00000001

CHAPTER 1

Part 1 Preliminaries: Baseline Development Environment

Learning Objectives

- *State the importance of a properly-configured development environment*
- *Configure folder settings in to show hidden files and file suffixes*
- *Install Homebrew package manager on macOS*
- *Identify package managers for Linux and Windows*
- *Install and configure Python*
- *Set environment variables in macOS, Linux, and Windows*
- *Install and configure Visual Studio Code*
- *Select and install Python development extensions in Visual Studio Code*
- *Perform basic text editing with VIM*
- *Edit text files with Nano*
- *Install ITerm2 on macOS*
- *Configure important .bash_profile settings*
- *Configure Bash aliases*

Introduction

In this chapter I show you how to set up and configure your *baseline development environment*. I use the term development environment to refer to the set of hardware and software tools required to develop applications outside of the programming language itself. It's a baseline setup in that it provides a foundation upon which additional tools can be added and configured to meet the needs required in subsequent parts of the book.

A lot of what you'll learn in this chapter can best be described as *incidental system administration*. Every competent software developer needs to know enough system administration to get stuff done. You'll configure operating system settings and environment variables, install and configure software, and ensure your development environment is ready in all respects to support software development.

In this chapter and throughout the book I will focus on three operating systems: *macOS (A UNIX® certified OS)*, *Linux*, and *Windows*. I do this because students in my course come to class with laptops running these three operating systems. I generally spend two whole class sessions working with them to ensure their computers are configured correctly to support software development.

The primary goal of this effort is to ensure that when you download code examples from the book's GitHub repository they run as expected regardless of operating system you happen to be using. The secondary but no less important goal is to establish a familiar development experience across all three operating system platforms, which is something you may find helpful.

As you progress through this chapter you need only focus on the sections related to your operating system(s). However, to facilitate the secondary goal mentioned above, I will use the Bash shell on all three operating systems. This may force you out of your comfort zone especially if you are relatively new to programming and have never used a terminal. I assume readers have no familiarity with terminal usage of any kind but it's a critical skill to cultivate as a software developer.

Let's begin with some initial housekeeping...

1 Initial Housekeeping

This section guides you through some basic system configuration tasks that makes your computer developer friendly. Note that I will use the terms *folder* and *directory* interchangeably throughout the text. Let's start with the most important thing you can do to eliminate potential headaches regardless of what type of operating system you're using.

1.1 No Spaces In Home Directory Name

There should be no spaces in your home directory name. This holds true for all operating systems but especially for Microsoft Windows. The reason is that some software installers and development tools deal poorly with spaces in a user's home directory name and this causes a lot of grief. So, to avoid the headaches, ensure your home directory has no spaces. **You have been warned**. (*I would extend this advice to no spaces in any directory or filename, period.*)

1.2 Windows

If you use Microsoft Windows the following settings and suggestions will prevent a ton of headaches.

1.2.1 Show Hidden Files and Folders

You need to see hidden files and folders. This applies to folders viewed via a graphical user interface, which is how most novice programmers interact with an operating system until they learn to embrace the power of a terminal. You can open any folder to do this. On the taskbar you should see a folder icon as shown in figure 1-1.

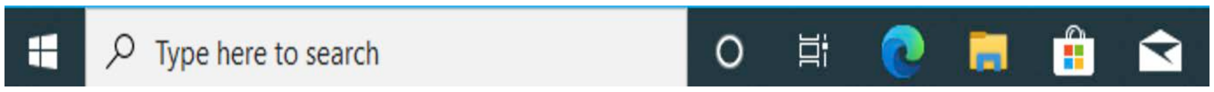


Figure 1-1: Windows Taskbar with Folder Icon

Referring to figure 1-1 — Click on the folder icon to open File Explorer as shown in figure 1-2.

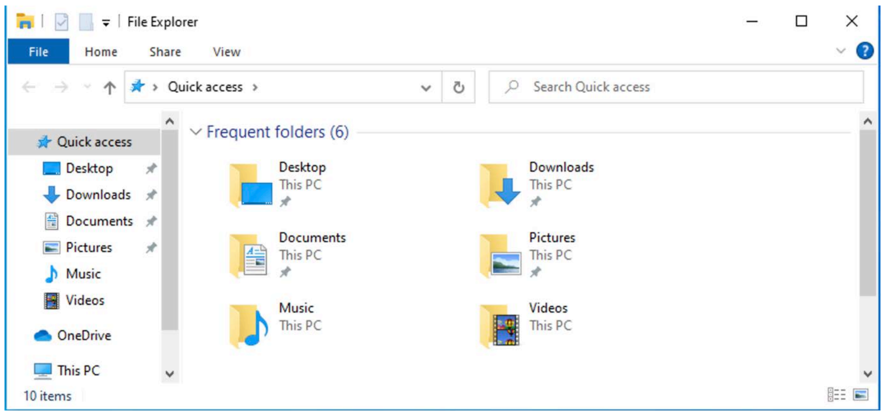
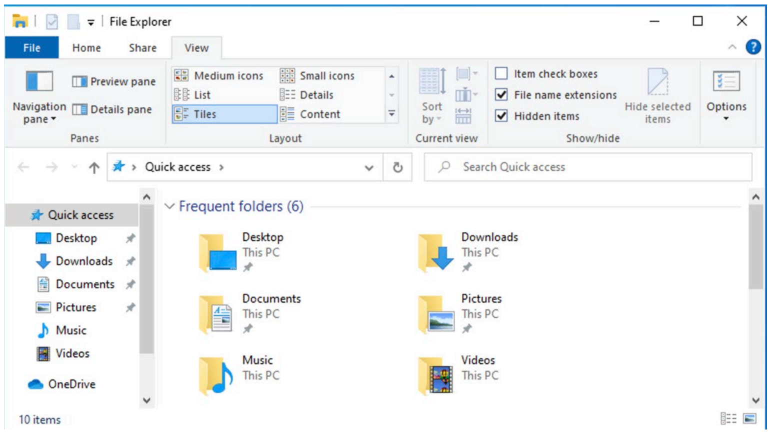


Figure 1-2: Windows File Explorer

Referring to figure 1-2 — By default, File Explorer opens the Quick access view, which lists frequently accessed items. Click on the **View** tab to reveal the view options then check the *File name extensions* and *Hidden items* check boxes as shown in figure 1-3.

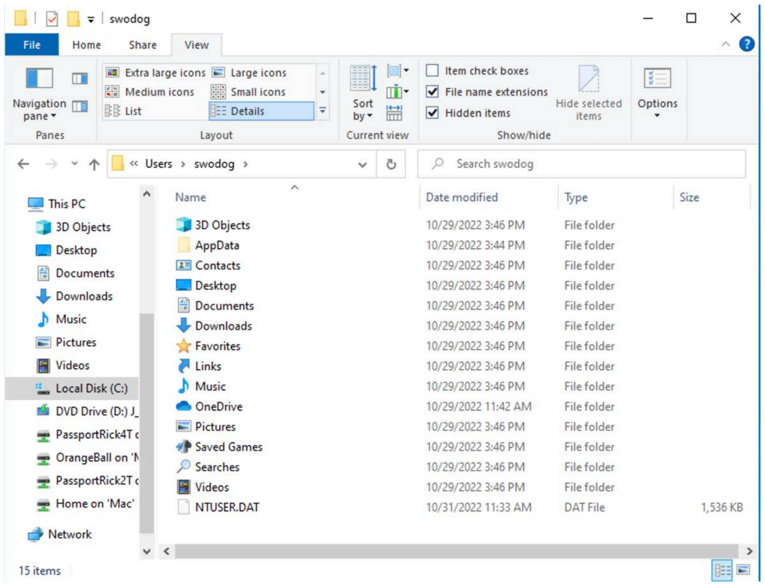
Referring to figure 1-3 — While you're at it, change to the Details Layout then navigate to your home folder, which can be found in This PC > Local Disk(C:) > Users > *username*, where *username* is the account you use to log into your computer. See figure 1-4.

Referring to figure 1-4 — What you see in the left-hand column and the items listed under This PC may be different from what you see here. I have external hard drives attached to my machine. In the right-hand column you'll see the items in your user account's home folder. Locate the AppData folder, which will appear lightly grayed out. The AppData folder is a hidden folder and is where installers will often install software if you don't provide an explicit installation location. Being able to see hidden files and folders will save you hours of searching and banging your head against your desk. OK, let's now add some handy shortcuts to the desktop and taskbar.



- Layout: Tiles
- [ ] Item check boxes
- [x] File name extensions
- [x] Hidden items

Figure 1-3: View Options Set



- Layout: Details
- [ ] Item check boxes
- [x] File name extensions
- [x] Hidden items

Name	Date modified	Type	Size
3D Objects	10/29/2022 3:46 PM	File folder	
AppData	10/29/2022 3:44 PM	File folder	
Contacts	10/29/2022 3:46 PM	File folder	
Desktop	10/29/2022 3:46 PM	File folder	
Documents	10/29/2022 3:46 PM	File folder	
Downloads	10/29/2022 3:46 PM	File folder	
Favorites	10/29/2022 3:46 PM	File folder	
Links	10/29/2022 3:46 PM	File folder	
Music	10/29/2022 3:46 PM	File folder	
OneDrive	10/29/2022 11:42 AM	File folder	
Pictures	10/29/2022 3:46 PM	File folder	
Saved Games	10/29/2022 3:46 PM	File folder	
Searches	10/29/2022 3:46 PM	File folder	
Videos	10/29/2022 3:46 PM	File folder	
NTUSER.DAT	10/31/2022 11:33 AM	DAT File	1,536 KB

Figure 1-4: File Explorer Details Layout in User's Home Directory

1.2.2 Handy Desktop and Taskbar Shortcuts

It's super helpful to add shortcuts to frequently used applications on the desktop and taskbar. Let's start with the Command Prompt. In the taskbar search box type `cmd`. This will locate the Command Prompt application and display a pop-up window with some choices as shown in figure 1-5.

Referring to figure 1-5 — In the right-hand column click **Pin to Start** and **Pin to taskbar**. You should see the Command Prompt icon appear in the taskbar. Next, click **Open file location**. This will open a window with a list of Windows System Start Menu items as shown in figure 1-6.

Referring to figure 1-6 — Right-click on the Command Prompt shortcut and from the pop-up menu select **Send to > Desktop (create shortcut)** as shown in figure 1-7.

Referring to figure 1-7 — This will send a new Command Prompt shortcut to the desktop.

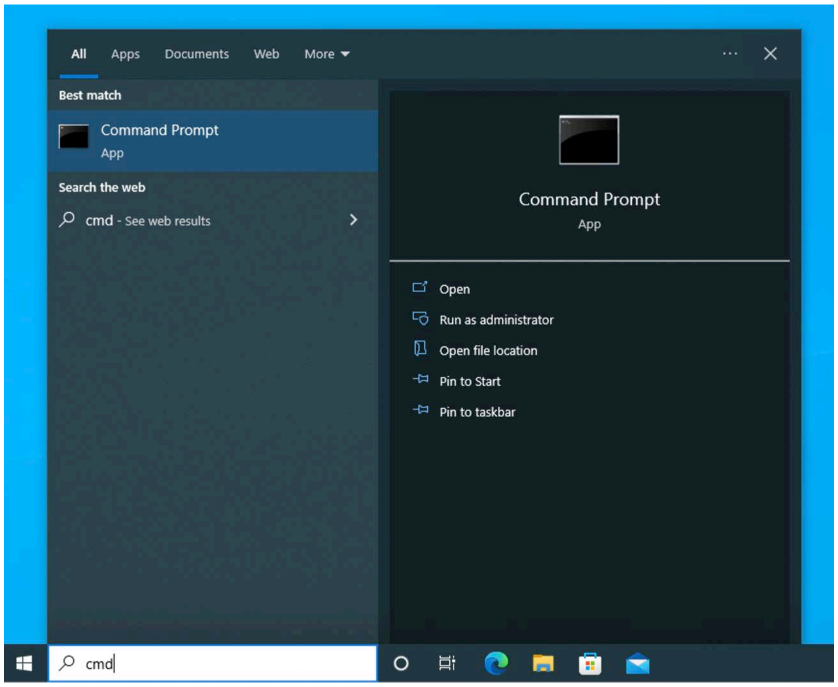
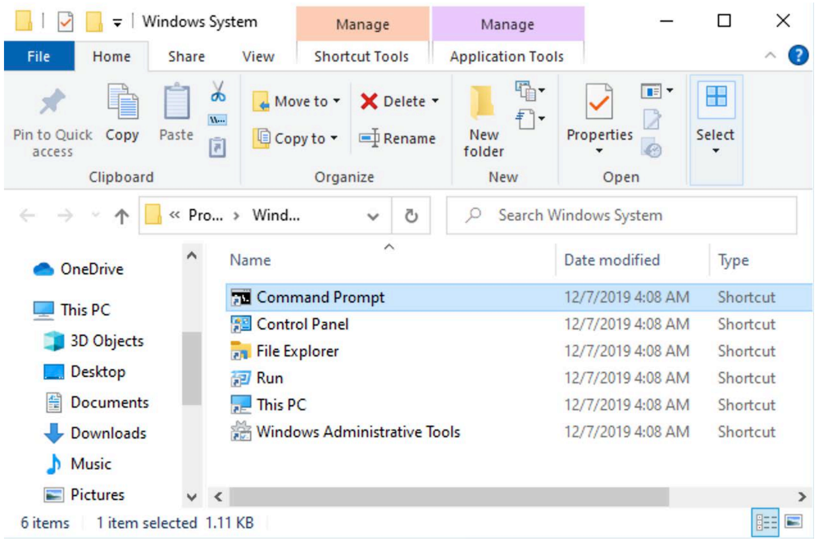


Figure 1-5: Searching for cmd



Name	Date modified	Type
Command Prompt	12/7/2019 4:08 AM	Shortcut
Control Panel	12/7/2019 4:08 AM	Shortcut
File Explorer	12/7/2019 4:08 AM	Shortcut
Run	12/7/2019 4:08 AM	Shortcut
This PC	12/7/2019 4:08 AM	Shortcut
Windows Administrative Tools	12/7/2019 4:08 AM	Shortcut

Figure 1-6: List of Shortcuts

1.2.3 Additional Application Shortcuts — For Now

OK, repeat what you just did with the Command Prompt for the following applications:

- PowerShell
- Services
- Control Panel
- Computer Management

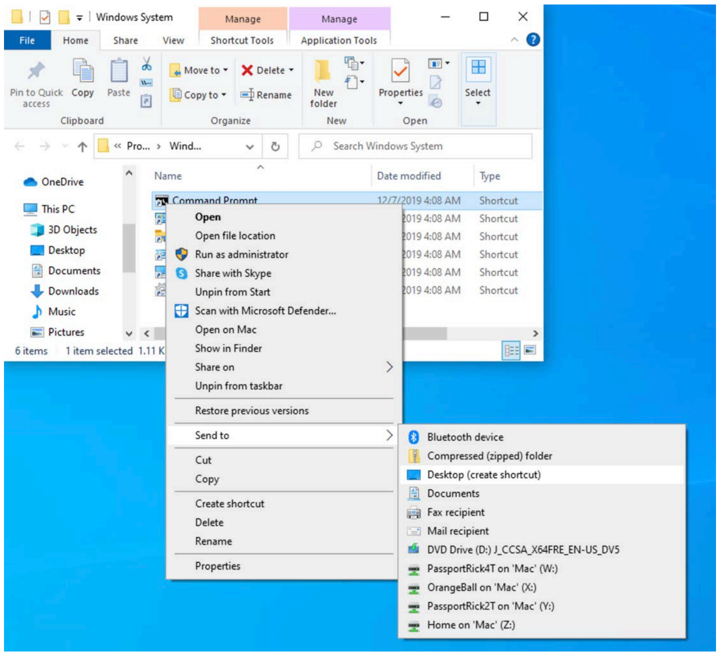


Figure 1-7: Send Command Prompt Shortcut to Desktop

As you add more software development applications, you'll want to add shortcuts to these as well. I'll let you know when I think it's a good idea to add a shortcut for something to your task-bar and desktop.

