the prompt. This is called *command history*. Most Linux distributions remember the last 1,000 commands by default. Press the down arrow & the previous command disappears.” – Shotts, 2019, p. 39

1.1.4 Cursor Movement

“Recall the previous command by pressing the up arrow again. If we try the left & right arrows, we'll see that we can position the cursor anywhere on the command line. This makes editing commands easy.

A few words about mice & focus. While the shell is all about the keyboard, you can also use a mouse with your terminal emulator. A mechanism built into the X Window System (the underlying engine that makes the GUI go) supports a quick copy-&-paste technique. If you highlight some text by holding down the left mouse button & dragging the mouse over it (or double-clicking a word), it is copied into a buffer maintained by X. Pressing the middle mouse button will cause the text to be pasted at the cursor location. Try it.

Don't be tempted to use `CTRL-C` & `CTRL-V` to perform copy & paste inside a terminal window. They don't work. These control codes have different meanings to the shell & were assigned many years before the release of Microsoft Windows.

Your graphical desktop environment (most likely KDE or GNOME), in an effort to behave like Windows, probably has its focus policy set to “click to focus.” This means for a window to get focus (become active), you need to click on it. This is contrary to the traditional X behavior of “focus follows mouse,” which means that a window gets focus just by passing the mouse over it. The window will not come to the foreground until you click on it, but it will be able to receive input. Setting

the focus policy to “focus follows mouse” will make the copy-&-paste technique even more useful. Give it a try if you can (some desktop environments such as Ubuntu’s Unity no longer support it). I think if you give it a chance, you will prefer it. You will find this setting in the configuration program for your window manager.” – Shotts, 2019, p. 40

1.1.5 Try Some Simple Commands

“Now that we have learned to enter text in the terminal emulator, let’s try a few simple commands. Let’s begin with the `date` command, which displays the current time & date.

```
[me@linuxbox ~]$ date
Fri Feb 2 15:09:41 EST 2018
```

A related command is `cal`, which, by default, displays a calendar of the current month.

```
[me@linuxbox ~]$ cal
February 2018
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28
```

The console behind the curtain. Even if we have no terminal emulator running, several terminal sessions continue to run behind the graphical desktop. We can access these sessions, called *virtual consoles*, by pressing CTRL-ALT-F1 through CTRL-ALT-F6 on most Linux distributions. When a session is accessed, it presents a login prompt into which we can enter our username & password. To switch from 1 virtual console to another, press `alt-F1` through `ALT-F6`. On most systems, we can return to the graphical desktop by pressing `alt-F7`.

To see the current amount of free space on our disk drives, enter `df`.

```
[me@linuxbox ~]$ df
tmpfs          783256      2288      780968    1% /run
/dev/nvme0n1p2 491039648 79254728 386768040 18% /
tmpfs          3916268     154944     3761324    4% /dev/shm
tmpfs          5120         4         5116     1% /run/lock
/dev/nvme0n1p1  94759       5329      89430     6% /boot/efi
tmpfs          783252       156      783096    1% /run/user/1000
```

Likewise, to display the amount of free memory, enter the `free` command.” – Shotts, 2019, pp. 41–42

```
[me@linuxbox ~]$ free
              total        used        free      shared  buff/cache   available
Mem:          7832540      6709784      217512      327756     905244     502532
Swap:           0           0           0
```

1.1.6 Ending a Terminal Session

“We can end a terminal session by closing the terminal emulator window, by entering the `exit` command at the shell prompt, or by pressing `ctrl-D`. [me@linuxbox ~]\$ `exit`.” – Shotts, 2019, p. 42

1.2 Navigation

“The 1st thing we need to learn (besides how to type) is how to navigate the file system on our Linux system. In this chapter, we will introduce the following commands: `pwd`: print name of current working directory, `cd`: change directory, `ls`: list directory contents.” – Shotts, 2019, p. 43

1.2.1 Understanding the File System Tree

“Like Windows, a Unix-like operating system such as Linux organizes its files in what is called a *hierarchical directory structure*. This means they are organized in a tree-like pattern of directories (sometimes called *folders* in other systems), which may contain files & other directories. The 1st directory in the file system is called the *root directory*. The root directory contains files & subdirectories, which contain more files & subdirectories, & so on.

