Awesome Command Line

21 Commands to Get You Started With Your Computer's Magic Portal!

Christopher S. Jones

Awesome Command Line
by Christopher S. Jones

Published by Christopher S. Jones, Post Office Box 7201, Boulder, CO 80306

November 2025: First Edition

Typeset by the author using the Asciidoc® Language™ [https://asciidoc.org][1] and a sprinkling of custom Ruby code. All source code for the book is available as a *free and open source software project* [https://github.com/christopher-s-jones/awesome-command-line]. The body text font is EB Garamond by Georg Duffner and Octavio Pardo. The heading font is PT Sans Narrow by Alexandra Korolkova, Olga Umpeleva and Vladimir Yefimov, and released by ParaType. The fixed width code font is Adobe Source Code Pro by Paul D. Hunt. The cover page title font is Qwitcher Grypen by Robert Leuschke, with PT Sans Narrow. Illustrations and logo design are by the author using Sketch.

[1] AsciiDoc® and AsciiDoc Language™ are trademarks of the Eclipse Foundation, Inc. Asciidoctor® is a trademark owned by Dan Allen, Sarah White, and the project's individual contributors.

To Kimberly, with love and gratitude.
Thank you for leading the Dawn.

CONTENTS

Preface **1**

1. The Command Line Is For Everyone **3**

Getting Started 3
Terminal Applications 5
The Shell 16
The Command Prompt 17
The Parts of a Command 18
Single Line and Multi-Lined Commands 21
Command Line Interfaces are Awesome! 24

2. Core Commands: pwd, man, clear, cd, ls **25**

For the Love of Files and Folders 25
The pwd Command—Print the Working Directory 26
File and Directory Paths 27
The man Command—Accessing the Manual 30
The clear Command—Keeping It Tidy 35
The cd Command—Changing Directories 37
Tab Completion—Give Your Fingers A Rest 39
The ls Command—Listing Files and Folders 42
Core Commands Are Awesome! 51

3. File Commands: echo, mv, cp, rm, nano **52**

Powerful Ways To Work With Files 52
The echo Command—Easily Creating Files 53
Redirection, Standard Input, Output, and Error 54
The mv Command—Renaming and Relocating Files 56
The cp Command—Copying Files 58
The rm Command—Deleting Files 60
The nano Command—Creating and Editing Files 62
Command Line File Handling is Awesome! 66

4. Folder Commands: mkdir, rmdir, du 67

Lightning Fast Folder Management 67
The mkdir Command—Creating Directories 67
Expansion—Powerful Techniques To Speed Up Your Commands 69
The rmdir Command—Deleting Directories 72
The du Command—Viewing Disk Usage 77
Command Line Folder Handling is Awesome! 80

5. Text Data Commands: cat, sort, head, tail, grep 81

Versatile Ways To Work With Text Data 81
The cat Command—Viewing and Combining Files 82
Pipes—The Power Of Compound Commands 87
The sort Command—Sorting the Contents of a File 88
The head Command—Previewing the Top of a File 91
The tail Command—Previewing the Bottom of a File 97
The grep Command—Filtering Data 100
Command Line Data Handling is Awesome! 105

6. Utility Commands: less, history, open 106

Utilities That Make Your Life Easier 106
The less Command—Paging Output for Easy Viewing 106
The history Command—Using Your Command History 106
The open Command—Opening Files and Folders 107
Command Line Utilities are Awesome! 108

7. Next Steps **109**

Practice Makes Perfect! 109
Upgrade Your Terminal Colors and Prompt 109
Explore the Universe of Commands 109
Congratulations! 110
You Are Awesome! 111

Appendix A: Customizing Your Terminal **112**

Appendix B: Using a Package Manager **114**

Installing More Commands On Your Computer 114
For Mac—Installing Homebrew 114
For Windows and Linux—Using The Built-In Package Manager 114

Appendix C: Regular Expressions—Matching Patterns Like a Pro **116**

Welcome to *Awesome Command Line*! I am so grateful that you have found this book, and that you're interested in exploring the seemingly secret world of computing from the command line!

This book is a reminder that you have superpowers at your fingertips! We are all familiar with the ways that we interact with our computers on a daily basis — *click*, *drag*, *drop*, *scroll*, *touch*, *tap*. The graphical nature of our screens and applications are absolutely amazing and indispensible. In this book, we explore the secret little gems available on all operating systems with another familiar way to interact — *type*! And the best part is that the magical commands we explore are largely derived from the English words that describe them, so acquainting ourselves with these commands is straight-forward with a bit of guidance. Once you become familiar with this small set of essential commands, adding new commands to your toolbox is even easier. With some simple dedication and practice, you will be able to enhance your computing workflow to be even more efficient and powerful. You'll manage your projects in ways that would otherwise be labor intensive when using your mouse. The intention of this book is to empower you in your creative journey by showing you how the *command line* is an awesome tool to get things done!

The UNIX® computer operating system developed by Bell Laboratories in the 1970s and 1980s was groundbreaking technology. When I first learned about *Linux*®—a Unix-inspired operating system for personal computers, I was fascinated by the freedom of it, and I spent a substantial amount of time figuring out how to work with it via text commands. Apple® transitioned to a Unix-like operating system with *Mac OS X*, and Microsoft® introduced the Windows® Subsystem for Linux to bring a Unix-like environment to Windows®. I've been excited about these moves because we now have access to these incredible tools in very polished desktop environments. My intention with this book is to save you as much start-up time and energy as possible in learning the command line, and to empower you to get the most out of the operating system of your choice, be it *macOS*®, *Linux*®, or *Microsoft*® *Windows*®. Have fun with it![1]

Who Should Read This Book

This book is written for everyone who loves computing! It's written as an introductory book to get you started with the infinite universe of text commands. The twenty one commands that I highlight are essential to building your foundation, and the command line concepts throughout each chapter solidify it further. This is intended as a launching point for everyone who likes to be organized and productive. Those in creative works—*writers*, *artists*, *designers*, *photographers*, and *editors*, and those beginning their journey with computers—*students*, early career *scientists*, *librarians*, aspiring *engineers*, *software developers*, *system administrators*, and *project managers*, can all learn the foundations and then pick up the commands that are specific to their course of study, craft, or industry. Those interested in joining the *decentralized web by running their own Bitcoin or Lightning node, Nostr relay, and other freedom technologies* will benefit from knowing these essentials. No matter your interests, this book is for those who love to learn!

Conventions Used in This Book

The following informational icons are used throughout the book:

This icon indicates an informational note.

This icon indicates a tip, suggestion, or a point of awesomeness.

This icon indicates a time to be very careful with a command or action. Many commands are extremely powerful, and the results may be irreversible.

Getting the Command Line Examples

The examples in this book are accessible for download from a link on the book release page [https://github.com/christopher-s-jones/awesome-command-line]. The entire book source code is released as an open source project under the Creative Commons Attribution-ShareAlike 4.0 International license, and may be used under those terms.