1.2.4 Drive and Folder Shortcuts

A few other helpful shortcuts to have on your desktop are ones that provide quick access to your hard drive and important folders. To add a shortcut to the C drive click on the taskbar folder icon to open File Explorer, locate Local Disk (C:) in the left-hand column, and click and drag it to the desktop as shown in figure 1-8.

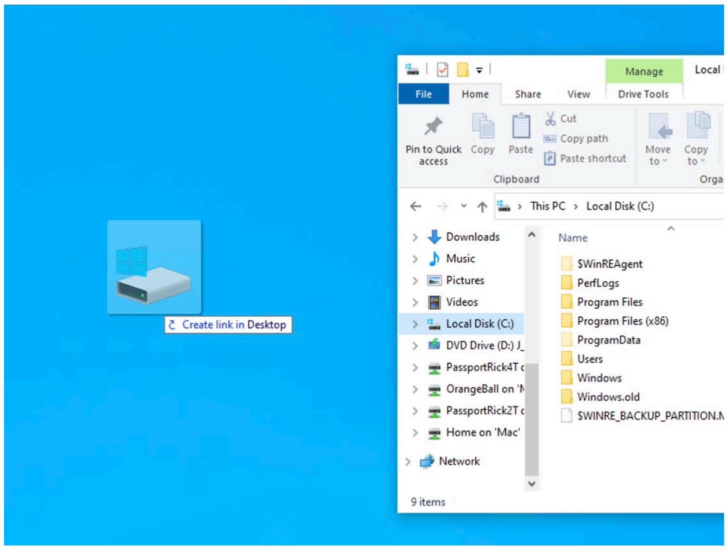
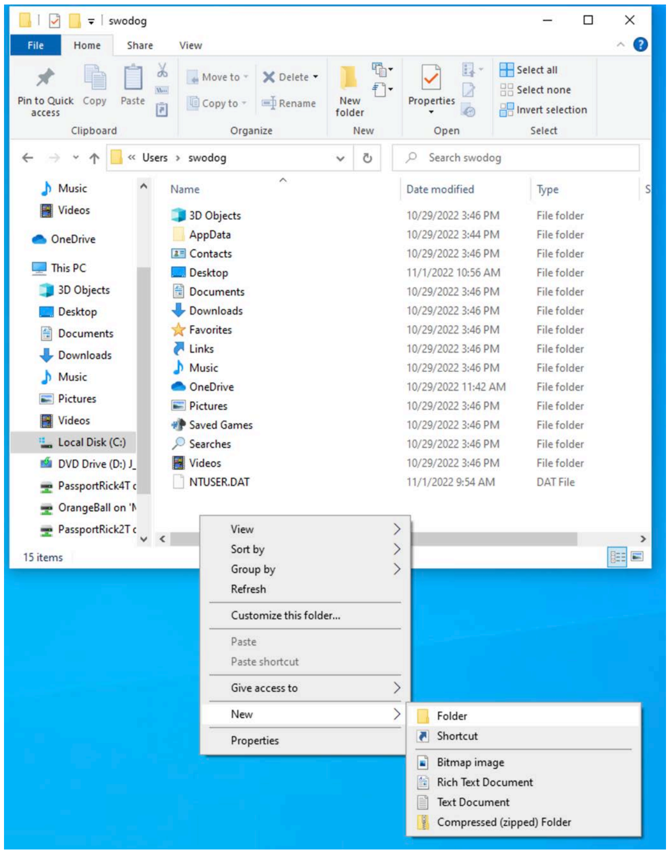


Figure 1-8: Drag and Drop C Drive to Create Shortcut on Desktop

Referring to figure 1-8 — Release the mouse button to create the link on the desktop. Repeat this for any drives and folders you want quick access to, such as a development projects folder, which I discuss in the next section.

1.2.5 Create Development Projects Folder

You'll find it convenient to have all your development projects centrally located. I like to store my projects in a folder called **dev** located in my home directory. To create your dev folder, first navigate to your home directory, which is This PC -> Local Disk(C:) > Users > *username*, where *username* is the account you used to log in to your computer. Now, there are a few ways to skin the cat here. Click on the **Home** tab and click the **New folder** icon, or right-click in the right-hand column and select **New > Folder** as shown in figure 1-9.



```
Name            Date modified        Type
3D Objects      10/29/2022 3:46 PM   File folder
AppData         10/29/2022 3:44 PM   File folder
Contacts        10/29/2022 3:46 PM   File folder
Desktop         11/1/2022 10:56 AM   File folder
Documents       10/29/2022 3:46 PM   File folder
Downloads       10/29/2022 3:46 PM   File folder
Favorites       10/29/2022 3:46 PM   File folder
Links           10/29/2022 3:46 PM   File folder
Music           10/29/2022 3:46 PM   File folder
OneDrive        10/29/2022 11:42 AM  File folder
Pictures        10/29/2022 3:46 PM   File folder
Saved Games     10/29/2022 3:46 PM   File folder
Searches        10/29/2022 3:46 PM   File folder
Videos          10/29/2022 3:46 PM   File folder
NTUSER.DAT      11/1/2022 9:54 AM    DAT File
```

Figure 1-9: Creating a New Folder

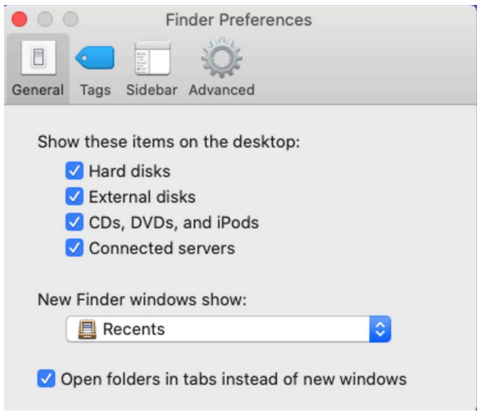
Referring to figure 1-9 — Name the new folder *dev*. You can name it anything you like, actually, but I'm going to assume you named it dev going forward.

1.3 macOS

The macOS user interface can be configured similarly to Microsoft Windows. In this section, I'll show you how to show hidden files and folders, reveal file suffixes, and show the hard drive and other attached media on the Finder desktop.

1.3.1 Show Hard Drives, Connected Servers, and file suffixes

To show your hard drive and any connected drives on the Finder desktop, click on the desktop to make it the active application and then from the Finder menu select **Preferences** or **Settings** to open the Finder Preferences panel as shown in figure 1-10.

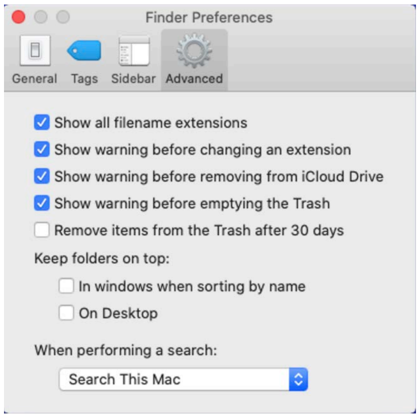


Show these items on the desktop:

- ✓ Hard disks
- ✓ External disks
- ✓ CDs, DVDs, and iPods
- ✓ Connected servers

New Finder windows show: Recents

✓ Open folders in tabs instead of new windows



- ✓ Show all filename extensions
- ✓ Show warning before changing an extension
- ✓ Show warning before removing from iCloud Drive
- ✓ Show warning before emptying the Trash
- ☐ Remove items from the Trash after 30 days

Keep folders on top:

- ☐ In windows when sorting by name
- ☐ On Desktop

When performing a search: Search This Mac

Figure 1-10: Finder Preferences General and Advanced Tabs

Referring to figure 1-10 — Shown are both the Finder Preferences *General* and *Advanced* tabs. In the General tab check all the boxes; in the Advanced tab check the top four boxes. Feel free to customize further as you see fit.

1.3.2 Add Application Aliases To Finder Desktop and Dock

You'll want to add frequently accessed applications to the desktop and dock. I'll show you how to add the Terminal application and suggest others you can add yourself.

Double-click your hard drive icon, which should now be visible in the upper right-hand corner of your desktop, navigate to the **Applications > Utilities** folder and locate the Terminal application. (Terminal.app if you checked Show all filename extensions in Finder Preferences.) Drag the Terminal icon to the dock to add an alias as shown in figure 1-11.

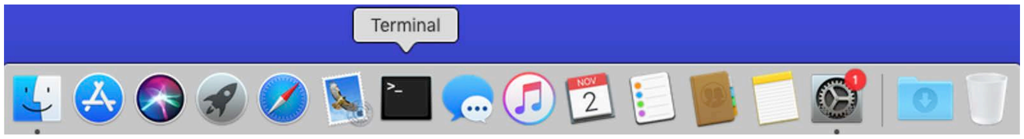


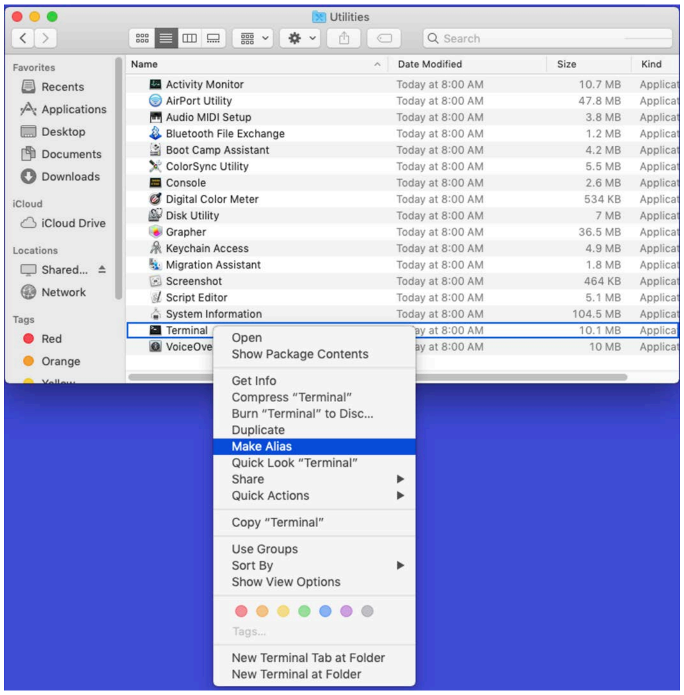
Figure 1-11: Terminal Alias Added to macOS Dock

To add a Terminal alias on the desktop, right-click on the Terminal application in the Utilities folder and from the pop-up menu select **Make alias** as shown in figure 1-12.

Referring to figure 1-12 — This will create an alias in the same directory. Simply drag the alias to the desktop. You can change the name of the alias or leave it as-is. I like to remove the name 'alias' but it's not necessary.

1.3.3 Show Hidden Files and Folders

Double-click your hard drive. This should launch a Finder window as shown in figure 1-13.



Name	Date Modified	Size	Kind
Activity Monitor	Today at 8:00 AM	10.7 MB	Applicat
AirPort Utility	Today at 8:00 AM	47.8 MB	Applicat
Audio MIDI Setup	Today at 8:00 AM	3.8 MB	Applicat
Bluetooth File Exchange	Today at 8:00 AM	1.2 MB	Applicat
Boot Camp Assistant	Today at 8:00 AM	4.2 MB	Applicat
ColorSync Utility	Today at 8:00 AM	5.5 MB	Applicat
Console	Today at 8:00 AM	2.6 MB	Applicat
Digital Color Meter	Today at 8:00 AM	534 KB	Applicat
Disk Utility	Today at 8:00 AM	7 MB	Applicat
Grapher	Today at 8:00 AM	36.5 MB	Applicat
Keychain Access	Today at 8:00 AM	4.9 MB	Applicat
Migration Assistant	Today at 8:00 AM	1.8 MB	Applicat
Screenshot	Today at 8:00 AM	464 KB	Applicat
Script Editor	Today at 8:00 AM	5.1 MB	Applicat
System Information	Today at 8:00 AM	104.5 MB	Applicat
Terminal	ay at 8:00 AM	10.1 MB	Applicat
VoiceOve	ay at 8:00 AM	10 MB	Applicat

Figure 1-12: Right-Click on Application to Create Alias

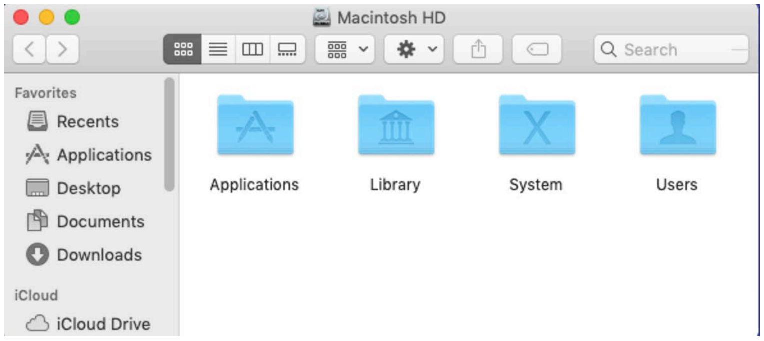
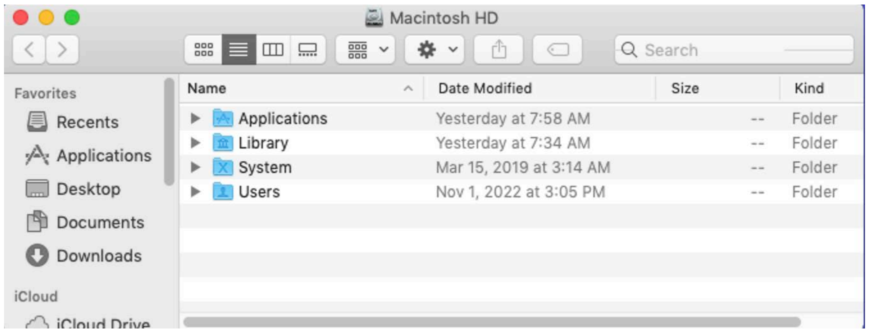


Figure 1-13: Default Finder Window Icon View

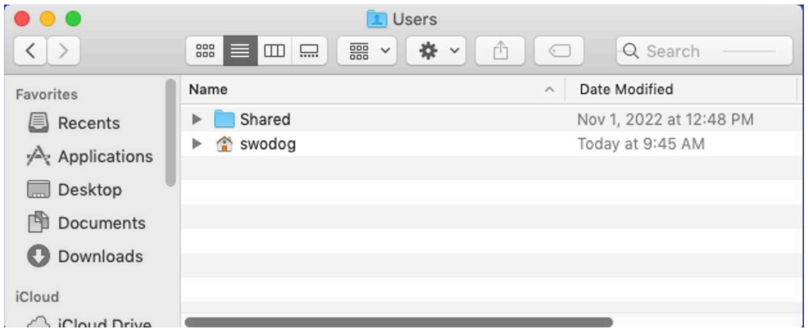
Referring to figure 1-13 — If you haven't bothered to change the default window view this is what you'd normally see when you open a finder window: big icons. I recommend switching to list view as shown in figure 1-14.