Note that unlike Windows, which has a separate file system tree for each storage device, Unix-like systems such as Linux always have a single file system tree, regardless of how many drives or storage devices are attached to the computer. Storage devices are attached (or more correctly, *mounted*) at various points on the tree according to the whims of the *system administrator*, the person (or people) responsible for the maintenance of the system.” – Shotts, 2019, pp. 43–44

1.2.2 The Current Working Directory

“Most of us are probably familiar with a graphical file manager that represents the file system tree. Notice that the tree is usually shown upended, i.e., with the root at the top & the various branches descending below.

However, the command line has no pictures, so to navigate the file system tree, we need to think of it in a different way.

Imagine that the file system is a maze shaped like an upside-down tree & we are able to stand in the middle of it. At any given time, we are inside a single directory, & we can see the files contained in the directory & the pathway to the directory above us (called the *parent directory*) & any subdirectories below us. The directory we are standing in is called the *current working directory*. To display the current working directory, we use the `pwd` (print working directory) command.

```
[me@linuxbox ~]$ pwd
/home/me
```

When we 1st log in to our system (or start a terminal emulator session), our current working directory is set to our *home directory*. Each user account is given its own home directory, & it is the only place a regular user is allowed to write files.” – Shotts, 2019, pp. 44–45

1.2.3 Listing the Contents of a Directory

“To list the files & directories in the current working directory, we use the `ls` command.

```
[me@linuxbox ~]$ ls
Desktop Documents Music Pictures Public Templates Videos
```

Actually, we can use the `ls` command to list the contents of any directory, not just the current working directory, & there are many other fun things it can do as well. We’ll spend more time with `ls` in Chap. 3.” – Shotts, 2019, p. 45

1.2.4 Changing the Current Working Directory

“To change our working directory (where we are standing in the tree-shaped maze), we use the `cd` command. To do this, type `cd` followed by the pathname of the desired working directory. A pathname is the route we take along the branches of the tree to get to the directory we want. We can specify pathnames in 1 of 2 different ways: as *absolute pathnames* or as *relative pathnames*. Let’s deal with absolute pathnames 1st.” – Shotts, 2019, p. 45

1.2.5 Absolute Pathnames

“An absolute pathname begins with the root directory & follows the tree branch by branch until the path to the desired directory or file is completed. E.g., there is a directory on your system in which most of the system’s programs are installed. The directory’s pathname is `/usr/bin`. I.e., from the root directory (represented by the leading slash in the pathname) there is a directory called `usr` that contains a directory called `bin`.

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
[me@linuxbox bin]$ ls
...Listing of many, many files ...
```

Now we can see that we have changed the current working directory to `/usr/bin` & that it is full of files. Notice how the shell prompt has changed? As a convenience, it is usually set up to automatically display the name of the working directory.” – Shotts, 2019, p. 46

1.2.6 Relative Pathnames

“Where an absolute pathname starts from the root directory & leads to its destination, a relative pathname starts from the working directory. To do this, it uses a couple of special notations to represent relative positions in the file system tree. These special notations are `.` (dot) & `..` (dot dot).

The `.` notation refers to the working directory, & the `..` notation refers to the working directory’s parent directory. Here is how it works. Let’s change the working directory to `/usr/bin` again.

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

Now let's say that we wanted to change the working directory to the parent of `/usr/bin`, which is `/usr`. We could do that 2 different ways, either with an absolute pathname:

```
[me@linuxbox bin]$ cd /usr
[me@linuxbox usr]$ pwd
/usr
```

or with a relative pathname:

```
[me@linuxbox bin]$ cd ..
[me@linuxbox usr]$ pwd
/usr
```

2 different methods with identical results. Which one should use? ¹³

Likewise, we can change the working directory from `/usr` to `/usr/bin` in 2 different ways, either using an absolute pathname:

```
[me@linuxbox usr]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

or using a relative pathname:

```
[me@linuxbox usr]$ cd ./bin
[me@linuxbox bin]$ pwd
/usr/bin
```

Now, there is something important to point out here. In almost all cases, we can omit the `./` part because it is implied. Typing the following does the same thing:

```
[me@linuxbox usr]$ cd bin
```

In general, if we do not specify a pathname to something, the working directory will be assumed.