[1] UNIX is a registered trademark of The Open Group. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. Apple, Mac, and macOS are trademarks of Apple Inc., registered in the U.S. and other countries and regions. Microsoft and Microsoft Windows are trademarks of the Microsoft group of companies. References to companies or trademarked products within this work are provided for informational purposes only and do not imply endorsement, sponsorship, or affiliation by those companies. The inclusion of such references is solely for the purpose of illustration or commentary, and the author and publisher do not intend to suggest any formal relationship or approval by the respective trademark holders.

The Command Line Is For Everyone

Getting Started

The *command line* is a tool of empowerment—a magic portal to your computer that helps you navigate, organize, create, and refine your files on your computer. It also helps you automate and manage tasks running on your computer. When repeatedly clicking and choosing with your mouse becomes tedious, the command line *prompt* allows you to free yourself and just issue a powerful command that does all that you want, quickly and with ease, so you can get back to the creative side of your work. You can turn complex tasks that require a series of many steps into one workflow that accomplishes what was once inconvenient. Each of the major computer operating systems—including macOS, Linux, and Windows—all provide ways to use the command line[1]. While it is an extremely helpful tool on your local desktop or laptop machine, the command line is essential for running remote services, such as a website. This idea in itself is magical! The Internet predominantly runs on Linux-based servers in remote data centers around the world, and access to these servers largely happens through the command line. Can you imagine that? Managing computers remotely around the world? Yes! Thank you command line! We will be focusing on using the command line locally on your computer and how it can be a boost in your workflow, while laying the groundwork for you to expand your computing skills even further.

This book serves as an introduction to the command line and how to get started with it. We first get set up with the amazing tool you need to use the command line—a *terminal* application that knows how to handle your commands. But wait, you say—What is a *terminal*? What is a *command*? What is a *prompt*? In this chapter we introduce these and other concepts in a step by step manner so you have a solid understanding of the terms and lingo needed to use this magic little gem that may be unfamiliar now, but will soon be your trusty friend at your side. In the subsequent chapters we dive in with hands-on learning by highlighting twenty-one essential commands that build your command line foundation. Take a look at *Table 1*! It shows our road map of the commands we cover in each chapter.

Table 1. The 21 commands covered in the book, by chapter and category.

Chapter 2 Core Commands	Chapter 3 File Commands	Chapter 4 Folder Commands	Chapter 5 Text Data Commands	Chapter 6 Utility Commands
`pwd`	`echo`	`mkdir`	`cat`	`less`
`man`	`mv`	`rmdir`	`sort`	`history`
`clear`	`cp`	`du`	`head`	`open`
`cd`	`rm`		`tail`	
`ls`	`nano`		`grep`	

Each chapter builds on the previous. In addition to each command, we learn key concepts along the way that make using the command line progressively more powerful. These concepts include *file and directory paths*, *tab completion*, *redirection*, *expansion*, and *pipes*. Before we get into the details, let's first take a step back and think about the metaphors we use to interact with our computers.

The graphical desktop metaphor

When we turn on our computers, we are greeted with beautiful desktop images, icons, windows, menus, a pointer, and other *graphical* symbols that help us navigate our system and enable our digital creativity. Collectively, these components are known as the *graphical user interface*, or *GUI* for short. We predominantly work with our computers in this manner, and for good reason—it works really well! So think of the graphical user interface as an analogy, or metaphor, that helps us work with the underlying hardware of our machines—processors, storage drives, memory, etc. For instance, a *folder* icon on your Desktop represents a collection of *files*, and a *file* icon represents some data stored on your computer's drive. These metaphors are highly intuitive, but there are times when graphical tools slow us down—usually when you need to work on something repeatedly, remotely, when working with complex or large amounts of data, or when there is currently no way to do what you want using the graphical interface.

The command line metaphor

A *command line interface*, or *CLI*, is also a way to work with your computer by means of a metaphor, but in this case we use plain text words and other combinations of characters as symbols to indicate to the computer what you want to accomplish. Sometimes they resemble English words, other times they are shortened versions of words, or acronyms of multiple words. For example, a command that we cover in Chapter 2 is `pwd`. This is just a three-character command that stands for "print working

directory". Here's a quick example:

```
pwd           <-- This is the command
/Users/chris  <-- This is the result of the command
```

What a little gem! We'll go into the details of this command later, but you can see that it is a very simple command that shows you what directory (i.e. folder) you are currently working in by dispaying the `/Users/chris` text below the command. So the combination of those three letters, when typed in an application that knows how to handle them (a terminal), will give you back a result that shows your current folder, all using text-based symbols.

In the remainder of Chapter 1, we will get started with setting up a terminal application that provides a command line interface for your operating system (macOS, Linux, or Windows). Once complete, we will open the terminal application, adjust the font family and size so that it is comfortably readable, and then introduce the various components—the *shell*, the *command line*, the *command prompt*, and the various parts of a *command*. Let's get rolling!

Terminal Applications

A computer *terminal* is merely a way to send input to a computer and receive the output results. You might also hear the word *console* used interchangeably with terminal, but the latter is a bit more specific in that it was historically a screen device and keyboard connected directly to a mainframe computer and used by the operators. Terminals, on the other hand, were connected to computers remotely on a network. We now of course have terminal applications that are considered *virtual terminals* that emulate these older physical terminals. See *A History of Modern Computing* (2003) by Paul E. Ceruzzi [2] for more background. There are many alternative terminal applications to choose from, but we will start with the default applications on each operating system in order to get set up. For Mac, the default application is called *Terminal.app*, and on Windows 11 we will focus on the default *Windows Terminal*. If you are using Linux, there are many distributions available, but we will focus on Ubuntu®[3] 24.04 and the *Gnome Terminal* that comes packaged with it. Skip to the next section that is pertinent to you for your operating system and we'll get started with a terminal application!

- Setup for Mac
- Setup for Linux
- Setup for Windows

Setup for Mac

If you are on a Mac, Apple has included the Terminal.app since it introduced Mac OS X in 2001, so it has had many years of refinement. You can search for "Terminal.app" using Apple's Spotlight search by pressing the `Command`+`Spacebar` keys at the same time and typing "terminal" (without the quotes) in the search bar. Alternatively you can open a *Finder* window and navigate to the *Applications* > *Utilities* folder to find the Terminal application, as shown in *Figure 1*. Double click the icon to open the application.

Figure 1. Open the Terminal.app in the Applications > Utilities folder.

Adjust the font size

That's it! You should see a window open similar to *Figure 2*, althought the default color may be different based on the Appearance settings on your Mac.

Figure 2. A terminal window example on a Mac.

If you are in Dark Mode, you'll likely see a dark window, and in Light Mode you should see a light window. In your Dock at the bottom of your screen, you can press `Control` + *click* on the Terminal icon (or use your *secondary-click* on your mouse) to bring up the icon menu, and choose *Options* > *Keep in Dock* to add it permanently to your Dock.