Name	Date Modified	Size	Kind
Applications	Yesterday at 7:58 AM	--	Folder
Library	Yesterday at 7:34 AM	--	Folder
System	Mar 15, 2019 at 3:14 AM	--	Folder
Users	Nov 1, 2022 at 3:05 PM	--	Folder

Figure 1-14: List View

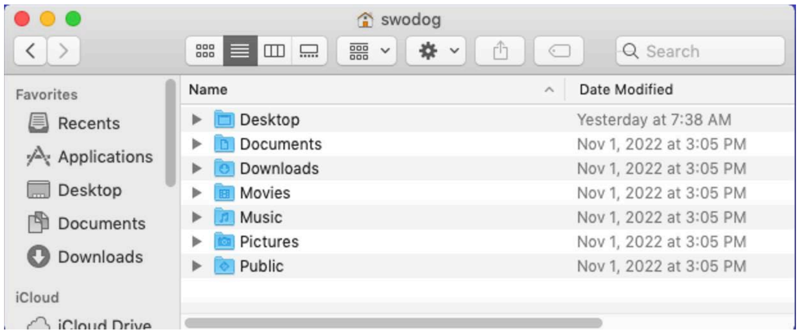
Referring to figure 1-14 — Oh yeah, that's better. Now, navigate to your home directory. This is in Users > *username*, where username is the name you use to log in to your computer. If you're the only one who uses your computer or there's only one account on your computer then there will be only two or three folders in the Users folder as shown in figure 1-15.



Name	Date Modified
Shared	Nov 1, 2022 at 12:48 PM
swodog	Today at 9:45 AM

Figure 1-15: Users Folder

Referring to figure 1-15 — This shows the contents of my Users folder. There's a Shared folder and a folder named swodog, which is my home folder and also the name of the account I use to log into my computer. **NOTE:** You may also have a folder named *Guest* if you've ever enabled the Guest account. — OK, navigate to your home directory. Mine is shown in figure 1-16.

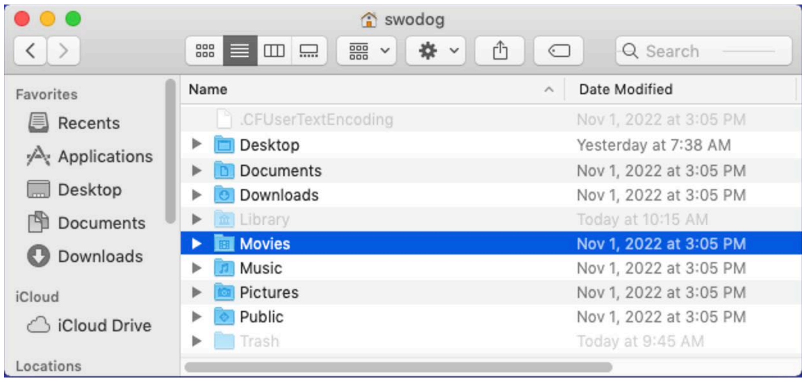


Name	Date Modified
Desktop	Yesterday at 7:38 AM
Documents	Nov 1, 2022 at 3:05 PM
Downloads	Nov 1, 2022 at 3:05 PM
Movies	Nov 1, 2022 at 3:05 PM
Music	Nov 1, 2022 at 3:05 PM
Pictures	Nov 1, 2022 at 3:05 PM
Public	Nov 1, 2022 at 3:05 PM

Figure 1-16: My Home Directory

Referring to figure 1-16 — This represents a fresh install of the macOS operating system. You may have more files and folders in your home directory. To show hidden files and folders make sure your home directory window is selected and type these three keys: **Command** + **Shift** + '**.**' — which means press and hold the *Command* (⌘ cmd) key followed by the *Shift* key and finally the dot or *period* key so that all three keys are pressed down at the same time — then let up. Hidden files and folders will appear light gray as shown in figure 1-17.

Referring to figure 1-17 — As you install and configure software and various tools, you'll see more hidden files and folders appear in your home directory. A word of caution — with great power comes great responsibility. Apple hides some of these files and folders for good reason. Tread cautiously. Also, this is only in effect until you either log out or restart the computer. I don't think that's an issue. When you learn about terminals in the next section, you'll be able to list hidden files and folders any time you like. And really, you don't always need to see them, just know how to see them when you need to. To hide them again simply repeat the **Command** + **Shift** + '**.**' key command.



Name	Date Modified
.CFUserTextEncoding	Nov 1, 2022 at 3:05 PM
Desktop	Yesterday at 7:38 AM
Documents	Nov 1, 2022 at 3:05 PM
Downloads	Nov 1, 2022 at 3:05 PM
Library	Today at 10:15 AM
Movies	Nov 1, 2022 at 3:05 PM
Music	Nov 1, 2022 at 3:05 PM
Pictures	Nov 1, 2022 at 3:05 PM
Public	Nov 1, 2022 at 3:05 PM
Trash	Today at 9:45 AM

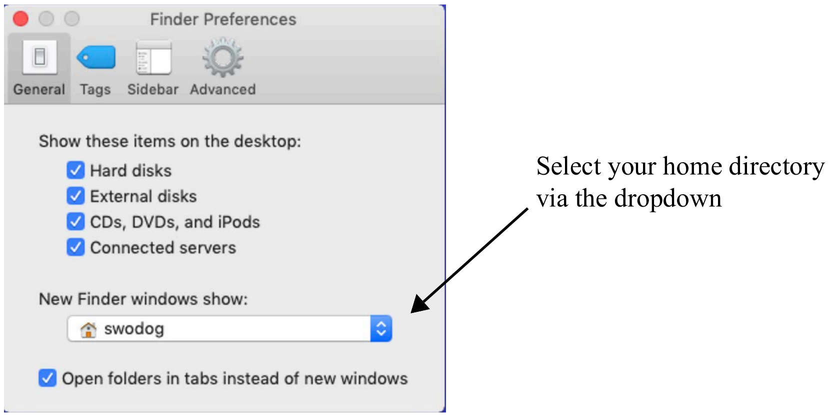
Figure 1-17: Hidden Files and Folders Appear as Light Gray Text

1.3.4 Create Development Projects Folder

While you're in your home directory create a new folder named **dev**. Simply right-click in your home directory window and select **New Folder** from the pop-up menu. To speed access to your dev folder create an alias and drag it to the desktop.

1.3.5 Open Finder Windows In Home Directory

Return once again to the Finder Preferences General Tab and set the **New Finder windows show:** dropdown to your home directory as shown in figure 1-18.



Show these items on the desktop:

- ✓ Hard disks
- ✓ External disks
- ✓ CDs, DVDs, and iPods
- ✓ Connected servers

New Finder windows show: swodog

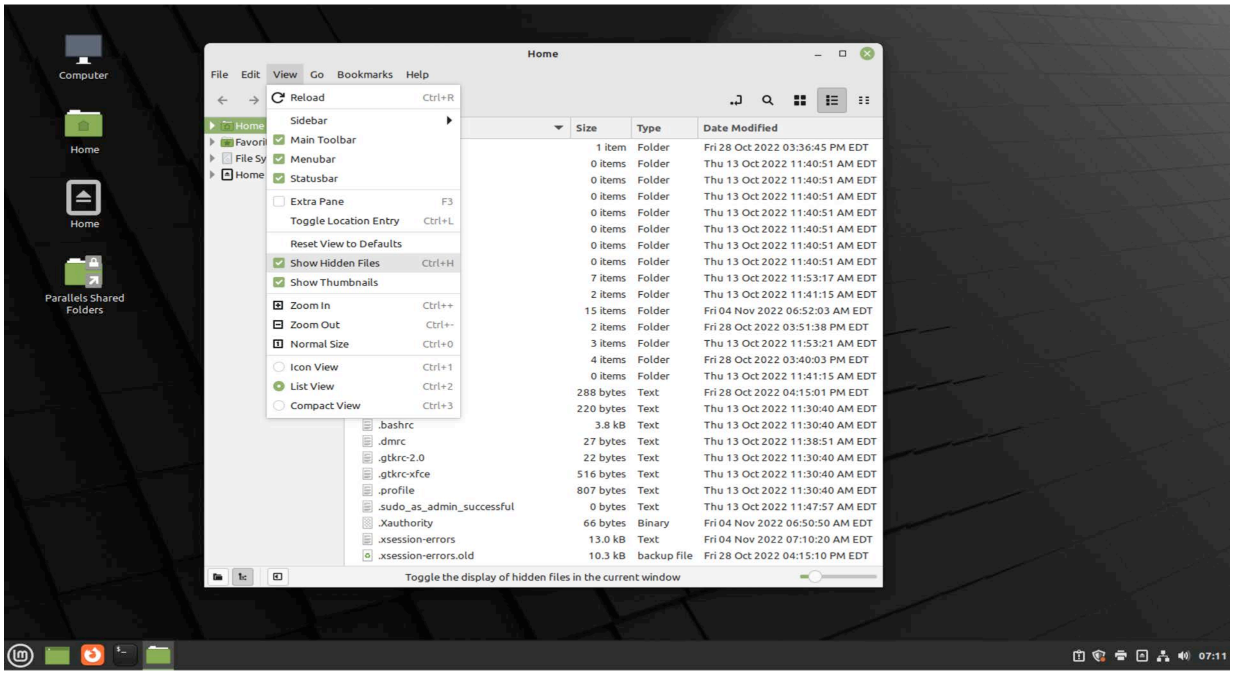
✓ Open folders in tabs instead of new windows

Figure 1-18: Set New Finder Windows to Open in Home Directory

Referring to figure 1-18 — To test this out select **File > New Finder Window** from the main menu or use the keyboard shortcut **Command + 'N'** to open a new Finder window in your home directory.

1.4 Linux

There are so many Linux variants in the wild there's no telling which one you might be using, so I'll target *Linux Mint*, a Debian and Ubuntu derivative running the *Cinnamon* desktop. Linux Mint is perhaps the most novice-friendly Linux there is and the Cinnamon desktop delivers a look-and-feel mashup between macOS and Microsoft Windows. A fresh Linux Mint install with the Cinnamon desktop comes fairly well configured as is shown in figure 1-19.



	Size	Type	Date Modified
	1 item	Folder	Fri 28 Oct 2022 03:36:45 PM EDT
	0 items	Folder	Thu 13 Oct 2022 11:40:51 AM EDT
	0 items	Folder	Thu 13 Oct 2022 11:40:51 AM EDT
	0 items	Folder	Thu 13 Oct 2022 11:40:51 AM EDT
	0 items	Folder	Thu 13 Oct 2022 11:40:51 AM EDT
	0 items	Folder	Thu 13 Oct 2022 11:40:51 AM EDT
	0 items	Folder	Thu 13 Oct 2022 11:40:51 AM EDT
	0 items	Folder	Thu 13 Oct 2022 11:40:51 AM EDT
	7 items	Folder	Thu 13 Oct 2022 11:53:17 AM EDT
	2 items	Folder	Thu 13 Oct 2022 11:41:15 AM EDT
	15 items	Folder	Fri 04 Nov 2022 06:52:03 AM EDT
	2 items	Folder	Fri 28 Oct 2022 03:51:38 PM EDT
	3 items	Folder	Thu 13 Oct 2022 11:53:21 AM EDT
	4 items	Folder	Fri 28 Oct 2022 03:40:03 PM EDT
	0 items	Folder	Thu 13 Oct 2022 11:41:15 AM EDT
	288 bytes	Text	Fri 28 Oct 2022 04:15:01 PM EDT
	220 bytes	Text	Thu 13 Oct 2022 11:30:40 AM EDT
.bashrc	3.8 kB	Text	Thu 13 Oct 2022 11:30:40 AM EDT
.dmrc	27 bytes	Text	Thu 13 Oct 2022 11:38:51 AM EDT
.gtkrc-2.0	22 bytes	Text	Thu 13 Oct 2022 11:30:40 AM EDT
.gtkrc-xfce	516 bytes	Text	Thu 13 Oct 2022 11:30:40 AM EDT
.profile	807 bytes	Text	Thu 13 Oct 2022 11:30:40 AM EDT
.sudo_as_admin_successful	0 bytes	Text	Thu 13 Oct 2022 11:47:57 AM EDT
.Xauthority	66 bytes	Binary	Fri 04 Nov 2022 06:50:50 AM EDT
.xsession-errors	13.0 kB	Text	Fri 04 Nov 2022 07:10:20 AM EDT
.xsession-errors.old	10.3 kB	backup file	Fri 28 Oct 2022 04:15:10 PM EDT

Toggle the display of hidden files in the current window

Figure 1-19: Linux Mint with Cinnamon Desktop

Referring to figure 1-19 — You can see the desktop already has icons for the Computer and the Home folder. There's also a shortcut to the Terminal on the *panel*, which is Cinnamon's version of the dock or taskbar, although it's hard to see in the image because its icon is black set against a dark gray panel but you can just make out the white characters $_ which represent a shell prompt.

1.4.1 View Hidden Files And Folders

To view hidden files and folders click the green folder located on the panel to launch the file manager. Notice the file manager opens in your home directory. From the **View** menu check **Show Hidden Files** or use the key combination **Ctrl + H**. While you're at it, also from the View menu, change the Sidebar to **Tree**. I find this most helpful. Finally, click the **List** layout icon.

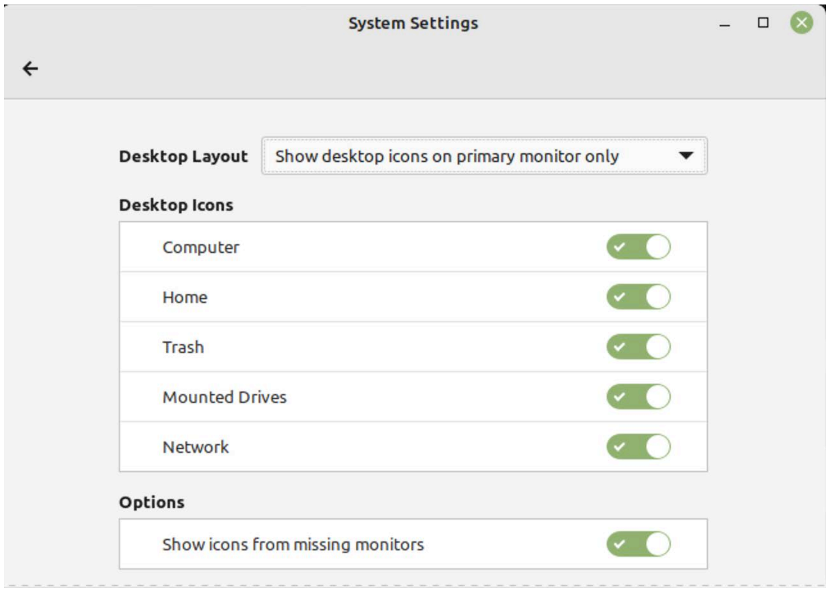
1.4.2 Create Development Project Folder

Right-click in your home folder and select **Create New Folder** from the pop-up menu. Name it dev.

1.4.3 Add Missing Icons To Desktop

What I don't see on the desktop is a Trash or Recycle Bin icon. It's probably not a big deal but may as well show it. In the lower left-hand corner of the screen click the circular **LM icon (Menu)** > **System Settings** > **Desktop** to display the desktop settings as shown in figure 1-20.

Referring to figure 1-20 — Click the buttons for *Trash* and *Network* to On. You want all the buttons to be green. You should now see the Trash and Network icons on the desktop.