Important facts about filenames. On Linux systems, files are named in a manner similar to that of other systems such as Windows, but there are some important differences.

- Filenames that begin with a period character are hidden. This only means that `ls` will not list them unless you say `ls -a`. When your account was created, several hidden files were placed in your home directory to configure things for your account. In Chap. 11 we will take a closer look at some of these files to see how you can customize your environment. In addition, some applications place their configuration & setting files in your home directory as hidden files.
- Filenames & commands in Linux, like Unix, are case sensitive. The filenames `File1` & `file1` refer to different files.
- Though Linux supports long filenames that may contain embedded spaces & punctuation characters, limit the punctuation characters in the names of files you create to period, dash, & underscore. *Most important, do not embed spaces in filenames.* If you want to represent spaces between words in a filename, use underscore characters. You will thank yourself later.
- Linux has no concept of a “file extension” like some other operating systems. You may name files any way you like. The contents or purpose of a file is determined by other means. Although Unix-like operating systems don't use file extensions to determine the contents/purpose of files, many application programs do.” – Shotts, 2019, pp. 46–48

1.2.7 Some Helpful Shortcuts

Some useful ways to quickly change the current working directory. CD Shortcuts: • `cd`: Changes the working directory to your home directory. • `cd -`: Changes the working directory to the previous working directory. • `cd ~user_name`: Changes the working directory to the home directory of `user_name`.

¹³The one that requires the least typing!

1.3 Exploring the System

“Now that we know how to move around the file system, it’s time for a guided tour of our Linux system. Before we start, however, we’re going to learn some more commands that will be useful along the way. • **ls**: List directory contents. • **file**: Determine file type. • **less**: View file contents.” – Shotts, 2019, p. 50

1.3.1 More Fun with ls

“The **ls** command is probably the most used command, & for good reason. With it, we can see directory contents & determine a variety of important file & directory attributes. As we have seen, we can simply enter **ls** to get a list of files & subdirectories contained in the current working directory.

```
[me@linuxbox ~]$ ls
Desktop Documents Music Pictures Public Templates Videos
```

Besides the current working directory, we can specify the directory to list, like so:

```
me@linuxbox ~]$ ls /usr
bin games include lib local sbin share src
```

We can even specify multiple directories. In the following example, we list both the user’s home directory (symbolized by the `~` character) & the `/usr` directory:

```
[me@linuxbox ~]$ ls ~ /usr
/home/me:
Desktop Documents Music Pictures Public Templates Videos
/usr:
bin games include lib local sbin share src
```

We can also change the format of the output to reveal more detail.

```
[me@linuxbox ~]$ ls -l
total 56
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Desktop
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Documents
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Music
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Pictures
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Public
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Templates
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Videos
```

By adding `-l` to the command, we changed the output to the long format.” – Shotts, 2019, pp. 50–51

1.3.2 Options & Arguments

“This brings us to a very important point about how most commands work. Commands are often followed by 1 or more *options* that modify their behavior &, further, by 1 or more *arguments*, the items upon which the command acts. So, most commands look kind of like this: **command -options arguments**. Most commands use options, which consist of a single character preceded by a dash, e.g., `-l`. Many commands, however, including those from the GNU Project, also support *long options*, consisting of a word preceded by 2 dashes. Also, many commands allow multiple short options to be strung together. In the following example, the **ls** command is given 2 options, which are the `l` option to produce long format output, & the `t` option to sort the result by the file’s modification time.

```
[me@linuxbox ~]$ ls -lt
```

We’ll add the long option `--reverse` to reverse the order of the sort.

```
[me@linuxbox ~]$ ls -lt --reverse
```

Remark 1.1. *Command options, like filenames in Linux, are case sensitive.*

The **ls** command has a large number of possible options, the most common of which are listed in Table 3.1. Common **ls** Options. Option : Long option : Description.

- `-a`: `--all`: List all files, even those with names that begin with a period, which are normally not listed (i.e., hidden).