To finalize your setup, adjust the font size in your terminal so that you can comfortably see the text. You can also change the font family, but be sure to use a *fixed width* font since the terminal expects it for layout purposes. In order to change the font size, select the *Terminal* menu item and choose the *Settings ...* item. In the Terminal.app Settings window, select the *Basic* (Default) Profile, and the *Text* tab in the window panel. Use the *Change ...* button to change the font size, as shown in *Figure 3*.

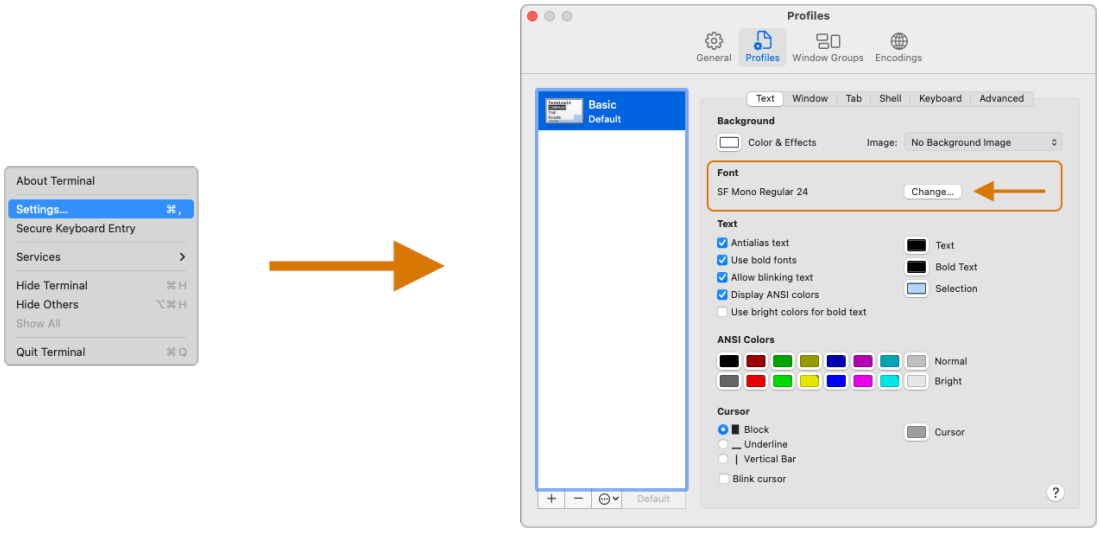

Figure 3. Use the File > Settings... menu item to change the font size as needed in the Terminal.app settings.

You are all set! It's that simple to get configured to use the command line on a Mac. You can continue on to the section entitled *The Shell* to become more acquainted with the command line. Thank you for focusing your power on the magic of the command line!

Setup for Linux

Getting set up on Linux is quite easy as well. On Ubuntu 24.04, the default desktop manager is Gnome. To search for applications, similar to Apple's Spotlight function, press the `Super` key next to the `Alt` key on your keyboard.

If you are on a Windows-branded machine, the `Super` key may have the Windows logo on it. It's also called the `System` key. If you have Linux installed on Mac hardware, this is the `Command` key.

In the search box, type "terminal" (without the quotes), and the default Terminal application icon should be in view. Click on that icon to open the application. You're all set! Once open, you may want to right click on the icon in the *Dash* (i.e. the Application Dock), and choose the *Pin to Dash* menu item so that you have quick access to the Terminal application. See *Figure 4* showing how to search for applications on the Ubuntu Linux Desktop.

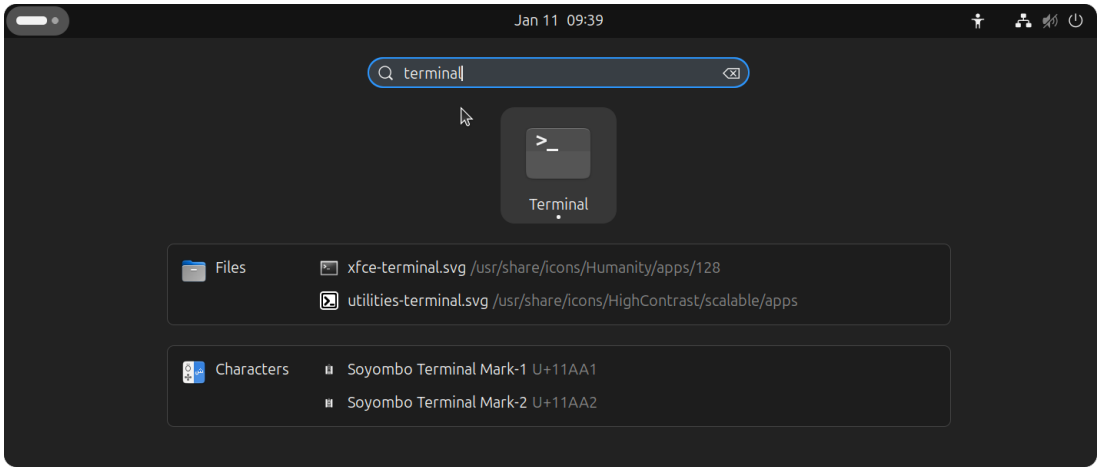

Figure 4. Search for the Terminal application on Ubuntu Linux.

Great! Now that you have the Terminal application running, you should see a window similar to *Figure 5*. Your colors may be different depending on your Appearance settings, but you will either see a Light Mode or Dark Mode window.

Figure 5. A terminal window example on Ubuntu Linux.

Adjust the font size

To finalize your setup, adjust the font size in your terminal so that you can comfortably see the text. You can also change the font family, but be sure to use a *fixed width* font since the terminal expects it for layout purposes. In order to change the font size, select the menu button in the top window bar and choose the *Preferences* item. In the Terminal Preferences window, select the *Unnamed* (Default) Profile, and the *Text* tab in the window panel. Use the *Custom font* checkbox and then the font button to change the font size, as shown in *Figure 6.*

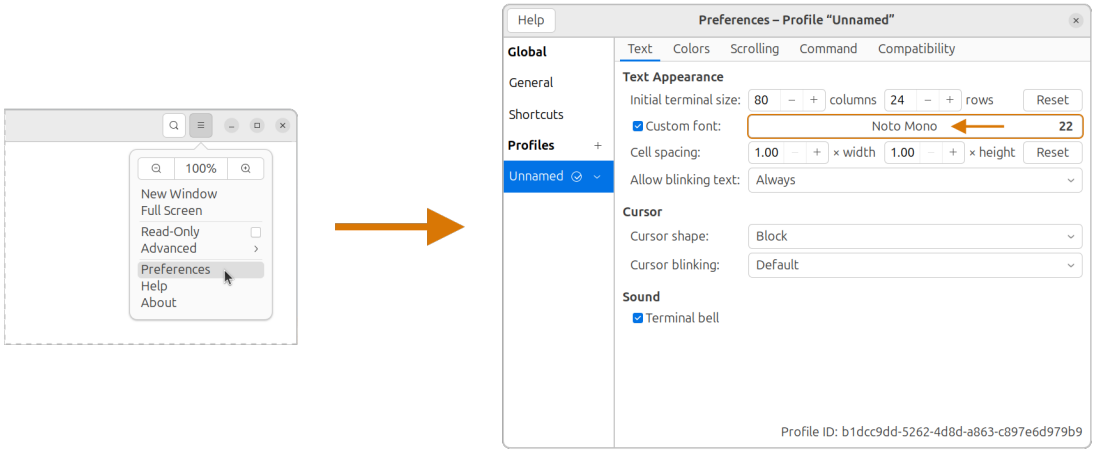

Figure 6. Change the font size as needed in the Terminal preferences.