System Settings

Desktop Layout: Show desktop icons on primary monitor only

Desktop Icons

- Computer ✓
- Home ✓
- Trash ✓
- Mounted Drives ✓
- Network ✓

Options

- Show icons from missing monitors ✓

Figure 1-20: Desktop Settings

1.5 Parting Thoughts

That about covers the initial housekeeping setup for all three operating systems. As you can see, if you squint hard enough, they all look vaguely alike — for the most part. Of course you can perform further customization and personalization if you so desire but this is a good start.

Quick Review

The purpose of initial housekeeping configuration is to provide quick access to frequently used files, folders, and applications. Although terminology may be different between macOS, Linux, and Microsoft Windows for similar concepts, the objectives are the same. You want to see hidden files and folders, especially in your home directory, and you want quick and easy access to frequently used applications, folders, and files on either your desktop, taskbar, or both.

The most important configuration you can make is to ensure your home directory name has no spaces.

2 Terminal Applications

As a software developer, you will spend a lot of time using a *terminal application* to interact with your computer. While using a terminal may come as a shock to novice programmers who until now have only interacted with a computer via a graphical user interface, you will find, as you gain skill and learn a handful of important commands, you can get way more done and move a zillion times faster via the command line than fiddling with mice and menus. In many cases the *only* way to get something done as a programmer is via the command line.

2.1 What Is A Terminal?

A *terminal*, *terminal application*, or *terminal emulator* is a text-based interface to a computer that runs some type of *command interpreter* or *command shell*.

The names of these terminal apps will vary depending on the operating system. Linux and macOS systems come with **Terminal** applications whereas Microsoft Windows ships with **PowerShell** and **Command Prompt** applications. All these are referred to loosely as *command-line interfaces*. Regardless of their name their purpose is the same; they enable a user to interact with the computer via text commands.

Terminal applications are responsible for translating user input into a format (codes and control signals) the computer understands, and rendering output from the computer into a format the user understands or desires. Examples of output formatting include character color and highlighting, column arrangement, etc.

Terminal applications run a program called a *shell*. The shell determines what types of commands can be entered and how they are interpreted.

2.1.1 Some History

In the early days of computing an operator entered commands and data directly into the computer either via patch cables or switches. There were no operating systems or terminals. This approach to human-computer interaction was both time consuming and error prone and was eventually followed by the use of dedicated teletype terminals as shown in figure 1-21.

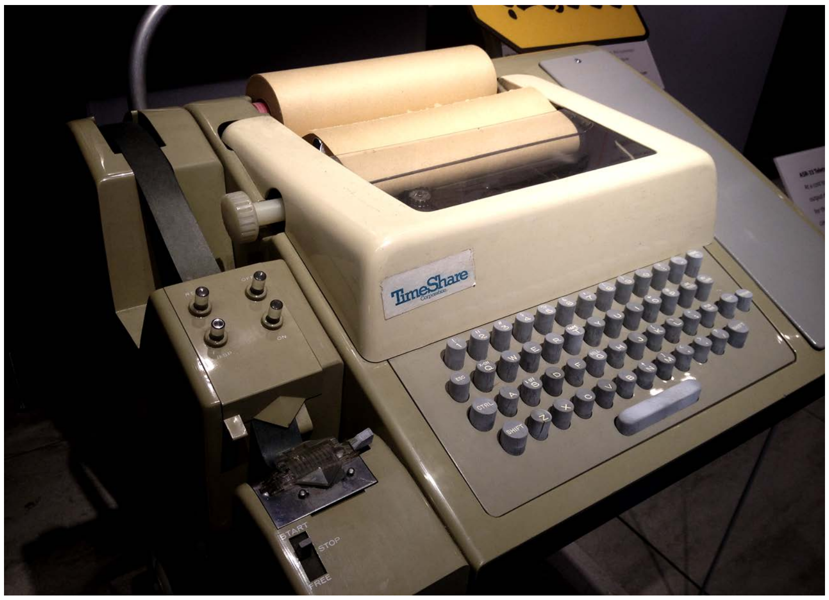


Figure 1-21: Teletype Computer Terminal with Attached Paper Tape Punch/Reader

Referring to figure 1-21 — A teletype is an electromechanical device which combines the features of a typewriter and printer. A lot of the codes used to control a teletype are still used in programs today. One need only examine the ASCII table located in Appendix B for familiar examples, which include the carriage return '`\r`' and line feed or new line '`\n`' characters. The teletype also gave us an abbreviation still used today in UNIX® and Linux systems: *tty*.

As computers evolved so too did the terminal devices used to work with them. Teletypes gave way to terminals with an integrated or detachable keyboard and cathode ray tube (CRT) screen as is shown in figure 1-22.

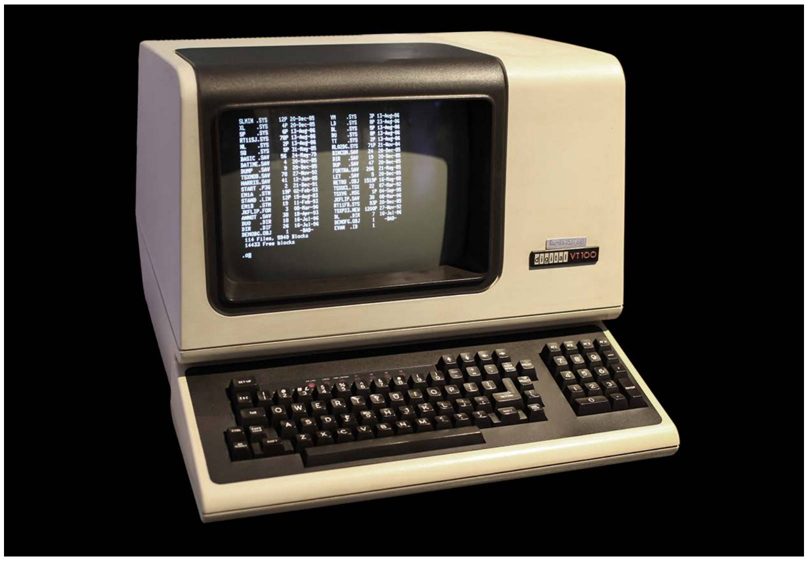


Figure 1-22: DEC VT100 Intelligent Terminal

Referring to figure 1-22 — This is an example of a Digital Equipment Corporation (DEC) Video Terminal 100 or VT100. This was an *intelligent terminal* in that it employed a microprocessor to assist with display formatting and keyboard processing. It did no processing on its own other than to manage the screen and keyboard. The sole job of a terminal like this is to send input to and receive output from the main computer, which would have been a DEC PDP, VAX, or related system.

The VT100 still lives on today especially for those who use terminal applications on UNIX® or Linux machines, namely that most terminal applications are actually **terminal emulators** and can emulate a wide range of terminals, including the VT100.

You don't generally need to worry what terminal your terminal application is emulating because they are set to work properly by default with your operating system. However, if you establish a remote connection to an older computer you may need to pay attention to the terminal emulation setting.

Note that the evolution of terminals and other types of equipment used to interact with computers developed along with the evolution of operating systems.

2.2 What Is A Shell?

A shell is a program your terminal launches when you log in to the computer. The shell dictates what commands you can use and how you can string them together to get things done on the machine.

Popular shells on UNIX®/Linux operating systems include the *Born Again Shell (Bash)* and the *Z Shell (zsh)*. Different shells have different capabilities and features. You can automate tasks using shell scripts, something you'll learn how to do later in the book.

You can also change the shell in UNIX® and Linux terminals but on Microsoft Windows the included terminals and their shells are tightly coupled. PowerShell is more powerful and extensible than the Command Prompt but the later is still useful in its own right. You can automate tasks in the Command Prompt by writing batch files which have a file suffix of (.bat). You can write scripts in PowerShell and create custom commands called cmdlets in C#. PowerShell scripts have the file suffix (.ps1).

2.3 Standardizing On Bash

I will use *Bash* as the shell and shell scripting language of choice throughout this book. I will use Bash scripts to run Python programs and the Bash scripts will run regardless of operating system. This is not an issue on Linux and macOS, but on Windows you'll need to install Git for Windows to have a Bash terminal. On Windows 11, you may need to install the Windows Subsystem for Linux (WSL).

2.4 Windows

This section explains how to install and configure Git for Windows, which provides a Bash terminal (mintty).

2.4.1 Git For Windows

To get a Bash terminal on Windows 10 & 11 without installing the Windows Subsystem for Linux (WSL), download and install *Git for Windows*, which is based on MSYS2 *https://www.msys2.org/docs/what-is-msys2/* and provides all the tools you need to perform Git Software Configuration Management (Git SCM), which will come in handy when I discuss software configuration management with Git and GitHub in Part II, Chapters 7 and 8.

2.4.1.1 Installation

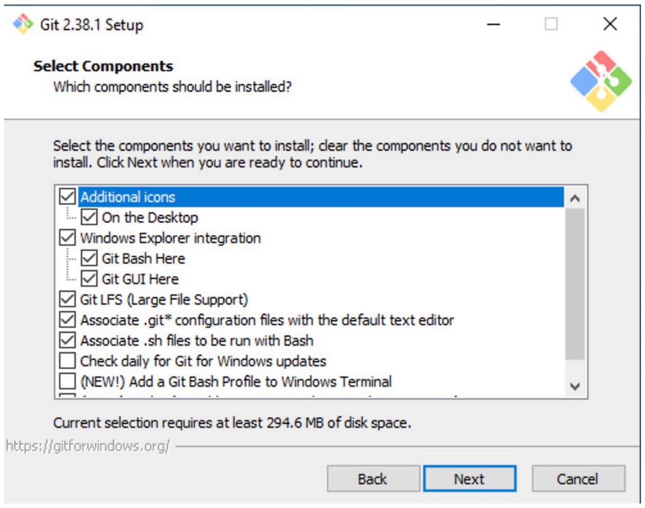
On your Windows machine open a browser and navigate to *https://gitforwindows.org* and click the **Download** button. Launch the installer and accept the default installation location. Click **Next** and check the box at the top of the list to install additional icons on the desktop as shown in figure 1-23.

Referring to figure 1-23 — Ensure the Additional icons box is checked and click the **Next** button. Click **Next** again until you arrive at the **Choosing the default editor used by Git** dialog as shown in figure 1-24.

Referring to figure 1-24 — Accept the default editor (Vim) for now. You can change it later if you so desire. Click the **Next** button to proceed. You'll need to make a decision on the **Adjusting the name of the initial branch in new repositories** dialog as shown in figure 1-25.

Referring to figure 1-25 — Click the **Override the default branch name for new repositories** radio button and leave the name `main`. Click the **Next** button to show the **Adjusting your PATH environment** dialog as shown in figure 1-26.

Referring to figure 1-26 — Click the **middle radio button** to use git from the command line and also from 3rd party software. Click **Next** and accept the default settings for the remaining dialogs. On the last dialog **Configuring experimental options** leave both boxes unchecked. Click the **Install** button. When installation completes, click the **Finish** button. You'll see a Git Bash



Git 2.38.1 Setup

Select Components
Which components should be installed?

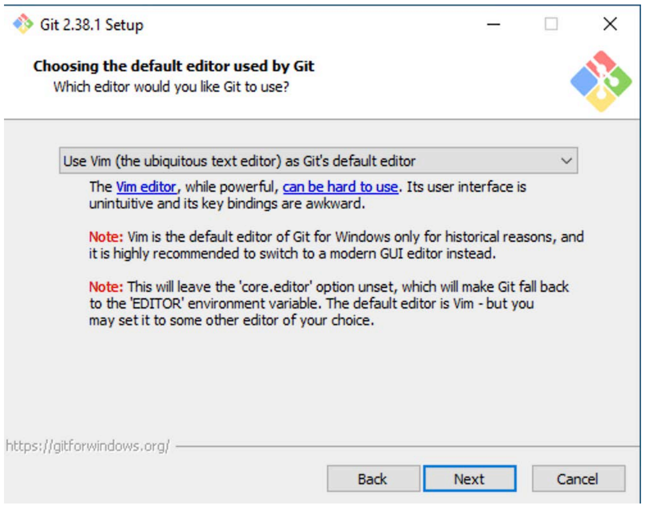
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [x] Additional icons
  - [x] On the Desktop
- [x] Windows Explorer integration
  - [x] Git Bash Here
  - [x] Git GUI Here
- [x] Git LFS (Large File Support)
- [x] Associate .git* configuration files with the default text editor
- [x] Associate .sh files to be run with Bash
- [ ] Check daily for Git for Windows updates
- [ ] (NEW!) Add a Git Bash Profile to Windows Terminal

Current selection requires at least 294.6 MB of disk space.

https://gitforwindows.org/

Figure 1-23: Check Additional Icons Box



Git 2.38.1 Setup

Choosing the default editor used by Git
Which editor would you like Git to use?

Use Vim (the ubiquitous text editor) as Git's default editor

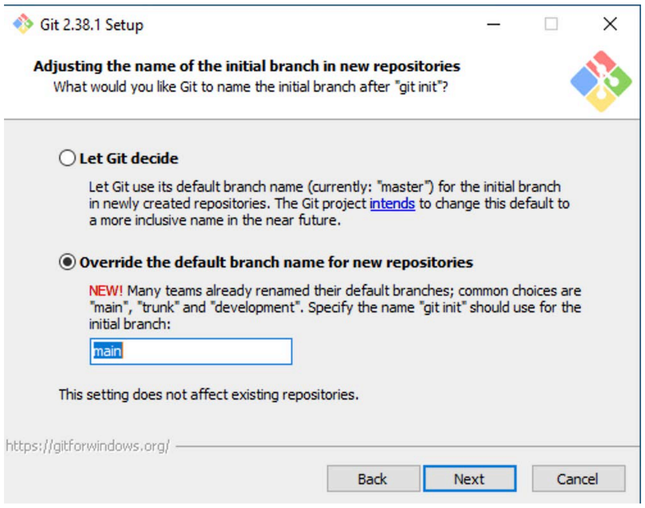
The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.

Note: Vim is the default editor of Git for Windows only for historical reasons, and it is highly recommended to switch to a modern GUI editor instead.

Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.

https://gitforwindows.org/

Figure 1-24: Use Vim as Default Editor



Git 2.38.1 Setup

Adjusting the name of the initial branch in new repositories
What would you like Git to name the initial branch after "git init"?

○ **Let Git decide**

Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.

● **Override the default branch name for new repositories**

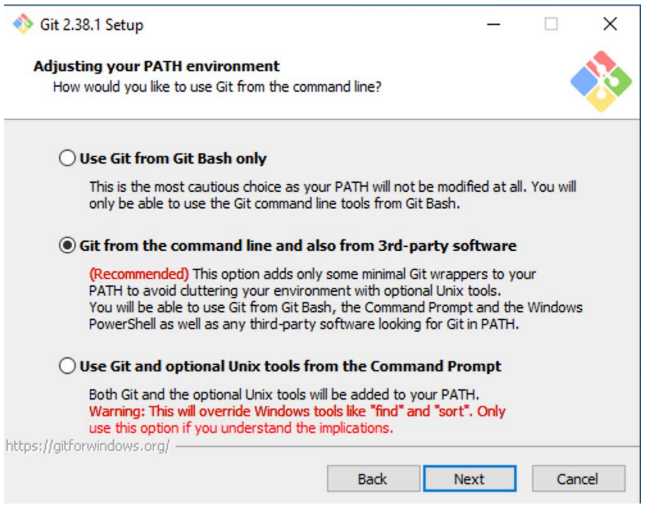
NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:

main

This setting does not affect existing repositories.

https://gitforwindows.org/

Figure 1-25: Check the Override Button and Name New Branches main



Git 2.38.1 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

○ **Use Git from Git Bash only**

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

[✓ **Git from the command line and also from 3rd-party software**]

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

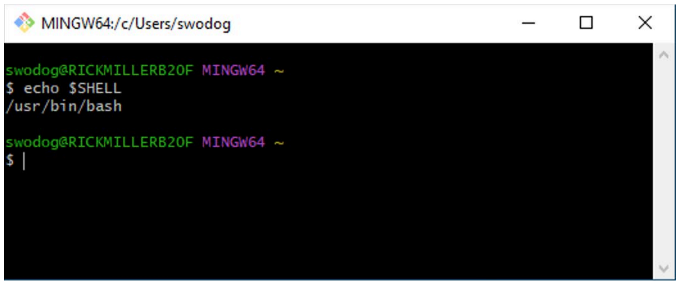
○ **Use Git and optional Unix tools from the Command Prompt**

Both Git and the optional Unix tools will be added to your PATH. Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://gitforwindows.org/

Figure 1-26: Click the Middle Radio Button

icon on your desktop. Right click the icon and from the pop-up menu select **Pin to taskbar**. Launch Git Bash to open a terminal window as shown in figure 1-27.