- **-A: --almost-all:** Like the **-a** option except it does not list. (current directory) & **..** (parent directory).
- **-d: --directory:** Ordinarily, if a directory is specified, **ls** will list the contents of the directory, not the directory itself. Use this option in conjunction with the **-l** option to see details about the directory rather than its contents.
- **-F: --classify:** This option will append an indicator character to the end of each listed name. E.g., it will append a forward slash **/** if the name is a directory.
- **-h: --human-readable:** In long format listings, display file sizes in human-readable format rather than in bytes.
- **-l:** Display results in long format.
- **-r: --reverse:** Display the results in reverse order. Normally, **ls** displays its results in ascending alphabetical order.
- **-s:** Sort results by file size.
- **-t:** Sort by modification in time.” – Shotts, 2019, pp. 51–53

1.3.3 A Longer Look at Long Format

As we saw earlier, the **-l** option causes **ls** to display its results in long format. This format contains a great deal of useful information. Here is the **Examples** directory from an Ubuntu system:

```
-rw-r--r-- 1 root root 3576296 2017-04-03 11:05 Experience ubuntu.ogg
-rw-r--r-- 1 root root 1186219 2017-04-03 11:05 kubuntu-leaflet.png
-rw-r--r-- 1 root root 47584 2017-04-03 11:05 logo-Edubuntu.png
-rw-r--r-- 1 root root 44355 2017-04-03 11:05 logo-Kubuntu.png
-rw-r--r-- 1 root root 34391 2017-04-03 11:05 logo-Ubuntu.png
-rw-r--r-- 1 root root 32059 2017-04-03 11:05 oo-cd-cover.odf
-rw-r--r-- 1 root root 159744 2017-04-03 11:05 oo-derivatives.doc
-rw-r--r-- 1 root root 27837 2017-04-03 11:05 oo-maxwell.odt
-rw-r--r-- 1 root root 98816 2017-04-03 11:05 oo-trig.xls
-rw-r--r-- 1 root root 453764 2017-04-03 11:05 oo-welcome.odt
-rw-r--r-- 1 root root 358374 2017-04-03 11:05 ubuntu Sax.ogg
```

Table 3-2 provides us with a look at the different fields from 1 of the files & their meanings. Table 3-2: **ls** Long Listing Fields. Field : Meaning.

- **-rw-r--r--:** Access rights to the file. The 1st character indicates the type of file. Among the different types, a leading dash means a regular file, while a **d** indicates a directory. The next 3 characters are the access rights for the file’s owner, the next 3 are for members of the file’s group, & the final 3 are for everyone else.
- **1:** File’s number of hard links.
- **root:** The username of the file’s owner.
- **root:** The name of the group that owns the file.
- **32059:** Size of the file in bytes.
- **2017-04-03 11:05:** Date & time of the file’s last modification.
- **oo-cd-cover.odf:** Name of the file.” – Shotts, 2019, pp. 53–54

1.3.4 Determining a File’s Type with file

“As we explore the system, it will be useful to know what files contain. To do this, we will use the **file** command to determine a file’s type. As we discussed earlier, filenames in Linux are not required to reflect a file’s contents. While a filename like **picture.jpg** would normally be expected to contain a JPEG-compressed image, it is not required to in Linux. We can invoke the **file** command this way: **file filename**. When invoked, the **file** command will print a brief description of the file’s contents. E.g.:

```
[me@linuxbox ~]$ file picture.jpg
picture.jpg: JPEG image data, JFIF standard 1.01
```

There are many kinds of files. In fact, 1 of the common ideas in Unix-like operating systems such as Linux is that “everything is a file.” As we proceed with our lessons, we will see just how true that statement is.

While many of the files on our system are familiar, e.g., MP3 & JPEG files, there are many kinds that are a little less obvious & a few that are quite strange.” – Shotts, 2019, pp. 54–55

1.3.5 Viewing File Contents with `less`

“The `less` command is a program to view text files. Throughout our Linux system, there are many files that contain human-readable text. The `less` program provides a convenient way to examine them.

Why would we want to examine text files? Because many of the files that contain system settings (called *configuration files*) are stored in this format, & being able to read them gives us insight about how the system works. In addition, some of the actual programs that the system uses (called *scripts*) are stored in this format. In later chapters, we will learn how to edit text files in order to modify system settings & write our own scripts, but for now we will just look at their contents.

The `less` command is used like this: `less filename`.