That's it! It's that simple to get set up to use the command line on Ubuntu Linux. You can continue on to the section entitled *The Shell* onto become more acquainted with the command line. Thank you for taking the next step as a command line magician!

Setup for Windows

The Microsoft Windows operating system has a rich history, but one that is diiferent from the Unix-like operating systems of macOS and Linux. Because of the low-level differences in the systems, Microsoft has created a component called the *Windows Subsytem for Linux*, otherwise known as *WSL*. WSL provides those of us using Windows an integrated system with a full Linux command line environment. In this section, we will complete the following list:

1. Open the Windows Terminal application as an Administrator.
2. Install the Windows Subsystem for Linux component.
    - Install a distribution of Ubuntu Linux.
    - Restart the computer.
3. Enable the Windows Subsystem for Linux required features.
    - Restart the computer.
4. Set up Ubuntu Linux in Windows Terminal
    - Open the Windows Terminal application.
    - Open an Ubuntu Linux tab.
    - Create a Linux user and password.
5. Adjust the terminal font size as needed.

After the Windows Subsystem for Linux installation, the Windows Terminal application will have built-in support and integration for WSL, and will give you a full Linux environment to work with. So let's get started!

Open the Windows Terminal application

Windows Subsystem for Linux is considered a developer tool, and as such, the recommended way to install it is by issuing a command in the terminal application as an Administrator of the computer. To get started, click on the Windows Start menu icon in the Windows Taskbar, or press the `Super` key on your keyboard.

As mentioned before, the `Super` key may have the Windows logo on it, and is usually

next to the `Alt` key.

In the search bar, type "Terminal" (without the quotes). You should see a search result with the Windows Terminal icon. As shown in *Figure 7*, choose the *Run as Administrator* option in the details pane for the Terminal application.

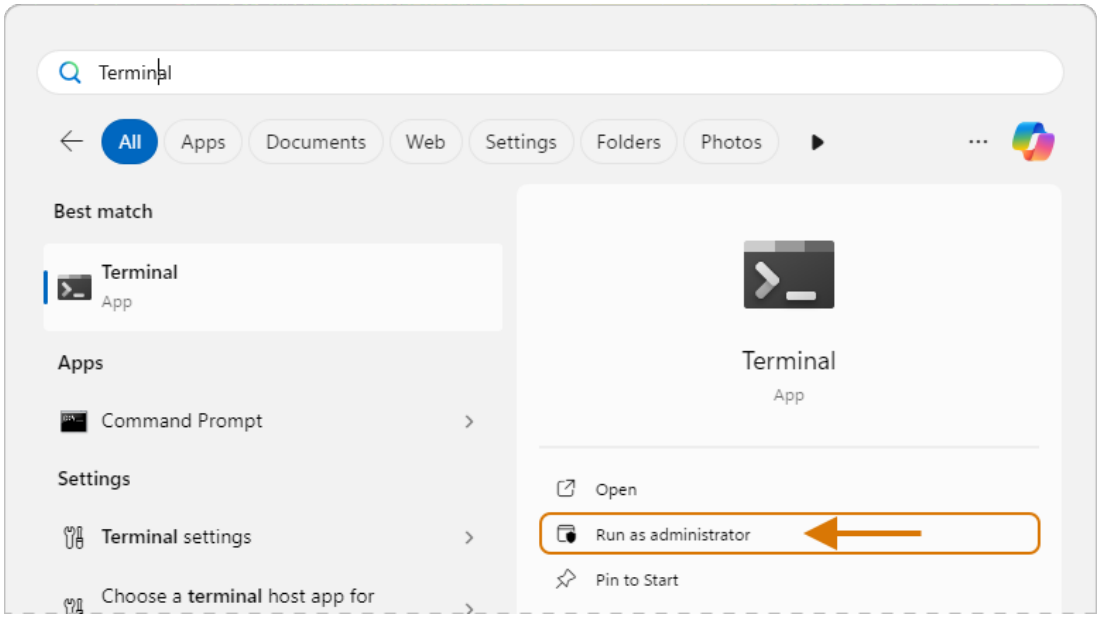

Figure 7. Search for Windows Terminal application and run it as an administrator.

When run as an Administrator, you will see a dialog asking you to make changes to your system, so be sure to choose "Yes" to continue. A terminal window should open and look similar to the window in Figure 8, although the colors may be different depending on your Appearance settings. The Terminal "Powershell" profile usually defaults to a dark background color. To keep this application readily available, *right-click* on the Windows Terminal icon you see in the taskbar, and choose the *Pin to taskbar* menu item.

Install Windows Subsystem for Linux

To install WSL using Windows Terminal, click inside the terminal window and type `wsl --install`, where there is a single space between the `wsl` and the `--install` parts, and press the `Return` key, as shown in *Figure 8*. By running this command, Windows will first download the latest version of the Windows Subsystem for Linux component, and will install the component. It will also install files that are part of the Virtual Machine Platform component that WSL needs for integrating

with the operating system. Once finished, it will prompt you to restart your machine, so do that now.

Figure 8. Run the `wsl --install` *command in the Windows Terminal application.*

Enable the Windows Subsystem for Linux required features

Once rebooted, you will need to ensure that the WSL components are enabled. To do so, click on the Windows Start menu icon in the Windows Taskbar, or press the `Super` key on your keyboard. In the search bar, type "Turn Windows features" (without the quotes). As shown in *Figure 9*, you should see a search result with a Control Panel option for "Turn Windows features on or off". Click on this option to open the features dialog, and scroll down in the dialog toward the bottom.