MINGW64:/c/Users/swodog

```
swodog@RICKMILLERB20F MINGW64 ~
$ echo $SHELL
/usr/bin/bash

swodog@RICKMILLERB20F MINGW64 ~
$ |
```

Figure 1-27: Git Bash Terminal Running on Windows

Referring to figure 1-27 — At the command prompt type `echo $SHELL` to see what shell is running. The output will be `/usr/bin/bash`. You now have a Bash shell running in a terminal on Microsoft Windows.

2.4.1.2 Changing Terminal Window Properties

You can customize many terminal window properties. Click on the Git Bash icon in the upper left corner of the terminal window and select **Options...** from the dropdown menu as shown in figure 1-28. This will open the Options dialog shown in figure 1-29.

Referring to figure 1-29 — Listed in the left-hand column are option groups. I've selected the Terminal options group to show you that the default terminal is set to *xterm*. You can see VT100 is also listed on the dropdown list. Leave xterm as the default emulation and explore the other option groups. I like to change the default terminal window size to make it larger when it opens as well as the color scheme.

2.4.2 Exploring The Git Bash (mintty) Terminal

Launch your Git Bash terminal if it's not already open. Notice it opens in your home directory, the path to which is displayed in the upper-right corner of the terminal window's title bar as shown in figure 1-30.

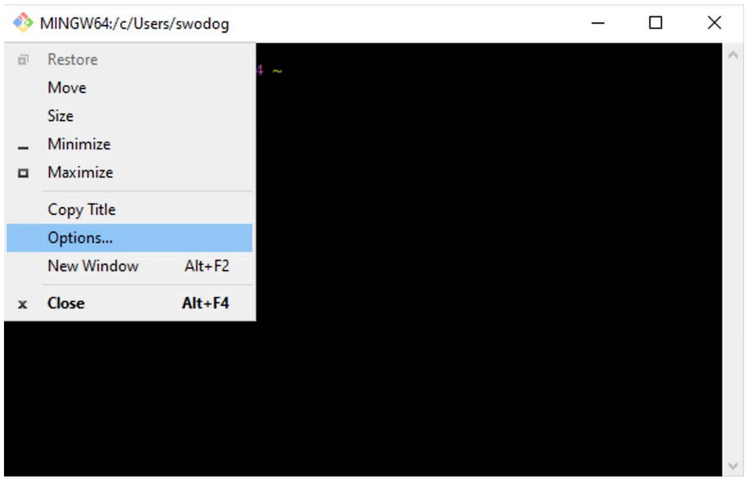


Figure 1-28: Git Bash Terminal Window Options

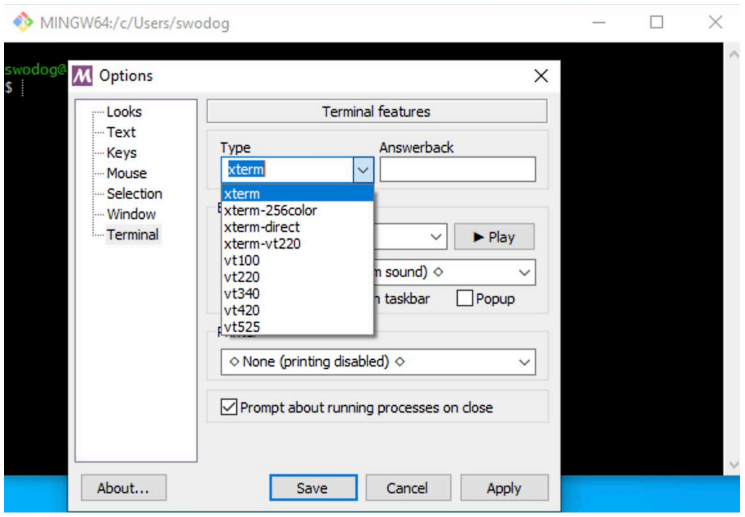
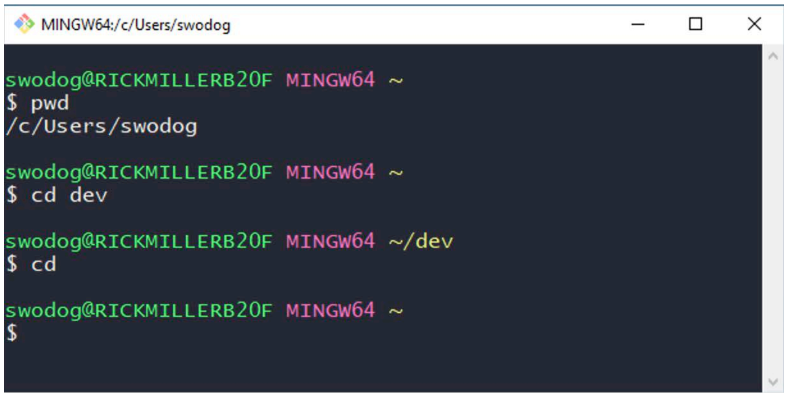


Figure 1-29: Git Bash Terminal Options



```
swodog@RICKMILLERB20F MINGW64 ~
$ pwd
/c/Users/swodog

swodog@RICKMILLERB20F MINGW64 ~
$ cd dev

swodog@RICKMILLERB20F MINGW64 ~/dev
$ cd

swodog@RICKMILLERB20F MINGW64 ~
$
```

Figure 1-30: Exploring The Git Bash (mintty) Terminal

Referring to figure 1-30 — To quickly find out what directory you're in you can also type `pwd` at the command-line prompt. The command `pwd` stands for **p**rint **w**orking **d**irectory. Also note that the tilde '~' character represents your home directory.

To change to a different directory use the command `cd`, which stands for **c**hange **d**irectory. At the prompt type `cd dev` to change to the `dev` directory. You'll see the path change in the title bar as well as in the terminal to *~/dev*. To return to your home directory type `cd`.

2.4.3 Parting Thoughts On Git For Windows

Git for Windows and the Git Bash terminal provide all the Bash features you'll need for this book. It's not a full-blown Linux system but it supports enough commands and features to enable Bash scripts to run on Windows as well as perform Git repository operations.

2.5 MacOS

MacOS comes with a perfectly good terminal emulator but I prefer to use *iTerm2* and if you're a Mac user I think you'll like it, too. But first, launch the default terminal application and make Bash your default shell with the following command:

```
chsh -s /bin/bash
```

Enter your password when prompted and hit return. Close the terminal window.

2.5.1 Install iTerm2

Navigate to the iTerm2 website *https://iterm2.com* and click the **Download** button. It downloads a zip file and automatically unpacks in your Downloads folder. Navigate to your Downloads folder and drag the iTerm.app file to your *~/Applications* folder. That is, drag the file to the Applications folder located in your home directory. If you don't have an Applications folder in your home directory, launch the Terminal application and at the prompt type `mkdir Applications` to create it. The command `mkdir` stands for **m**a**k**e **dir**ectory. Now, copy the iTerm.app file into your *~/Applications* folder. Your home directory should now look like figure 1-31.

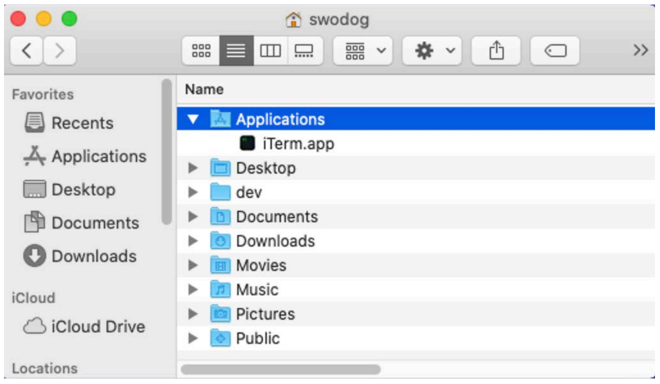
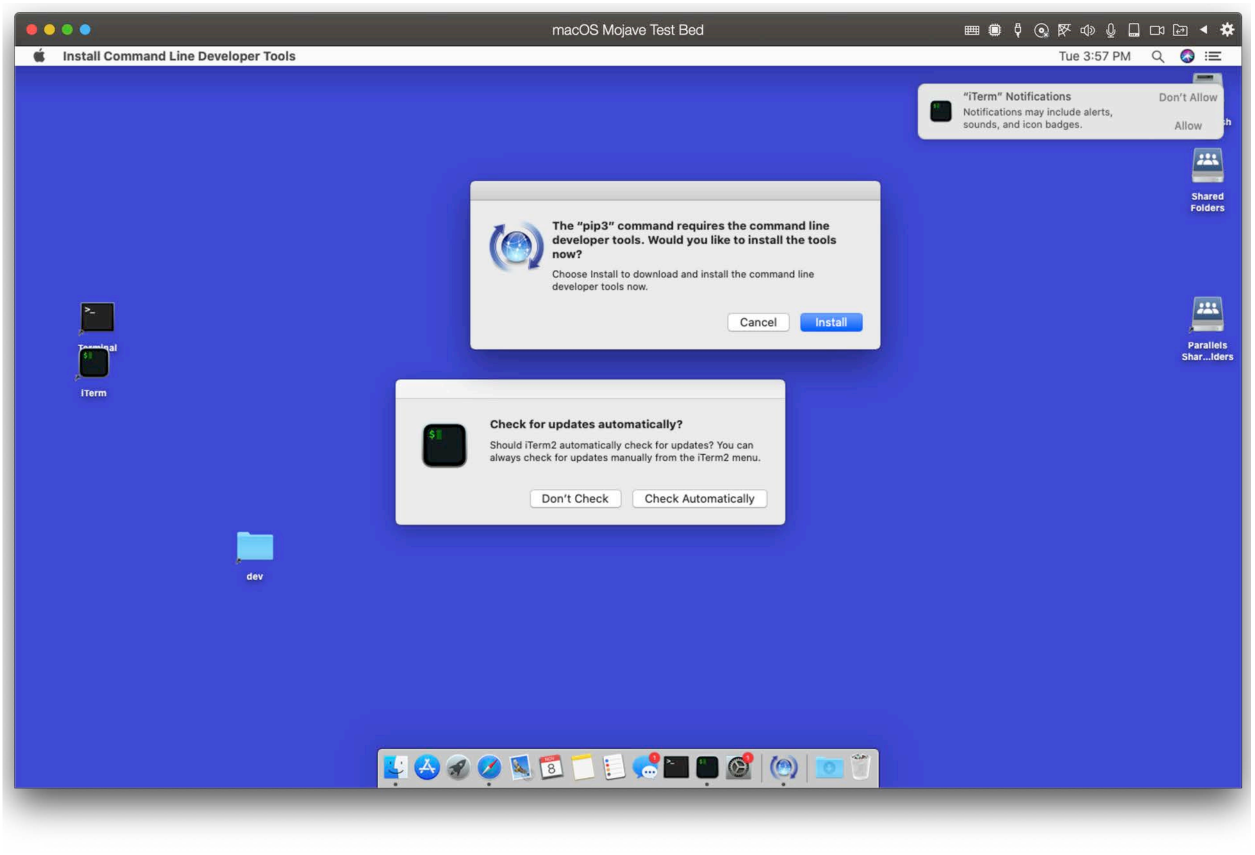


Figure 1-31: Home Directory with Applications Folder and iTerm.app

Referring to figure 1-31 — Drag iTerm to the dock, then make an alias and drag it to the desktop. Launch iTerm2. You'll see several pop-up messages, one that warns iTerm is an application downloaded from the Internet. Click **Open**. Then you'll see several pop-ups seemingly all at once as shown in figure 1-32.



"iTerm" Notifications
Notifications may include alerts, sounds, and icon badges.

The "pip3" command requires the command line developer tools. Would you like to install the tools now?
Choose Install to download and install the command line developer tools now.

Check for updates automatically?
Should iTerm2 automatically check for updates? You can always check for updates manually from the iTerm2 menu.

Figure 1-32: iTerm Popups

Referring to figure 1-32 — Allow iTerm Notifications. In the **Check for updates automatically** dialog click **Check Automatically**. At that point you'll see the iTerm terminal window appear.

Finally, in the dialog that says **The "pip3" command requires the command-line developer tools**, click the **Cancel** button. We have bigger fish to fry.

2.5.2 Install Apple Xcode and Command-Line Developer Tools

Xcode is Apple's integrated development environment (IDE) for building software for Apple platforms. The version of Xcode and the command-line tools you need to install depends on the version of macOS you are running. If you're running the latest version of macOS you will be able to obtain Xcode and the command-line tools from the Apple App Store. Note that Xcode and the corresponding command-line tools are separate downloads.

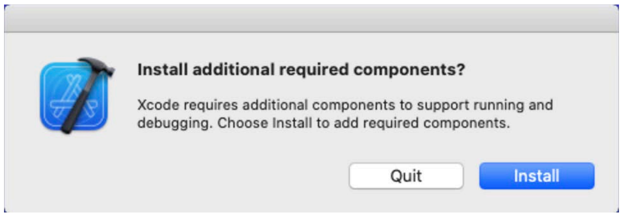
If you're running older versions of macOS you'll need to download the corresponding version of Xcode and command-line tools from Apple's developer website: *https://developer.apple.com* The tools and examples in this book are tested against macOS versions from Mojave (10.14.6) up to Tahoe (26.1). Table 1-1 lists macOS versions from Monterey onwards with their corresponding supported Xcode versions.

macOS Version	Xcode Version
Tahoe (26+)	Xcode 26.1
Sequoia (15+)	Xcode 16.4
Sonoma (14+)	Xcode 15.4
Ventura (13+)	Xcode 14.2
Monterey (12.5+)	Xcode 14.1
Source — Xcode Releases: *https://xcodereleases.com*	

Table 1-1: macOS Versions and Supported Xcode Versions

Referring to table 1-1 — I recommend visiting the Xcode Releases website *https://xcodereleases.com* because they provide direct download links to the tools on Apple's Developer website, which saves you a ton of time searching. Also note that every Xcode version has a corresponding version of the Xcode command-line tools. You need to download and install those as well.

When you first launch Xcode you'll see a pop-up dialog saying you need to install additional components as shown in figure 1-33.


Install additional required components?

Xcode requires additional components to support running and debugging. Choose Install to add required components.