What is “text”? There are many ways to represent information on a computer. All methods involve defining a relationship between the information & some numbers that will be used to represent it. Computers, after all, understand only numbers, & all data is converted to numeric representation.

Some of these representation systems are very complex (such as compressed video files), while others are rather simple. 1 of the earliest & simplest is called ASCII text. ASCII (pronounced “as-key”) is short for American Standard Code for Information Interchange. This is a simple encoding scheme that was 1st used on Teletype machines to map keyboard characters to numbers.

Text is a simple 1-to-1 mapping of characters to numbers. It is very compact. 50 characters of text translates to 50 bytes of data. It is important to understand that text only contains a simple mapping of characters to numbers. It is not the same as a word processor document such as one created by Microsoft Word or LibreOffice Writer. Those files, in contrast to simple ASCII text, contain many non-text elements that are used to describe its structure & formatting. Plain ASCII text files contain only the characters themselves & a few rudimentary control codes such as tabs, carriage returns, & line feeds.

Throughout a Linux system, many files are stored in text format, & there are many Linux tools that work with text files. Even Windows recognizes the importance of this format. The well-known NOTEPAD.EXE program is an editor for plain ASCII text files.

Once started, the `less` program allows us to scroll forward & backward through a text file. E.g., to examine the file that defines all the system’s user accounts, enter the following command:

```
[me@linuxbox ~]$ less /etc/passwd
```

Once the `less` program starts, we can view the contents of the file. If the file is longer than 1 page, we can scroll up & down. To exit `less`, press `q`. Table 3-3 lists the most common keyboard commands used by `less`. **Table 3-3: `less` Commands.**

- | Command | Action |
|-------------------------------------|--|
| • PAGE UP or <code>b</code> | Scroll back 1 page. |
| • PAGE DOWN or space | Scroll forward 1 page. |
| • Up arrow | Scroll up 1 line. |
| • Down arrow | Scroll down 1 line. |
| • <code>G</code> | Move to the end of the text file. |
| • <code>1G</code> or <code>g</code> | Move to the beginning of the text file. |
| • <code>/characters</code> | Search forward to the next occurrence of <code>characters</code> . |
| • <code>n</code> | Search for the next occurrence of the previous search. |
| • <code>h</code> | Display help screen. |
| • <code>q</code> | Quit <code>less</code> . |

Less is more. The `less` program was designed as an improved replacement of an earlier Unix program called *more*. The name *less* is a play on the phrase “less is more” – a motto of modernist architects & designers.

`less` falls into the class of programs called *papers*, programs that allow the easy viewing of long text documents in a page-by-page manner. Whereas the *more* program could only page forward, the `less` program allows paging both forward & backward & has many other features as well.” – Shotts, 2019, pp. 55–57

1.3.6 Taking a Guided Tour

“The file system layout on a Linux system is much like that found on other Unix-like systems. The design is actually specified in a published standard called the *Linux Filesystem Hierarchy Standard*. Not all Linux distributions conform to the standard exactly, but most come pretty close.

Remember the copy-&-paste trick! If you are using a mouse, you can double-click a filename to copy it & middle-click to paste it into commands.

Next, we are going to wander around the file system ourselves to see what makes our Linux system tick. This will give us a chance to practice our navigation skills. 1 of the things we will discover is that many of the interesting files are in plain human-readable text. As we go about our tour, try the following: • `cd` into a given directory. • List the directory contents with `ls -l`. 1. If you see an interesting file, determine its contents with `file`. 2. If it looks like it might be text, try viewing it with `less`. If we accidentally attempt to view a non-text file & it scrambles the terminal window, we can recover by entering the `reset` command.

As we wander around, don’t be afraid to look at stuff. Regular users are largely prohibited from messing things up. That’s the system administrator’s job! If a command complains about something, just move on to something else. Spend some time looking around. The system is ours to explore. Remember, in Linux, there are no secrets!

Table 3-4 lists just a few of the directories we can explore. There may be some slight differences depending on our Linux distribution. Don’t be afraid to look around & try more! Table 3-4: Directories Found on Linux Systems. Directory : Comments.