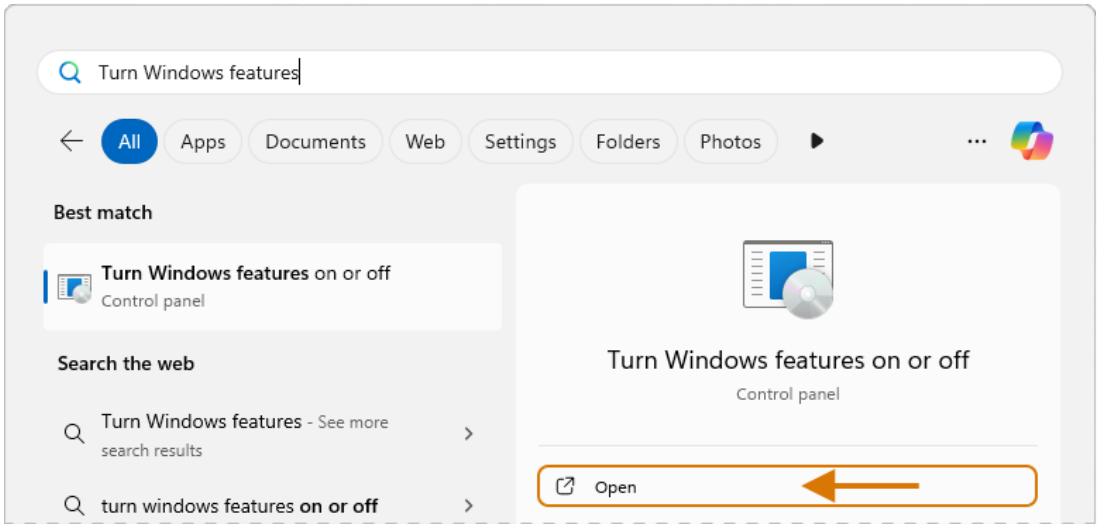

Figure 9. Use Windows Search to open the 'Turn Windows Features on or off' Control Panel.

As shown in *Figure 10*, ensure that the "Virtual Machine Platform" and the "Windows Subsystem for Linux" items are checked. After closing this dialog box, Windows will enable these components, and will prompt you to restart your machine.

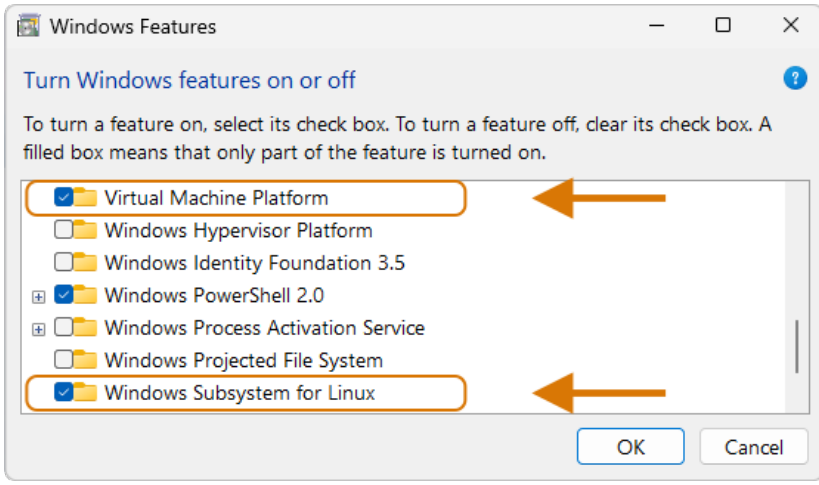

Figure 10. Enable the Virtual Machine Platform and Windows Subsystem for Linux components in the Control Panel.

Set up Ubuntu Linux in Windows Terminal

Great, the underlying components are now installed! It's now time to set up Ubuntu Linux using the Windows Terminal application. So, open the Windows Terminal application again, either from your taskbar or the Windows Start menu. By default, it will open with a Windows PowerShell profile tab. As shown in *Figure 11*, click on the down-arrow icon next to the '+' icon at the top of the window to open a new tab, and select the Ubuntu profile item.

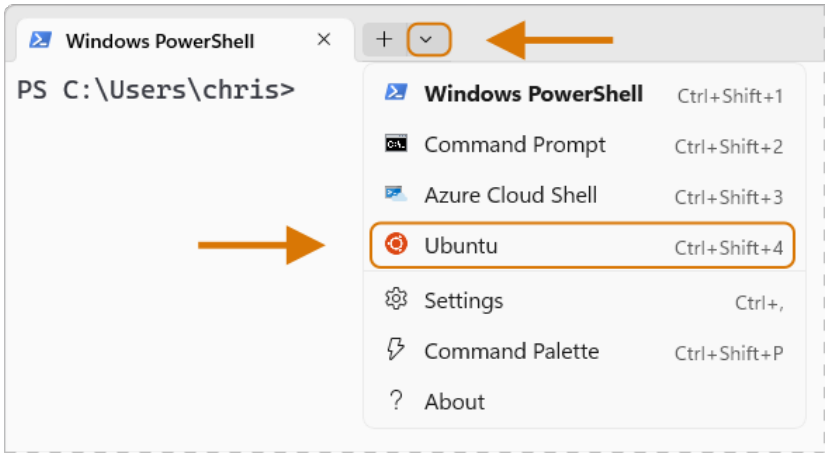

Figure 11. Open an Ubuntu Linux profile using the drop-down icon in the Windows Terminal tab bar (next to the + sign.)

This will initiate the Windows Subsystem for Linux, and will start Ubuntu Linux. It will take a few minutes to initialize, but will then prompt you to create a UNIX username (i.e. Linux username). You can use the same name as your Windows user name, or a different one. After entering your name, and pressing the `Return` key, it will then prompt you for a password. Type in a password of your choosing, and also write it down.

As you type in the password field, your typing will not be visible, which is typical behavior for command line password entry.

Confirm your password a second time when prompted, and your Linux environment will be set up for you! Once the text has stopped scrolling in the window, you will have a fully-functional Linux command line, similar to what is shown in *Figure 12*.

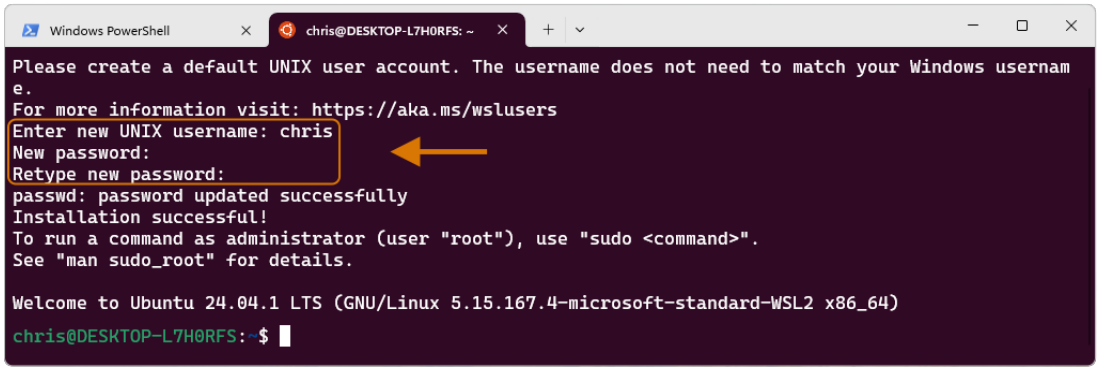

Figure 12. A complete Linux command line running within Windows.