Figure 1-33: Install Additional Xcode Components

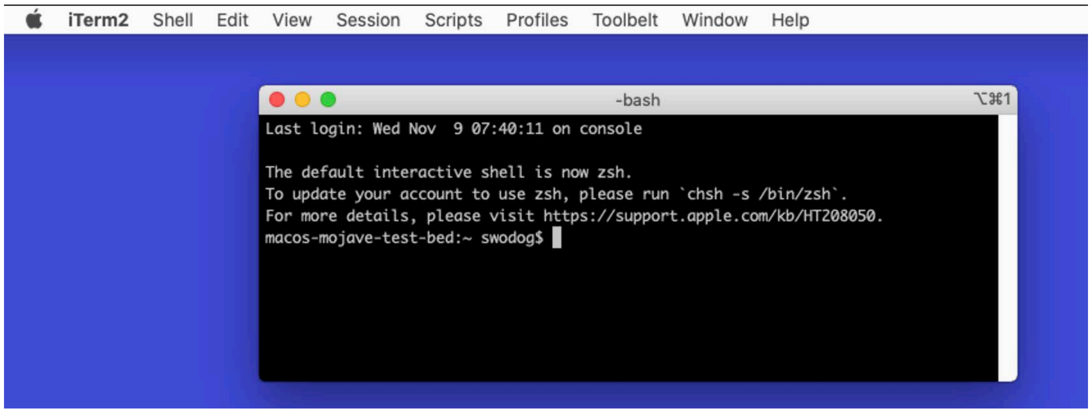
Referring to figure 1-33 — Click **Install** to install the additional components. When installation completes, Xcode will launch. Install the Xcode command-line tools and when you've completed that task you can return to iTerm2 terminal configuration.

2.5.3 iTerm2 Terminal Configuration

When you've completed Xcode and Xcode command-line tools installation launch iTerm2. The first thing you'll notice is there are a whole lot more menu items to choose from verses what the default macOS Terminal application offers as is shown in figure 1-34.

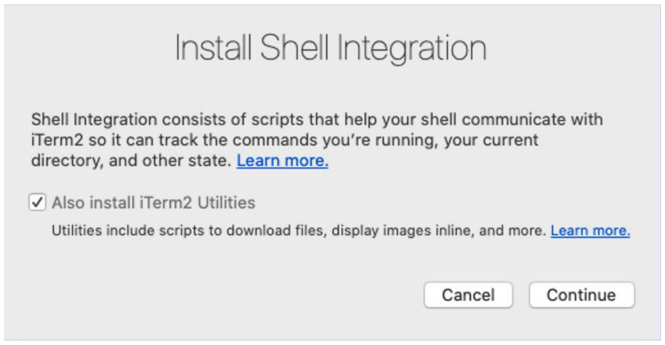
Referring to figure 1-34 — First thing you'll want to do is to install *Shell Integration*. From the **iTerm2** menu select **Install Shell Integration**. This will open a pop-up dialog as shown in figure 1-35.

Referring to figure 1-35 — Leave the **Also install iTerm2 Utilities** box checked and click **Continue**. On the next window click **Download and Run Installer** then **Continue**. You will see scripts execute in the iTerm2 terminal window and when installation completes you'll see an Installation Complete dialog. Click **OK** to complete installation. Quit iTerm2 and relaunch.



```
Last login: Wed Nov  9 07:40:11 on console

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
macos-mojave-test-bed:~ swodog$ 
```

Figure 1-34: iTerm2 Menu Bar Choices



Install Shell Integration

Shell Integration consists of scripts that help your shell communicate with iTerm2 so it can track the commands you're running, your current directory, and other state. Learn more.

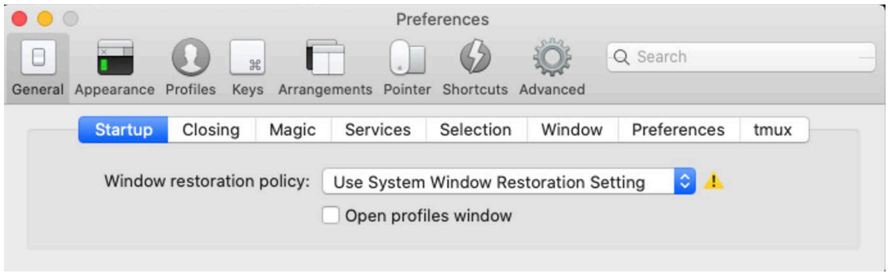
[✓] Also install iTerm2 Utilities

Utilities include scripts to download files, display images inline, and more. Learn more.

Figure 1-35: Install Shell Integration Dialog Window

2.5.3.1 iTerm2 Preferences

Click **iTerm2 > Preferences** to open the Preferences dialog as shown in figure 1-36.

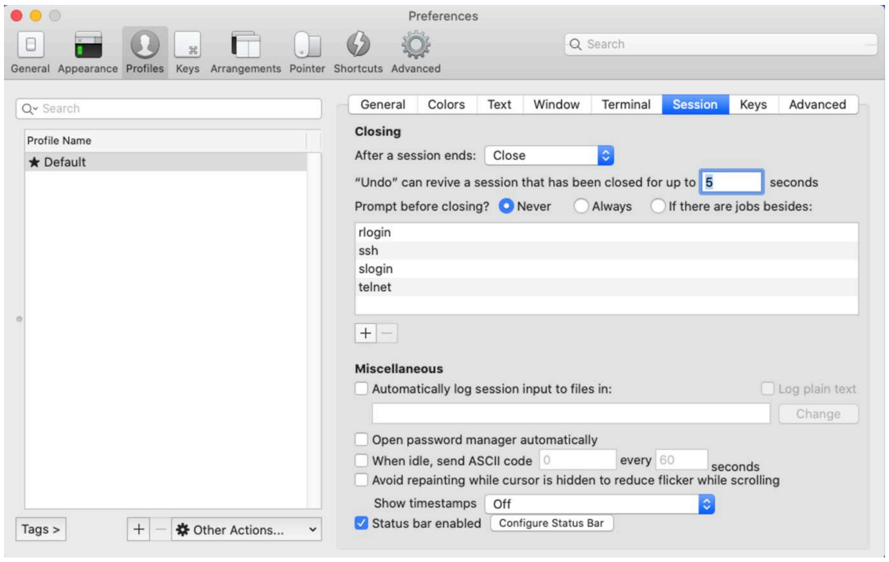


Window restoration policy: Use System Window Restoration Setting

[ ] Open profiles window

Figure 1-36: iTerm2 Preferences Dialog

Referring to figure 1-36 — There are too many iTerm2 configuration options to cover in this chapter and much depends on your particular preferences with regards to how you want the terminal to look and behave. I do recommend tweaking the color scheme, font size, and window dimensions to your suit your taste. In this regard, iTerm2 is similar to Terminal. One particular feature I like and use which is unique to iTerm2 is the *status bar*. To show the status bar, click on the **Profiles** icon then select the **Session** tab and at the bottom of the window check the **Status bar enabled** check box as shown in figure 1-37.



Closing

After a session ends: Close

"Undo" can revive a session that has been closed for up to 5 seconds

Prompt before closing? [✓ Never] Always If there are jobs besides:

rlogin
ssh
slogin
telnet

Miscellaneous

Automatically log session input to files in: Log plain text Change

Open password manager automatically

When idle, send ASCII code 0 every 60 seconds

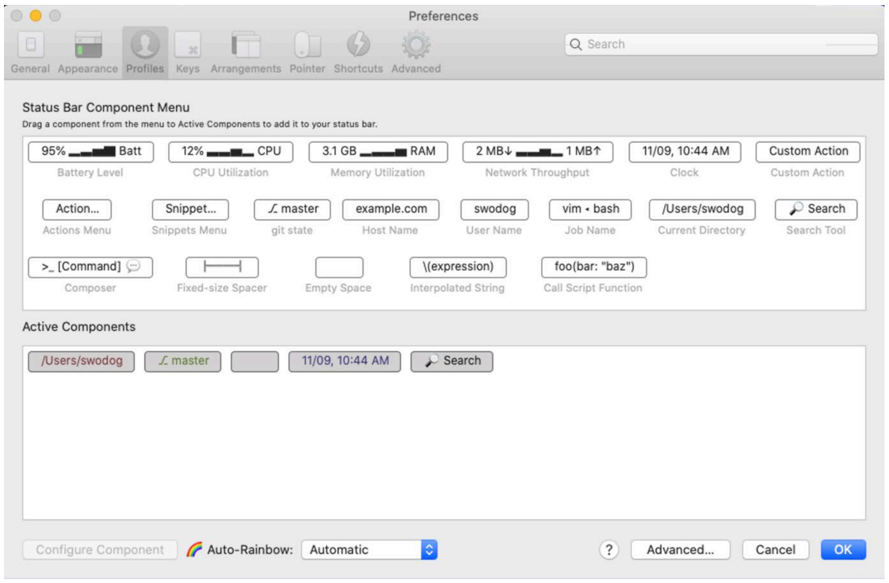
Avoid repainting while cursor is hidden to reduce flicker while scrolling

Show timestamps Off

[✓ Status bar enabled] Configure Status Bar

Figure 1-37: Enabling iTerm2 Status Bar

Referring to figure 1-37 — Enabling the status bar will also enable the **Configure Status Bar** button. Go ahead and click that button to launch the status bar configuration window shown in figure 1-38.



Status Bar Component Menu

Drag a component from the menu to Active Components to add it to your status bar.

Active Components

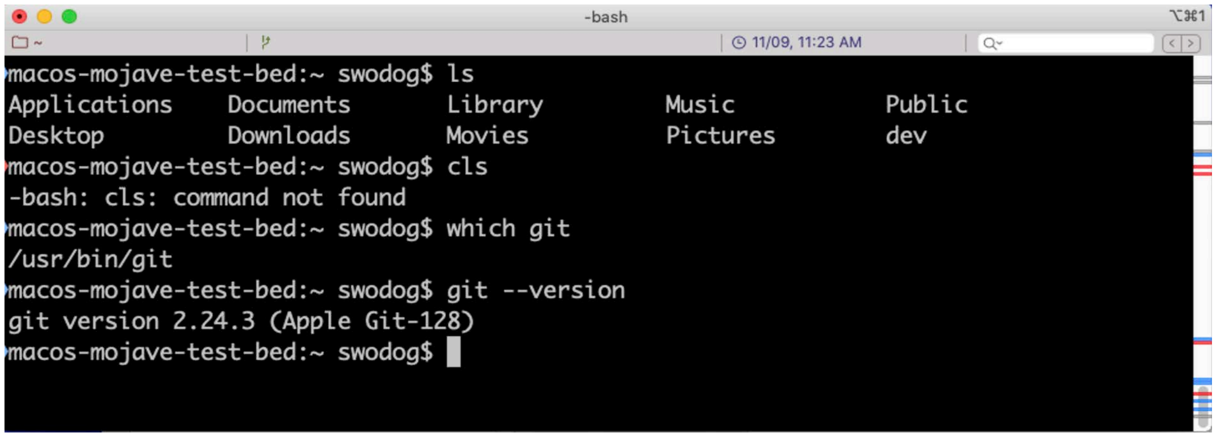
/Users/swodog | master | | 11/09, 10:44 AM | Search

Configure Component Auto-Rainbow: Automatic ? Advanced... Cancel OK

Figure 1-38: iTerm2 Status Bar Configuration Dialog

Referring to figure 1-38 — You'll enjoy playing around with status bar configuration. To add a component to the status bar simply drag it from the Component Menu panel to the Active Components panel. I like to add the *Current Directory*, *Git State*, *Current Date & Time*, and *Search* components. I add a space component in the middle. To set the component background color click the **Advanced** tab. If the iTerm2 terminal window is open you'll see the status bar components added in real time. I also like to set Auto-Rainbow to Automatic. When done, click **Configure Component** and click **OK** to return to the Profiles > Session tab.

If you used the components selected in figure 1-38 along with a light-gray component background your iTerm2 terminal should look like figure 1-39.



```
macos-mojave-test-bed:~ swodog$ ls
Applications    Documents       Library         Music           Public
Desktop         Downloads       Movies          Pictures        dev
macos-mojave-test-bed:~ swodog$ cls
-bash: cls: command not found
macos-mojave-test-bed:~ swodog$ which git
/usr/bin/git
macos-mojave-test-bed:~ swodog$ git --version
git version 2.24.3 (Apple Git-128)
macos-mojave-test-bed:~ swodog$
```

Figure 1-39: iTerm2 Terminal Window with Status Bar

Referring to figure 1-39 — You can see the status bar along the top of the terminal window. One feature of Shell Integration is *Marks*. You can barely make out a small colored triangle to the left of each shell prompt line. On the first line I entered the `ls` command. It executed with no errors so the mark on the left is blue. At the next prompt I entered the `cls` command, which clears the screen in a Windows command prompt but is not a native command in UNIX® or Linux so using it results in an error and a red triangle mark. To visit previous marks type the **Command** + **Shift** + **Up-Arrow** keys. To move forward through marks type the **Command** + **Shift** + **Down - Arrow** keys. To select the result of the last command type **Command** + **Shift** + **A**. That comes in pretty handy at times. Now, for fun, at the prompt type `it2attention fireworks`. If you saw fireworks explode across your screen then you can rest assured iTerm2 shell integration and utilities are properly installed.

2.6 Linux

A good terminal for Linux is *Terminator*: *https://terminator-gtk3.readthedocs.io/en/latest/*

2.6.1 Install And Configure Terminator

Open a Linux terminal and type `apt search terminator`. You should see it listed. To install Terminator type `apt install terminator`. When installation is complete close the terminal window, open Menu, and in the search bar start typing terminator. You'll see it appear below the Terminal application as shown in figure 1-40.

Referring to figure 1-40 — Right-click Terminator and select **Add to panel** and **Add to desktop**. Launch Terminator. You will notice right off the bat it appears different from the standard Terminal application that ships with Linux. Right-click in the body of the Terminator window and select **Preferences** to open the Terminator Preferences dialog as shown in figure 1-41.

Referring to figure 1-41 — I have the **Profiles** > **Colors** tab selected. You probably want to try out different color schemes. I like Ambience. I find the blue of the Linux color scheme too hard to read especially against a black background. If you are into retro tech, you can set green text on a black background, but that's pretty hardcore.