- `/`: The root directory, where everything begins.
- `/bin`: Contains binaries (programs) that must be present for the system to boot & run.
- `/boot`: Contains the Linux kernel, initial RAM disk image (for drivers needed at boot time), & the boot loader. Interesting files include `/boot/grub/grub.conf`, or `menu.lst`, which is used to configure the boot loader, & `/boot/vmlinuz` (or something similar), the Linux kernel.
- `/dev`: This is a special directory that contains *device nodes*. “Everything is a file” also applies to devices. Here is where the kernel maintains a list of all the devices it understands.
- `/etc`: The `/etc` directory contains all the system-wide configuration files. It also contains a collection of shell scripts that start each of the system services at boot time. Everything in this directory should be readable text. While everything in `/etc` is interesting, here are some all-time favorites: `/etc/crontab`, a file that defines when automated jobs will run; `/etc/fstab`, a table of storage devices & their associated mount points; & `/etc/passwd`, a list of the user accounts.
- `/home`: In normal configurations, each user is given a directory in `/home`. Ordinary users can write files only in their home directories. This limitation protects the system from errant¹⁴ user activity.
- `/lib`: Contains shared library files used by the core system programs. These are similar to dynamic link libraries (DLLs) in Windows.
- `/lost+found`: Each formatted partition or device using a Linux file system, such as `ext3`, will have this directory. It is used in the case of a partial recovery from a file system corruption event. Unless something really bad has happened to your system, this directory will remain empty.
- `/media`: On modern Linux systems, the `/media` directory will contain the mount points for removable media such as USB drives, CD-ROMs, & so on, that are mounted automatically at insertion.
- `/mnt`: On older Linux systems, the `/mnt` directory contains mount points for removable devices that have been mounted manually.
- `/opt`: The `/opt` directory is used to install “optional” software. This is mainly used to hold commercial software products that might be installed on the system.
- `/proc`: The `/proc` directory is special. It’s not a real file system in the sense of files stored on your hard drive. Rather, it is a virtual file system maintained by the Linux kernel. The “files” it contains are peepholes¹⁵ into the kernel itself. The files are readable & will give you a picture of how the kernel sees your computer.
- `/root`: This is the home directory for the root account.

¹⁴**errant** [a] [only before noun] (*formal or humorous*) 1. doing something that is wrong; not behaving in an acceptable way; 2. (of a husband or wife) not sexually faithful.

¹⁵**peephole** [n] a small opening in a wall, door, etc. that you can look through.

- **/sbin**: This directory contains “system” binaries. These are programs that perform vital system tasks that are generally reserved for the superuser.
- **/tmp**: The **/tmp** directory is intended for the storage of temporary, transient files created by various programs. Some configurations cause this directory to be emptied each time the system is rebooted.
- **/usr**: The **/usr** directory tree is likely the largest one on a Linux system. It contains all the programs & support files used by regular users.
- **/usr/bin**; **/usr/bin** contains the executable programs installed by your Linux distribution. It is not uncommon for this directory to hold thousands of programs.
- **/usr/lib**: The shared libraries for the programs in **/usr/bin**.
- **/usr/local**: The **/usr/local** tree is where programs that are not included with your distribution but are intended for system-wide use are installed. Programs compiled from source code are normally installed in **/usr/local/bin**. On a newly installed Linux system, this tree exists, but it will be empty until the system administrator puts something in it.
- **/usr/sbin**: Contains more system administration programs.
- **/usr/share**: **/usr/share** contains all the shared data used by programs in **/usr/bin**. This includes things such as default configuration files, icons, screen backgrounds, sound files, & so on.
- **/usr/share/doc**: Most packages installed on the system will include some kind of documentation. In **/usr/share/doc**, we will find documentation files organized by package.
- **/var**: With the exception of **/tmp** & **/home**, the directories we have looked at so far remain relatively static; i.e., their contents don’t change. The **/var** directory tree is where data that is likely to change is stored. Various databases, spool files, user mail, & so forth, are located here.
- **/var/log**: **/var/log** contains *log files*, records of various system activity. These are important & should be monitored from time to time. The most useful ones are **/var/log/messages** & **/var/log/syslog**. Note that for security reasons on some systems, you must be the superuser to view log files.” – Shotts, 2019, pp. 57–61

1.3.7 Symbolic Links

“As we look around, we are likely to see a directory listing (e.g., **/lib**) with an entry like this:

```
lrwxrwxrwx 1 root root    11 2018-08-11 07:34 libc.so.6 -> libc-2.6.so
```

Notice how the 1st letter of the listing is **l** & the entry seems to have 2 filenames? This is a special kind of a file called a *symbolic link* (also known as a *soft link* or *symlink*). In most Unix-like systems, it is possible to have a file referenced by multiple names. While the value of this might not be obvious, it is really a useful feature.