Adjust the font size

To finalize your setup, adjust the font size in your terminal so that you can comfortably see the text. You can also change the font family, but be sure to use a fixed width font since the terminal expects it for layout purposes. In order to change the font size, click on the drop-down icon in the tab bar again, and choose the *Settings* item in the menu. This opens a new tab in the Windows Terminal with the settings for the application, and the settings for each profile, including the Ubuntu profile. In the sidebar on the left, scroll down and click on the Ubuntu profile, as shown in *Figure 13*. The Ubuntu profile settings will appear in the right window pane. Scroll down in this pane, and choose the *Appearance* section.

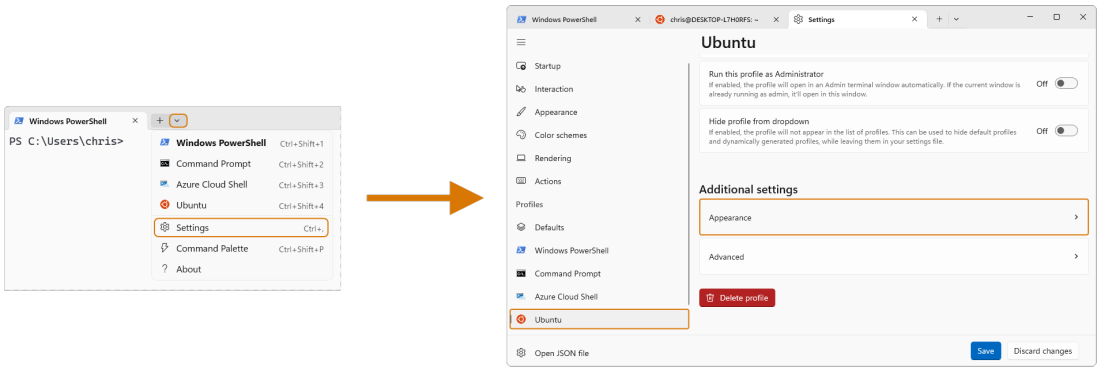

Figure 13. To change the font size, first open the Terminal Settings and choose the Ubuntu profile's Appearance section.

This opens a dialog that allows you to change the font size as needed. See the example in *Figure 14* for changing the font size. Once finished, close the Appearance dialog and click the *Save* button at the bottom of the Settings tab, as shown in *Figure 14*, and then close the Settings tab.

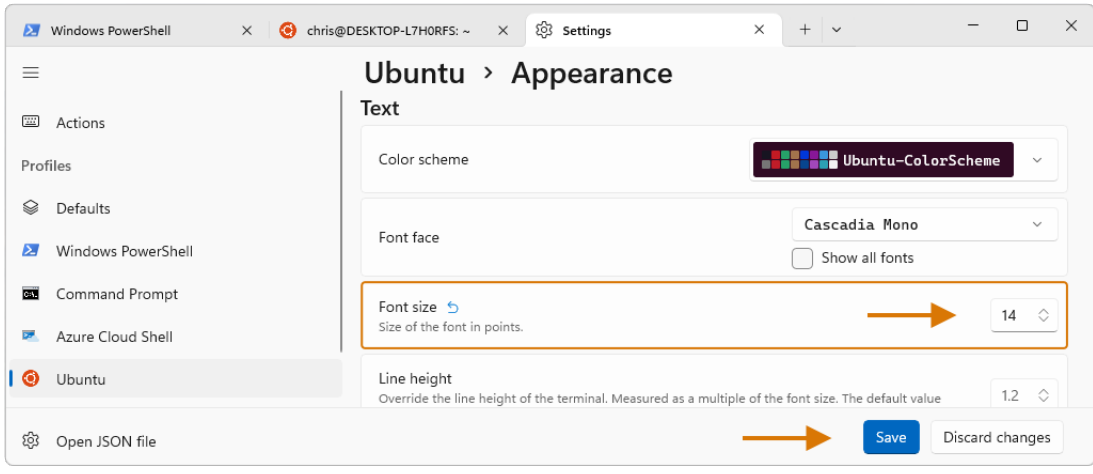

Figure 14. Adjust the font size as needed, and click on the Save button to save the profile changes.

Congratulations! You are ready to continue with your command line journey in the next section to learn about the concept of *The Shell*! Thank you for building your magic command line skills!

The Shell

Now that you have set up a working terminal application, you are well on your way to using the command line with ease! To help with some of the terminology, let's first discuss what a *shell* is. In the course of your work, someone may say "Open up a terminal", "Open up a console", or "Open up a shell". As we mentioned before, these terms are often used interchangeably. However, let's touch on the idea of a shell in more detail.

When you open your terminal application, a number of things happen in the background to set up your environment, such as loading your default settings profile. As part of this process, the terminal will start another process called a *shell interpreter*—which is a program running invisibly in the background—that is waiting for your command to be typed. When you do type the command and hit the `Return` key, the shell program kicks into gear, interprets all of the text that you entered, and runs the command like a programming language. In fact, you are actually writing commands in what is called a *shell language*!

Here's the same example as *Example 1*, but with a comment added to the command:

Example 1: Issuing the `pwd` *command with a comment*

```
chris@ophir ~ % pwd # Issue the pwd command
/Users/chris
```

Notice that the `pwd` characters are followed by a space, then a `#` (hashtag) symbol, and then another space and the comment sentence. The shell interpreter evaluates everything in the command, and validates it based on the shell language rules. In this case, we just learned that you can issue a command, followed by a `#` (hashtag) character and any other written comment, and the shell will ignore any characters to the right of the hashtag because it knows it is a comment, and will proceed to give you back a result.

The take home message here is that the shell interpreter is doing the heavy lifting behind the scenes, and there are many variants of these interpreters. The earliest shell interpreter is attributed to Louis Pouzin in 1964 for the CTSS/Multics operating system.[4] Since 1979 the UNIX operating system included the default shell interpreter called `sh`, and a free version of it is still the default on Linux and Mac. That said, there has been immense improvements to shell programming languages since the 1970s, and many different interpreters, with new features, have been written and shipped with various operating systems. To name a few, there is `ksh`, `csh`, `bash`, and `zsh`.[5] On modern versions of Linux, the default shell tends to be `bash`, and on a Mac it is now `zsh`. For the purposes of this handbook,

we'll see that these shells all work similarly if not identically in some cases. In the next section, we'll take a closer look at the *command prompt*, but know that the shell interpreter is the workhorse behind your magic commands!

The Command Prompt

We are now familiar with opening a terminal application, which in turn spins up a shell interpreter to handle your commands behind the scenes. Now let's familiarize ourselves with the idea of the *command prompt*, which is your go-to location for typing in commands. Once your terminal application has opened, you are presented with an almost empty window, with a few characters written at the top. These characters are followed by the *cursor*, which is some sort of flashing—or not flashing—block character, underscore or other inviting symbol that ever so subtly evokes "type here". Collectively, all of these characters are considered the command prompt—dutifully waiting for you to enter a command. See *Figure 15* for a labeled diagram of a typical command line.