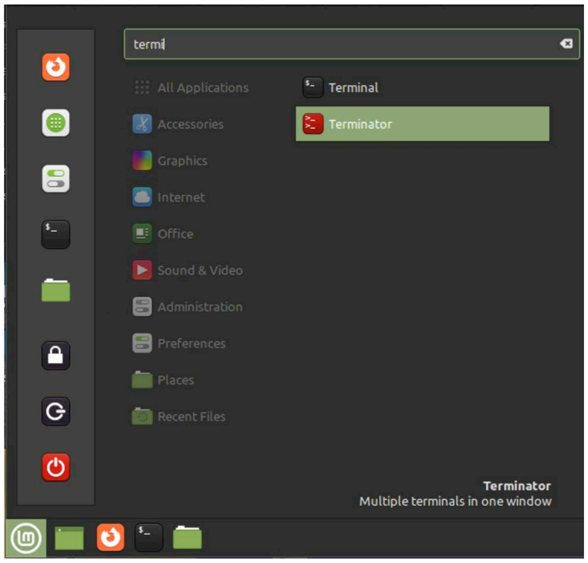
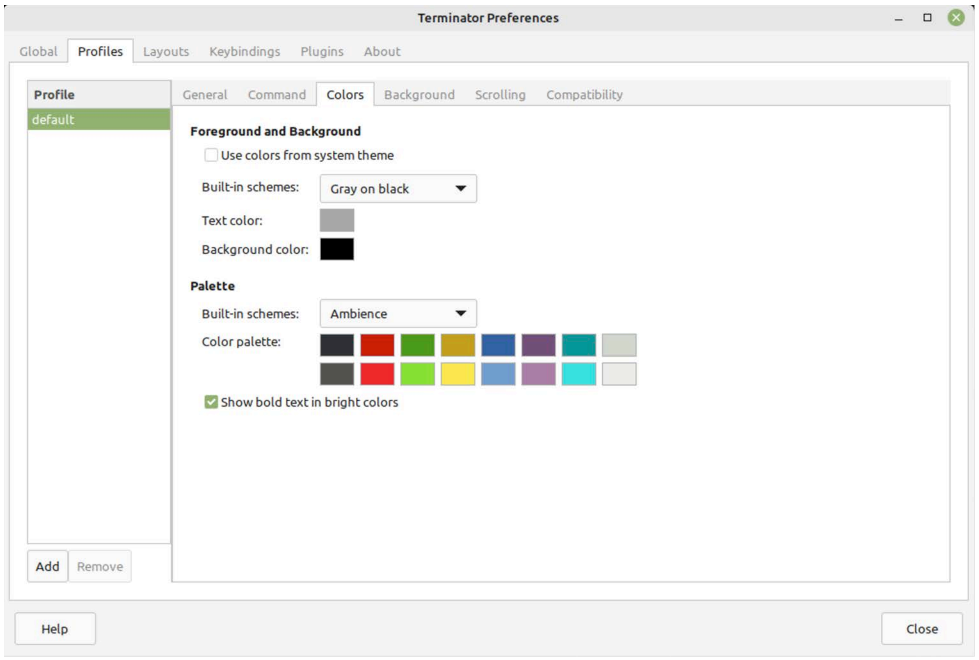


Figure 1-40: Locating Terminator Application



Foreground and Background

Use colors from system theme [ ]

Built-in schemes: Gray on black

Text color:

Background color:

Palette

Built-in schemes: Ambience

Color palette:

Show bold text in bright colors [✓]

Figure 1-41: Terminator Preferences Dialog on Colors Tab

I recommend you explore Terminator Preferences and see what's available. If you're new to terminals you may not understand everything you see nor the purpose of the setting, and that's OK. The best way to learn new concepts is in measured doses.

2.7 Parting Thoughts On Terminals And Shells

With the exception of Microsoft Windows, the default terminal applications that ship with macOS and Linux will work just fine for most of what you might need to accomplish, but the advanced features found in iTerm2 and Terminator will come in handy as you grow your terminal chops. Features like split screens and multiple sessions alleviate the need to have multiple terminal windows open and cluttering your desktop.

There are other terminal emulators besides iTerm2 and Terminator and developers have their favorites. In this chapter I presented a few popular options but you may have different tastes. Feel free to strike out on your own. The important thing is that you're able to use a terminal running the Bash shell to interact with your computer via text commands.

The Git Bash (mintty) terminal on Microsoft Windows is the lowest common denominator. All examples in this book that require Bash will work on the Git Bash terminal. Having said that, I will be favoring macOS and iTerm2 to show results of running Bash scripts and Python examples as that is my operating system and terminal of choice. I will highlight where necessary any differences between operating systems and terminal configurations.

As for shells there are others besides Bash and like terminal applications developers often display feverish emotions regarding their favorites. A few newer shells include the *Z Shell (zsh)* and the **fish** shell. The Z Shell (zsh) is the default shell on newer versions of macOS. I recommend setting the default shell to Bash, especially if you're new to Bash scripting, to prevent potential headaches and to ensure the scripts you find in this book fun as expected.

The version of Bash that ships with macOS is long in the tooth but works perfectly fine. Still, I'll be updating it to the latest version in the section on package managers below, just to show macOS users how it's done.

Quick Review

A terminal supports text based human/computer interaction. The purpose of a terminal is to present user input to the computer and render output from the computer. Most terminal applications are actually terminal emulators and if necessary can be used to connect to older computer systems that require specific terminal types.

A shell is a program launched by the terminal when you log in. The shell dictates what types of commands and scripts can be executed. The Bash shell is a popular shell, which is used throughout this book, but there are others.

Together, the terminal and the shell provide a command-line interface to the computer. The prompt is the line on which you enter shell commands and usually indicated by a dollar sign '`$`' or hash tag '`#`' character depending on whether you're logged in as a user or as root. Cursor position is indicated with an underscore '`_`' or box '█'character. You can customize the prompt via terminal preferences and shell profile settings.

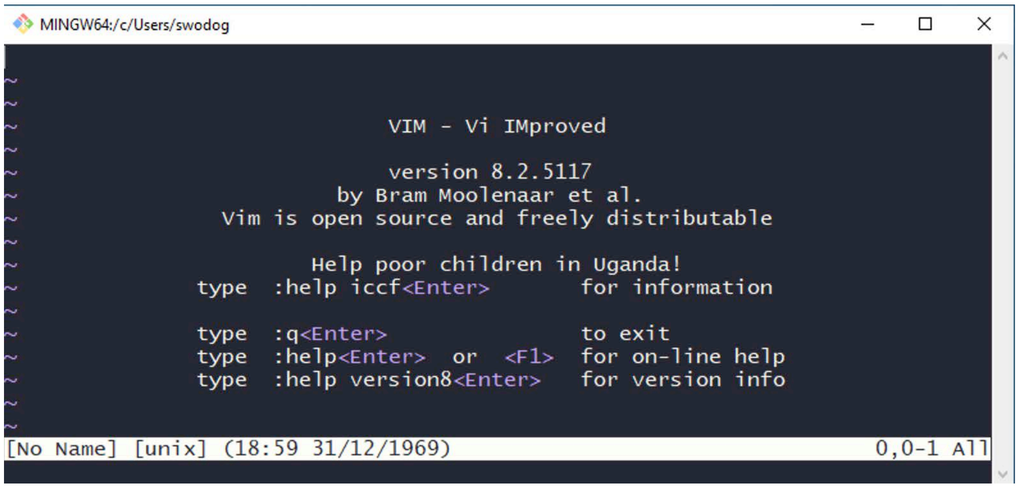
The Git Bash (mintty) terminal and its version of the Bash shell will serve as the lowest common denominator for all Bash script examples in this book.

3 Text Editors

A plain, ol' text editor is a must have tool for a software developer. You'll need a good text editor to edit code and scripts and to make long comments on Git commits. The text editors I recommend in this section run on all three platforms and in fact the first two editors I discuss, `vi` and `nano` come preinstalled on macOS and Linux and are included with Git for Windows.

3.1 Vi/Vim

Vi or Vim is an extremely powerful console-based text editor found on all modern UNIX and Linux distributions. It also comes with Git for Windows. Vim stands for **Vi IM**proved and is the default flavor of Vi shipped with modern UNIX-like operating systems with versions that run natively on Microsoft Windows as well. Figure 1-42 shows Vim running in the Git Bash terminal window.



MINGW64:/c/Users/swodog

```
~
~
~                              VIM - Vi IMproved
~
~                              version 8.2.5117
~                          by Bram Moolenaar et al.
~                 Vim is open source and freely distributable
~
~                        Help poor children in Uganda!
~               type  :help iccf<Enter>       for information
~
~               type  :q<Enter>               to exit
~               type  :help<Enter>  or  <F1>  for on-line help
~               type  :help version8<Enter>   for version info
~
~
[No Name] [unix] (18:59 31/12/1969)                              0,0-1 All
```

Figure 1-42: Vim Running in Git Bash Terminal on Windows

Referring to figure 1-42 — To launch Vim type `vi` or `vim` at the terminal prompt and hit enter. The `vi` command is linked to `vim`.

Vim comes straight out the box in a bare-bones configuration which is controlled almost exclusively via keyboard commands. Vim has a steep learning curve but learning Vim pays powerful dividends, especially if you work with remote virtual machines hosted in the cloud running Linux. Vim is a common denominator in that you can rest assured that some form of Vi or Vim will be available regardless of what flavor of UNIX or Linux greets you when you log in.

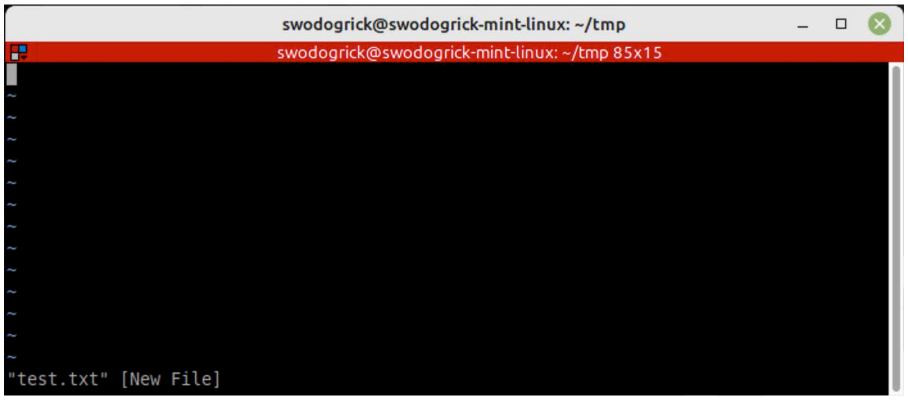
3.1.1 Basic Usage

Learning all the features and commands of Vim requires a fairly significant investment of time. It's not easy nor intuitive, but you can get stuff done in a pinch with just a handful of commands. For starters, you need to know the four modes of Vim.

3.1.1.1 Four Primary Vim Modes

Vim has four primary modes: *Normal*, *Insert*, *Command*, and *Visual*. First though, if you launched Vim earlier, type a colon '`:`' and the letter '`q`' to quit. Start by ensuring you're in your home directory, then create a directory named 'tmp' by typing the command `mkdir tmp` and hit return. Change to the `tmp` directory with the command `cd tmp`. Type the command `vi test.txt` to launch Vim and start editing a file named *test.txt*. Vim will open in the *Normal mode* as shown in figure 1-43.

Referring to figure 1-43 — Notice the Vim window appears different in Terminator running on Linux. This is because there is a different Vim configuration on the Linux machine. I'll show you how to customize Vim in short order.


swodogrick@swodogrick-mint-linux: ~/tmp
swodogrick@swodogrick-mint-linux: ~/tmp 85x15

```
~
~
~
~
~
~
~
~
~
~
~
~
~
"test.txt" [New File]
```

Figure 1-43: Vim in Normal Mode Editing Test.txt File in Terminator Window on Linux

3.1.1.1.1 Normal Mode

Vim launches in the *Normal mode*. You can perform basic text editing in Normal mode, but you can't type text directly. You can use the '**h**', '**j**', '**k**', and '**l**' keys to move *left*, *up*, *down*, and *right* respectively but this only works when you have text in which to move around. In Normal mode, Vim expects *control commands* until you enter the letter '`i`' to change to *Insert mode*.

3.1.1.1.2 Insert Mode

Type the character '`i`' to change to *Insert mode* and enter some text. At this point, you are typing text into a *buffer* or temporary memory area where edits are stored until you actually write them to the file. When you're done entering text, hit the ESC (escape) key to return to Normal mode.

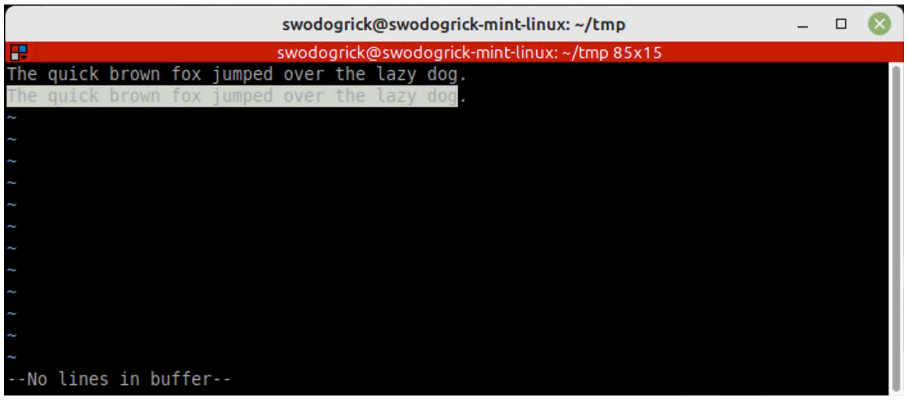
3.1.1.1.3 Command Mode

While in Normal mode type the colon '`:`' key to change to *Command mode*. To save the file, enter a colon '`:`' and '`w`' characters to *write* the changes made in the buffer to the file. If you are in Insert mode, you'd hit the ESC key to change to Normal mode followed by the '`:`' and '`w`' characters in succession.

3.1.1.1.4 Visual Mode

Visual mode is used to select sections of text for deleting or copy and paste operations. Visual mode has three submodes: *Visual*, *Visual Line*, and *Visual Block*. I'll demonstrate Visual Line mode here. In Normal mode place your cursor at the start of the line you want to copy and type **shift + V** to enter Visual Line mode. This selects an entire line as shown in figure 1-44.

Referring to figure 1-44 — To copy the selected text type '`y`' then type '`p`' to paste the selected text on a new line. To save the file and exit type ESC to enter Normal mode then '`:`', '`w`', '`q`'.



```
swodogrick@swodogrick-mint-linux: ~/tmp
swodogrick@swodogrick-mint-linux: ~/tmp 85x15
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
~
~
~
~
~
~
~
~
~
~
~
~
--No lines in buffer--
```

Figure 1-44: Text Selection in Visual Mode

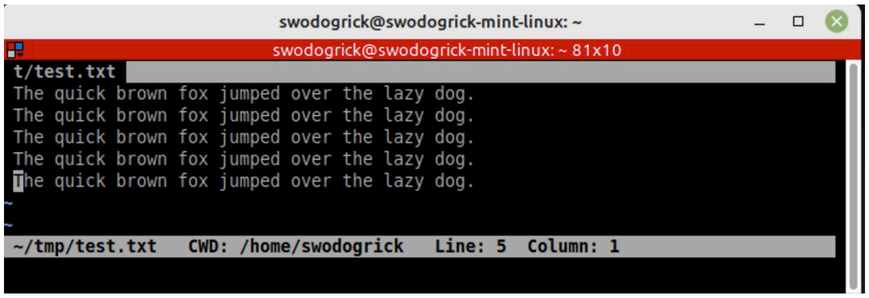
3.1.2 Customize Vim

Yeah, Vim is a total pain in the ass to learn but many hard core developers use Vim as their Integrated Development Environment (IDE) due to its numerous customization options. The quickest way to customize Vim is to obtain a ready-made Vim configuration file (.vimrc).

First, navigate to your home directory and open the existing *.vimrc* file with Vim. If you don't have a .vimrc file in your home directory, create it. Next, open a browser and navigate to *The Ultimate vimrc* repository on GitHub: *https://github.com/amix/vimrc*

OK, according to the repository's README.md there are two versions: 1. *Basic*, and 2. *Awesome*. I'll show you how to configure The Basic vim configuration and leave The Awesome configuration to you as an exercise since it requires cloning the repository, something that requires the use of Git, which I won't discuss until Part II.

Follow the link to the basic.vim file *https://github.com/amix/vimrc/blob/master/vimrcs/basic.vim* and at the top of the file click on the **Raw** button to switch to raw view. Select all and copy. Next, return to Vim and enter Insert mode. Right-click and paste the configuration settings into your .vimrc file. Save the file and exit Vim. Relaunch Vim and open the *~/tmp/test.txt* file. You should see a few changes to the Vim interface as shown in figure 1-45.