Picture this scenario: a program requires the use of a shared resource of some kind contained in a file named “foo,” but “foo” has frequent version changes. It would be good to include the version number in the filename so the administrator or other interested party could see what version of “foo” is installed. This presents a problem. If we change the name of the shared resource, we have to track down every program that might use it & change it to look for a new resource name every time a new version of the resource is installed. That doesn’t sound like fun at all.

Here is where symbolic links save the day. Suppose we install version 2.6 of “foo,” which has the filename “foo-2.6,” & then create a symbolic link simply called “foo” that points to “foo-2.6.” This means that when a program opens the file “foo,” it is actually opening the file “foo-2.6.” Now everybody is happy. The programs that rely on “foo” can find it, & we can still see what actual version is installed. What if it is time to upgrade to “foo-2.7,” we just add the file to our system, delete the symbolic link “foo,” & create a new one that points to the new version. Not only does this solve the problem of the version upgrade, it also allows us to keep both versions on our machine. Imagine that “foo-2.7” has a bug (damn those developers!), & we need to revert to the old version. Again, we just delete the symbolic link pointing to the new version & create a new symbolic link pointing to the old version.

The directory listing at the beginning of this section (from the **/lib** directory of a Fedora system) shows a symbolic link called **libc.so.6** that points to a shared library file called **libc-2.6.so**. This means that programs looking for **libc.so.6** will actually get the file **libc-2.6.so**.” – Shotts, 2019, pp. 61–63

1.3.8 Hard Links

“While we are on the subject of links, we need to mention that there is a 2nd type of link called *hard links*. Hard links also allow files to have multiple names, but they do it in a different way.” – Shotts, 2019, p. 63

1.3.9 Summing Up

“With our tour behind us, we have learned a lot about our system. We’ve seen various files & directories & their contents. 1 thing you should take away from this is how open the system is. In Linux there are many important files that are plain human-readable text. Unlike many proprietary systems, Linux makes everything available for examination & study.” – Shotts, 2019, p. 63

1.4 Manipulating Files & Directories

1.5 Working with Commands

1.6 Redirection

1.7 Seeing the World as the Shell Sees It

1.8 Advanced Keyboard Tricks

1.9 Permissions

1.10 Processes

Part II: Configuration & The Environment

1.11 The Environment

1.12 A Gentle Introduction to vi

1.13 Customizing the Prompt

Part III: Common Tasks & Essential Tools

- 1.14 Package Management
- 1.15 Storage Media
- 1.16 Networking
- 1.17 Searching for Files
- 1.18 Archiving & Backup
- 1.19 Regular Expressions
- 1.20 Text Processing
- 1.21 Formatting Output
- 1.22 Printing
- 1.23 Compiling Programs

Part IV: Writing Shell Scripts

- 1.24 Writing Your 1st Script
- 1.25 Starting a Project
- 1.26 Top-Down Design
- 1.27 Flow Control: Branching with `if`
- 1.28 Reading Keyboard Input
- 1.29 Flow Control: Looping with `while/until`
- 1.30 Troubleshooting
- 1.31 Flow Control: Branching with `case`
- 1.32 Positional Parameters
- 1.33 Flow Control: Looping with `for`
- 1.34 Strings & Numbers
- 1.35 Arrays
- 1.36 Exotica

Chapter 2

Miscellaneous

2.1 Calibre

Official website: <https://calibre-ebook.com/>. See also, e.g., [Wikipedia/Calibre \(software\)](#). Enable Dark Mode in Calibre:

```
$ sudo nano /etc/profile.d/calibre.sh
```

```
export CALIBRE_USE_SYSTEM_THEME=1
```

```
$ sudo service gdm restart
```

& further customization. [inserting]

Bibliography

Shotts, William (2019). “The Linux Command Line: A Complete Introduction”. In: p. 640.