Figure 15. A typical command line, with an example of a default `zsh` command prompt, showing the user name, the computer host name, the current folder (`~`), and the `%` sign, followed by a block cursor.

The command prompt on modern systems tend to include your user name, followed by an `@` (at) symbol, followed by the network host name of your computer. There is usually some kind of delimiter character (a space or colon), followed by a `~` (tilda) character (which, as we discuss later, represents your home folder). Lastly, you will see either a `$` (dollar sign) character (for `bash` shells) or a `%` (percent) character (for `zsh` shells). Command prompts can be customized to your liking—modern prompts can be very colorful and include a lot of information, or can be bare bones, depending on your preferences. Take a look at *Example 3* for various command line prompt examples.

Example 2: Examples of various command line prompts.

```
chris@ophir ~ % ❶
chris@nuthatch:~$ ❷
root@nuthatch:~# ❸
>_ ❹
# ❺
```

❶ A `zsh` prompt with username, hostname, current folder, a `%` symbol, with a block cursor

❷ A `bash` prompt with username, hostname, current folder, a `$` symbol, with a block cursor

❸ An administrator prompt with username, hostname, current folder, a `#` symbol, with a line cursor

❹ A minimalist prompt with a `>` (chevron) symbol and an `_` (underscore) cursor

❺ A typical root prompt (administrator) with a `#` symbol

What character shows up in the prompt is configurable, and some people prefer having a minimalist prompt with just a `>` (chevron) symbol, with no username or other information. The command prompt tends to be on the very first line of your terminal window. The combination of the command prompt, and this imaginary first line of text at the top of your window, is considered the *command line*. This is your magic portal that gives you superpowers with your computer, which we will see in the following chapters.

On Unix-like operating systems like macOS and Linux, an account for the administrator (also called the super-user, or root), conventionally is denoted by a `#` (hashtag) symbol in the command prompt rather than a `$` (dollar) or `%` (percent) sign, which denote a regular user. This reminds you to be cautious when issuing commands as the administrator.

The Parts of a Command

In the previous sections we've had a brief look at a very simple command called `pwd`, and we will discuss it further in *Chapter 2. Core Commands*. But to learn about the parts of a command, and to get a feel for command line syntax, let's look at an imaginary command called `stardb`, which is shown in *Figure 16*. The command stands for "star database", and so you could imagine that we have a database of star information stored within it, and the `stardb` command allows us to work with the database. In fact, one way to work with it is to *search* the database and filter the results based on what stars you are interested in. The command even has some built-in options to return very popular results, like only

returning giant-sized stars. The command can also save your search records to a file of your choosing, so you can share your starry sky knowledge with friends. So, given our fictitious `stardb` command, let's discuss the parts of a typical command that are shown in *Figure 16*.

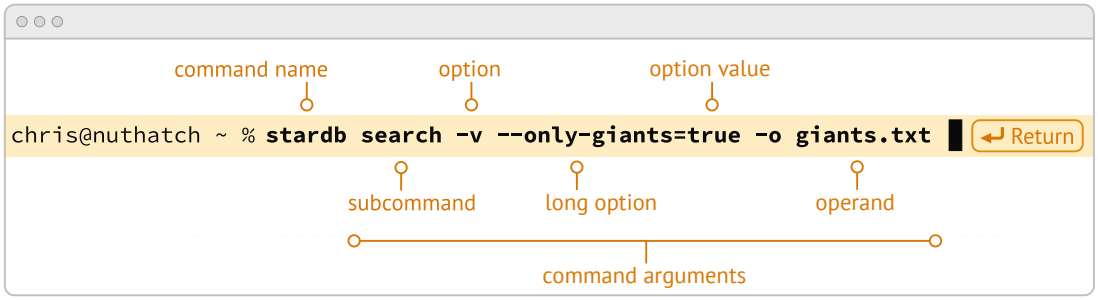

Figure 16. The labeled parts of a command and the command's arguments, including a subcommand, option, long option, option value, and operand.

Let's start with the *command name* itself, `stardb`. To be able to run this command, it has to be installed on your system, and located in a folder that is well-known to your shell interpreter.[6] Let's assume that our `stardb` command is installed correctly. Next, notice that there is a *space* character after the command itself, and in between the other parts of the full command. This is very important, because the space character acts as a boundary between the command parts, and the shell interpreter will parse the command parts based on these spaces. If you have two or more consecutive spaces between command parts, the shell interpreter treats them as a single space combined, so don't worry about having extra spaces. But yes, be sure to use a space between the parts of a command.

When working with file names that have spaces in the name, use either double-quote or single-quote characters around the file name to tell the shell to treat the spaces as part of the file name. For instance, use "the biggest stars.txt" or 'stars are awesome.jpg' if there are spaces in the file name.

After you've typed the command name, you then type the space-separated list of *command arguments*. Command arguments are a way to adjust the behavior of your command, and in the case of our imaginary `stardb` command, we pass in a *subcommand* called `search`, to tell the `stardb` command that we'd like to query the database. The twenty-one fundamental commands we cover in the book don't make use of subcommands, but it's good to know that other commands do use them.

It's important to note that all of the all of the characters we type on the command line are case-sensitive, so `stardb search` is all lowercase. Most commands tend to be

> lowercase, but it's not a steadfast rule. Commands can be created with both uppercase and lowercase, and numbers in them as well.

So after our first command argument called `search`, notice the `-v` argument, which is next in line after the required space character. This is known as a *command option*, which can also be called a *flag*, or a *switch*. Command options are like the knobs or dials on a coffee maker that let you adjust the settings and refine how the coffee turns out. The `-` (dash) character before the `v` is what tells the shell interpreter that this is a command option, and it will treat it as such. In our `stardb` scenario, the `-v` option means that we want it to return *verbose* output, meaning that we want all the star details we can get from the database. It's very common for commands to have a `-v` option that is a request for verbose output. However, note that the `-v` option is command-specific, so it could mean something entirely different when used with a different command. The way to know what options are available for a command is to read the *manual page* that explains how the command works and what it expects. We will cover this topic in *Chapter 2. Core Commands*.

Now we know that you can pass single-letter options to a command, and that the meaning of the option might not be entirely apparent. So a second and more expressive way to modify a command is with *long options*, such as the `--only-giants=true` command argument in our imaginary scenario. Long options spell out how they modify the command and can be easier to read, but are longer to type. In this case, the long option is `--only-giants`, and the `--` (double dash) is the indicator to the shell that this is an option. The `=true` portion is setting an *option value*, meaning that the command has a setting of `only-giants` (for the search), and the value will be set to `true`. So long options are helpful for readability, short options are quick and easy once you are familiar with the command. Both can potentially take option values, but are not required. For instance, the command may set a verbosity level with `-v 8` where the option value could be a number from 1 to 10. Commands often offer both a short option and a long option at the same time. For instance, `-h` and `--help` will often be available and will both print out a short synopsis of how to use the command and what all the options are.

> While we are focused on the short and long option styles, note that you may also see options like `-help` which has the single `-` (dash) of a short option and a full word like a long option. This format is valid as well. Just read the manual for each command to know what is expected.

We now come to the last argument of our `stardb` command, which is `-o giants.txt`. This is a short option that means "write the *output* to the given file name", and so our `stardb` command will create a file called `giants.txt` that contains the results of our search, with plenty of star-friendly information. The file name that we pass in is a type of argument called an *operand*, meaning that it is

being acted upon in some way by the command, which is the *operator*. Arguments that refer to the output, or results-side of the command are usually considered operands. This is a fine detail, but just know that the terms *arguments* and *operands* are at times used interchangeably.

We have made it to the very end our command, where we see the `Return` key symbol. Commands are run when you press the `Return` key, so be sure to do so when you've finished writing your command. When you do, in this case, star information will be written to the `giants.txt` file. You would also normally see additional information printed to your terminal screen in the lines below your command. So that's it! These are the general parts of a command we use on the command line, but what if our command is super long? Will it wrap to the next line? Will it still be readable? Let's discuss those topics.

Single Line and Multi-Lined Commands

A lot of commands can be short and sweet, like the `pwd` command we've seen in the previous sections. But many commands have a lot of options available to modify the command and refine the results that are returned. Some commands include dozens of options, and it may be helpful to use many of them at once. So our command will often not fit on a single line of text available in your terminal window, unless you have a very large screen and can widen the terminal window. So, we'll often see commands wrap to the second and third line of the window, as depicted in *Example 4*.

Example 3: A long command example with many options that wraps to the second line.

```
chris@ophir ~ %  stardb search -v --only-giants=true --star-classes "bright-
yellow-giant, orange-bright-giant, red-super-giant, white-super-giant, blue-
super-giant" -o super-giants.txt
```

Having the command wrap to the next line can work just fine, and will only be executed when you press the `Return` key. But there are times when the command gets very long and complex, and you just want to clean it up. We have the power! You can use a `\` (backslash) character followed by the `Return` key which is used as an *escape character*. The shell interpreter will ignore the `Return` keypress. You can use the `\` (backslash) character as many times as needed to make your single-line command a tidy *multi-line command*, as is shown in *Example 5*.

Example 4: A multi-line command example with options split across lines with a `\` (backslash) character.

```
chris@nuthatch:~$ stardb search -v \
> --only-giants=true \
> --star-classes "red-super-giant, white-super-giant, blue-super-giant" \
> -o super-giants.txt
```

As you type this command, notice that the shell places the `>` (greater-than) character on the following line, indicating that it is waiting for the rest of the command to be typed. Now, when you press the final `Return` key without a `\` (backslash) character, your command will execute. Show all the super giant stars!

When you are working with commands, you will notice that your mouse pointer has no effect on the position of your command line cursor, which takes a little getting used to! For very long commands, either as single line or multi-line commands, there are times when you need to go back and edit a portion of the command that may have been mistyped, or you may want to change an option. You can use your keyboard's `◂` left arrow and `▸` right arrow keys to move the cursor to the left and right, and the `Delete` key will delete characters at the cursor. Take some time to familiarize yourself with moving left and right along your command.

Of course, this can become tedious when you have a very long command and need to edit an option that is close to the beginning of the command, and your cursor is near the end. But wait, there's a handy trick! You can use the `Control`+`a` key combination to skip the cursor to the beginning of your command! To be specific—while holding the `Control` key, also type the `a` key, and zoom—your cursor has raced to the beginning of the command! Likewise, you can use the `Control`+`e` key combination to skip the cursor back to the end of your command. These two keyboard sequences can really speed up your command editing, when your commands get noticeably long.

Some shells also support the `Option`+`◂` left arrow key combination (or `Alt`+`◂`) to move the cursor word-by-word to the left, and the `Option`+`▸` right arrow key combination (or `Alt`+`▸`) to move the cursor word-by-word to the right.

As we type and execute commands with the return key, we inevitably issue a command that wasn't quite what we meant, but it was close! Perhaps there was a single typo in the middle of the command. Instead of re-typing the very long command again, you can use the `▴` up arrow key to scroll up to your previous command, and then edit it. Yes! It's so easy! In fact you can use the `▴` up arrow key multiple times to scroll through your command history, and can use the `▾` down arrow key multiple

times to scroll back to your more recent commands. Amazing!

All of these key combinations can be a game changer with command line productivity, so practice using them often, and they will become second nature. With dedication and repetition, using the command line will become extremely familiar, and you'll notice how rapidly you can get things accomplished without leaving your keyboard. We're just getting started!

Command Line Interfaces are Awesome!

Our computers are wonderful tools for creativity, particularly due to the graphical user interface metaphor that helps us navigate our machines. And now, as we familiarize ourselves with the command line interface, we see that the terminal application can become our trusty friend and a powerful addition to our toolbox. The command line helps us uncover seemingly secret functionality on our computers by using text-based commands to orchestrate our work in a concise and effective manner. In this chapter we have learned how to access a terminal application on Mac, Linux, and Windows. We now have a solid understanding of a shell interpreter that handles the commands we type, what a command prompt is, and how to construct a command with command arguments and the various styles of command options. We now know how to edit single and multi-line commands, and how to move our cursor within our commands with ease. These concepts set the foundation for the upcoming chapters where we learn individual commands that enable us to navigate our computers, create and manage files and folders, and work with our data files in ways that are often impossible with a graphical approach. The command line is truly a tool of empowerment, and a magic portal into your machine. In *Chapter 2. Core Commands*, we will get hands on experience with navigation commands, and will begin to traverse our files and folders with ease, while getting to know the structure of our storage file systems in better detail. Let's go!

[1] Before desktop computing arose, sending commands to a computer was the predominant way to work with them. The success of the UNIX operating system developed by AT&T Bell Laboratories inspired the development of Linux, the architecture of macOS, and later Windows Subsytem for Linux. We focus on commands in these Unix-like systems.

[2] Ceruzzi, Paul E.. A History of Modern Computing. United Kingdom: February, 2003. https://mitpress.mit.edu/9780262532037/a-history-of-modern-computing/

[3] Ubuntu and Canonical are registered trademarks of Canonical Ltd.

[4] See https://multicians.org/shell.html

[5] The Bourne shell (sh) was wriiten by Stephen Bourne at Bell Labs for UNIX and was released in 1979. Also at Bell Labs, David Korn created Korn Shell (ksh) which was released in 1983 for UNIX. An alternative for sh called CShell (csh) was written by Bill Joy at the University of California Berkeley for BSD UNIX, and Brian Fox wrote the Bourne Again Shell (bash), which is an open source rewrite of the Bourne Shell. In 1990, Paul Falstad released zsh as an open source program.

[6] Shell interpreters have a concept of a PATH variable, which is a list of folders that it will consult in order to find the command you want to run.