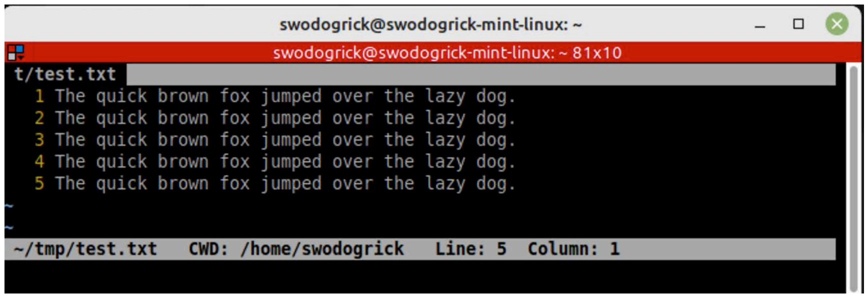


```
swodogrick@swodogrick-mint-linux: ~
swodogrick@swodogrick-mint-linux: ~ 81x10
t/test.txt
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
~
~
~/tmp/test.txt   CWD: /home/swodogrick   Line: 5  Column: 1
```

Figure 1-45: Vim After The Basic Configuration Code Added to .vimrc

Referring to figure 1-45 — The most visible changes are apparent in the gray status bar at the bottom of the Vim window that shows the path to the file you're editing, your current working directory (CWD), along with the cursor's current Line and Column number.

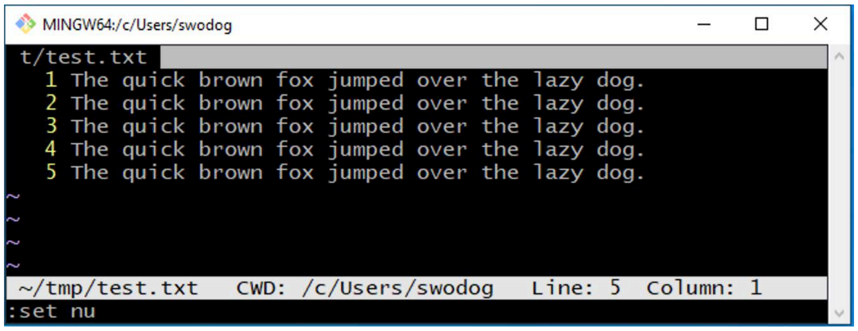
Another feature I like to activate in Vim is to show line numbers. Enter Command mode by typing a colon '`:`' and then type `set nu` and hit return. Your Vim lines will now begin with line numbers as shown in figure 1-46.



```
swodogrick@swodogrick-mint-linux: ~
swodogrick@swodogrick-mint-linux: ~ 81x10
t/test.txt
  1 The quick brown fox jumped over the lazy dog.
  2 The quick brown fox jumped over the lazy dog.
  3 The quick brown fox jumped over the lazy dog.
  4 The quick brown fox jumped over the lazy dog.
  5 The quick brown fox jumped over the lazy dog.
~
~
~/tmp/test.txt   CWD: /home/swodogrick   Line: 5  Column: 1
```

Figure 1-46: Vim with Line Numbers

Referring to figure 1-46 — If you always want to show line numbers simply edit your .vimrc file and add the `set nu` command. Note that The Basic vimrc configuration works on Vim running in the Git Bash terminal as well as shown in figure 1-47.



```
MINGW64:/c/Users/swodog
t/test.txt
  1 The quick brown fox jumped over the lazy dog.
  2 The quick brown fox jumped over the lazy dog.
  3 The quick brown fox jumped over the lazy dog.
  4 The quick brown fox jumped over the lazy dog.
  5 The quick brown fox jumped over the lazy dog.
~
~
~
~
~/tmp/test.txt   CWD: /c/Users/swodog   Line: 5  Column: 1
:set nu
```

Figure 1-47: Vim Running in Git Bash Terminal with The Basic vimrc Configuration

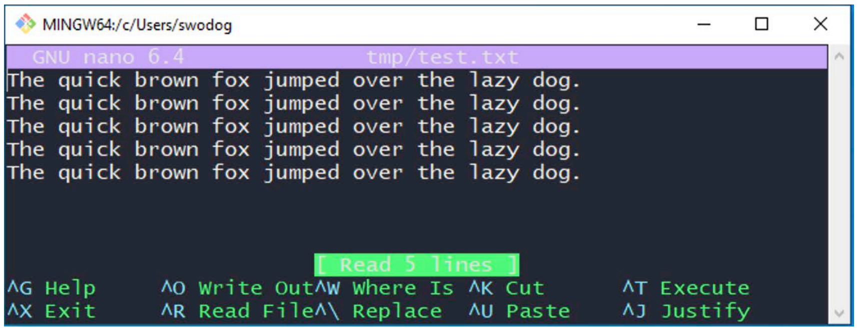
3.1.3 Parting Thoughts On Vi/Vim

Vim is a powerful, compact, speedy, ubiquitous, highly-configurable text editor. Knowing some basic Vim commands might save your hide and make you look like a hero someday when you've logged into a Linux EC2 instance in AWS or a Docker image and need to edit something in a pinch. If you're feeling overwhelmed by Vim and feel nauseated at the thought of using keyboard commands to navigate the text, you may prefer to use the next editor on the list.

3.2 Nano

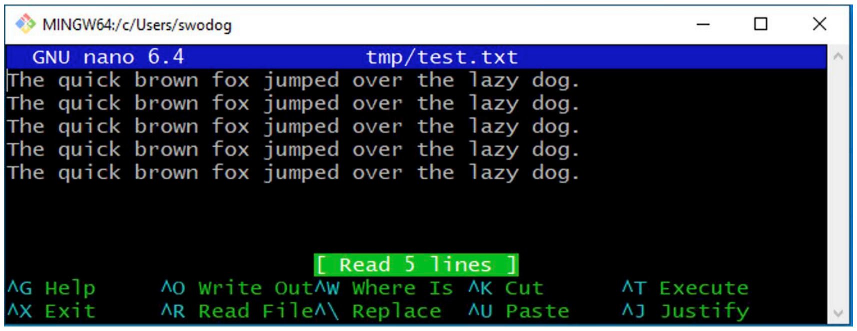
Like Vim, Nano is everywhere. Version numbers vary, but it comes on macOS, Linux, and Git for Windows. Like Vim there are some text commands involved but it's a whole lot easier to navigate and edit text. It's ready to go when you launch it, meaning it's ready for you to start entering text. Initially you're typing into an empty buffer but you can launch Nano with an existing file. To edit the *~/tmp/test.txt* file open a terminal window from your home directory and type `nano tmp/test.txt`. Figure 1-48 shows Nano running in a Git Bash terminal.

Referring to figure 1-48 — Nano is easy to use but I must have changed the terminal color scheme because I find the white text in the patch of neon green impossible to decipher even when looking at the screen close up in real life. To fix it, click in the upper left-hand corner of the Git Bash terminal and from the pop-up menu select **Options...** then select **Looks** and choose a more pleasing Theme from the dropdown menu. I selected the *mintty* theme and things look a whole lot easier to read as you can see from figure 1-49.



```
  GNU nano 6.4                   tmp/test.txt
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.



                          [ Read 5 lines ]
^G Help        ^O Write Out^W Where Is ^K Cut        ^T Execute
^X Exit        ^R Read File^\ Replace  ^U Paste      ^J Justify
```

Figure 1-48: Nano Running in Git Bash Terminal



```
  GNU nano 6.4                   tmp/test.txt
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.



                          [ Read 5 lines ]
^G Help        ^O Write Out^W Where Is ^K Cut        ^T Execute
^X Exit        ^R Read File^\ Replace  ^U Paste      ^J Justify
```

Figure 1-49: Oh, it says, "Read 5 Lines"!

Referring to figure 1-49 — I made this color change via terminal settings. You always have that option regardless of the application running in the terminal. However, Nano is configurable via a *.nanorc* file. To learn more about Nano editor configurations visit the GNU Nano website: *https://www.nano-editor.org/dist/v2.9/nanorc.5.html*

If you refuse to learn Vim then Nano is also found on modern UNIX and Linux operating systems. Again, the purpose of Nano is not to write novels and personally I don't use it much because for quick edits I prefer Vim, and for longer documents I use Visual Studio Code.

Quick Review

A software developer needs a good text editor. The editors discussed in this section, Vi/Vim and Nano work cross-platform. They work in a terminal and do not require a GUI or a mouse. They are ideal text editors for times when you must remote in to a UNIX-like virtual machine running in the cloud or in a Docker container.

Vim stands for **Vi IM**proved and ships along with Nano on macOS, Linux, and Git for Windows. Vim has a steep learning curve but is extremely powerful and highly configurable but it's not everyone's cup of tea. If the thought of using key-commands to navigate text makes you want to vomit then Nano is a good alternative to Vim.

Don't make the common novice mistake of editing text files with a word processor application as you may end up adding hidden codes to the file which will dork everything up.

4 Visual Studio Code

As powerful as text editors have become you'll still want a full-blown Integrated Development Environment (IDE) for software development. An IDE, as the word *integrated* suggests, offers source code editing, project management, debugging, and other features in one convenient package.

Some hardcore developers use Vim as their IDE but doing so requires a lot of configuration not to mention a steep learning curve. It probably gives them solid street creds to say that's the way they roll but if you're new to the profession then you're gonna want something easier to use with a ton of features and can be customized into exactly what you need with a small handful of extensions. You're going to love *Visual Studio Code*.

I've chosen Visual Studio Code because it runs on all three platforms: macOS, Linux, and Microsoft Windows and is easy to master.

4.1 Install Visual Studio Code

Installing Visual Studio Code on macOS and Windows is straightforward so I'll only review the Linux installation steps. Navigate to the Visual Studio Code website *https://code.visualstudio.com* and download the package that corresponds to your version of Linux. For Linux Mint download the *.deb* **Stable** package. When the download completes, open a terminal, change to your *~/Downloads* directory, and run the command `sudo dpkg -i` *package_name*`.deb` where *package_name* is the name of the package you want to install. Hit return. Enter your password when prompted and hit return to begin the installation.

When installation completes, open Menu and search for Visual Studio Code. Add it to the panel, the desktop, and to your favorites if so desired. You can delete the package file and clear your *~/Downloads* folder.

When you're ready, launch Visual Studio Code. It'll ask you to make some initial configuration decisions like choosing a color theme (I like the Dark theme...). Choose a color theme and click **Next Section** until you see **Mark Done**. Click **Mark Done**. Your Visual Studio Code window will open to the Get Started window as shown in figure 1-50.

Referring to figure 1-50 — Along the left-hand side of the VS Code window you'll find a series of icons. From the top you have *Explorer*, *Search*, *Source Control*, *Run and Debug*, and *Extensions*. Moving down towards the bottom you have *Accounts*, and *Manage*.

As you progress through this book you'll learn more about VS Code. The first thing you'll need to add are several extensions that make it easier to work with Python. Let's do that now.

4.2 Add Extensions For Python Development

Click the Extensions icon — it's 5th from the top and looks like a Tetris game — to open the EXTENSIONS panel. You'll see a search bar at the top along with a funnel icon (filter) and a circular arrow (refresh) as shown in figure 1-51.

Referring to figure 1-51 — A word on selecting extensions. I personally gravitate towards the ones with a blue certification badge from Microsoft. You see one at the top of the list named Python. The Python extension will add Python IntelliSense which is super helpful. To install the extension just click on the blue **Install** button. Go ahead and install the Python extension. Note

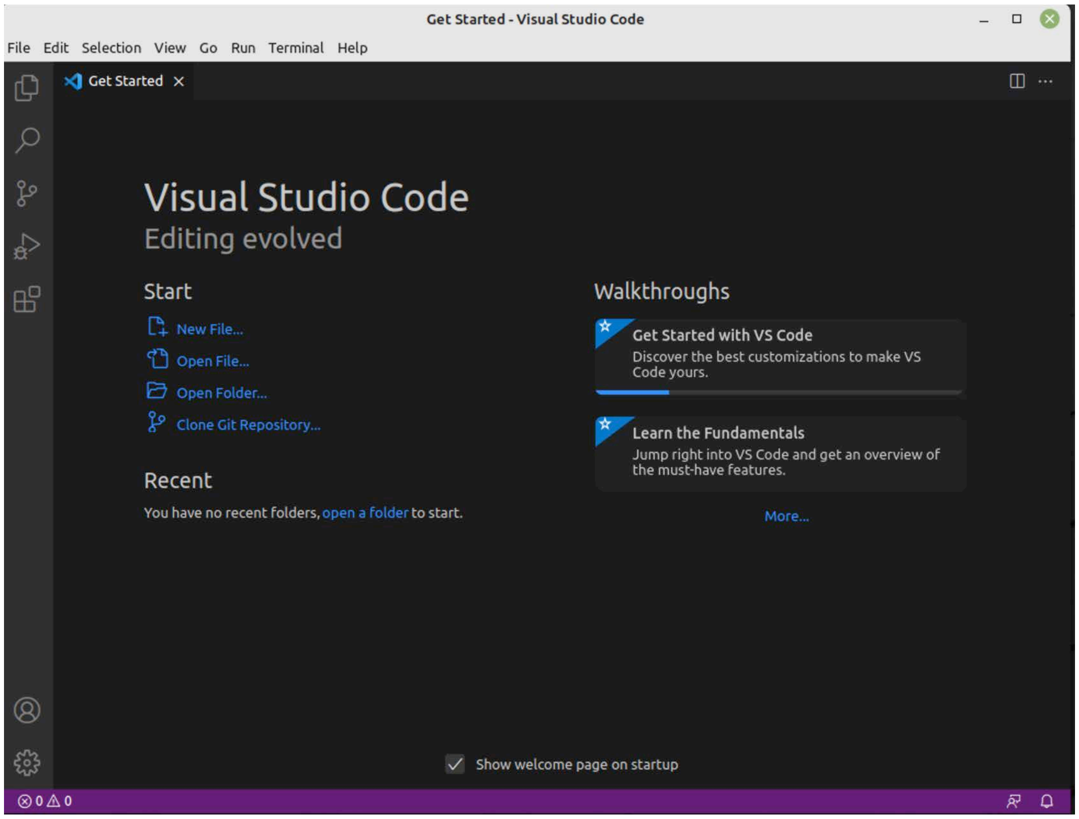

Get Started - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Recent
You have no recent folders, open a folder to start.
Walkthroughs
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.
More...
Show welcome page on startup
0 0

Figure 1-50: Visual Studio Code Running on Linux Mint

that installing one extension may install others. Such is the case with the Python extension, which also installs the Pylance and Jupyter notebook extensions. OK, that's all you need for now. VS Code has a ton of built-in features. If you need another extension I'll let you know when the time comes. Take time now to browse the extensions to see what's available.

4.3 Launch Visual Studio Code From The Command-Line

Visual Studio Code can be launched from the command-line. Simply type the command `code` followed optionally by the file or folder you want to open.

For example, to open the *~/tmp* folder as a project open a terminal, navigate to your home directory if not already there, and type `code tmp`. If you created the *~/tmp* folder earlier along with the `test.txt` file you'll see both in the Explorer panel. VS Code may ask you if you want to trust the authors of the code found in the folder. Answer in the affirmative and proceed.

Launching VS Code from the command line only works automatically after installing on Linux with the `dpkg` command. You'll need to add the location of the `code` executable to your PATH environment variable for this to work in the macOS and Git Bash terminals. I'll show you how to configure environment variables later in this chapter.

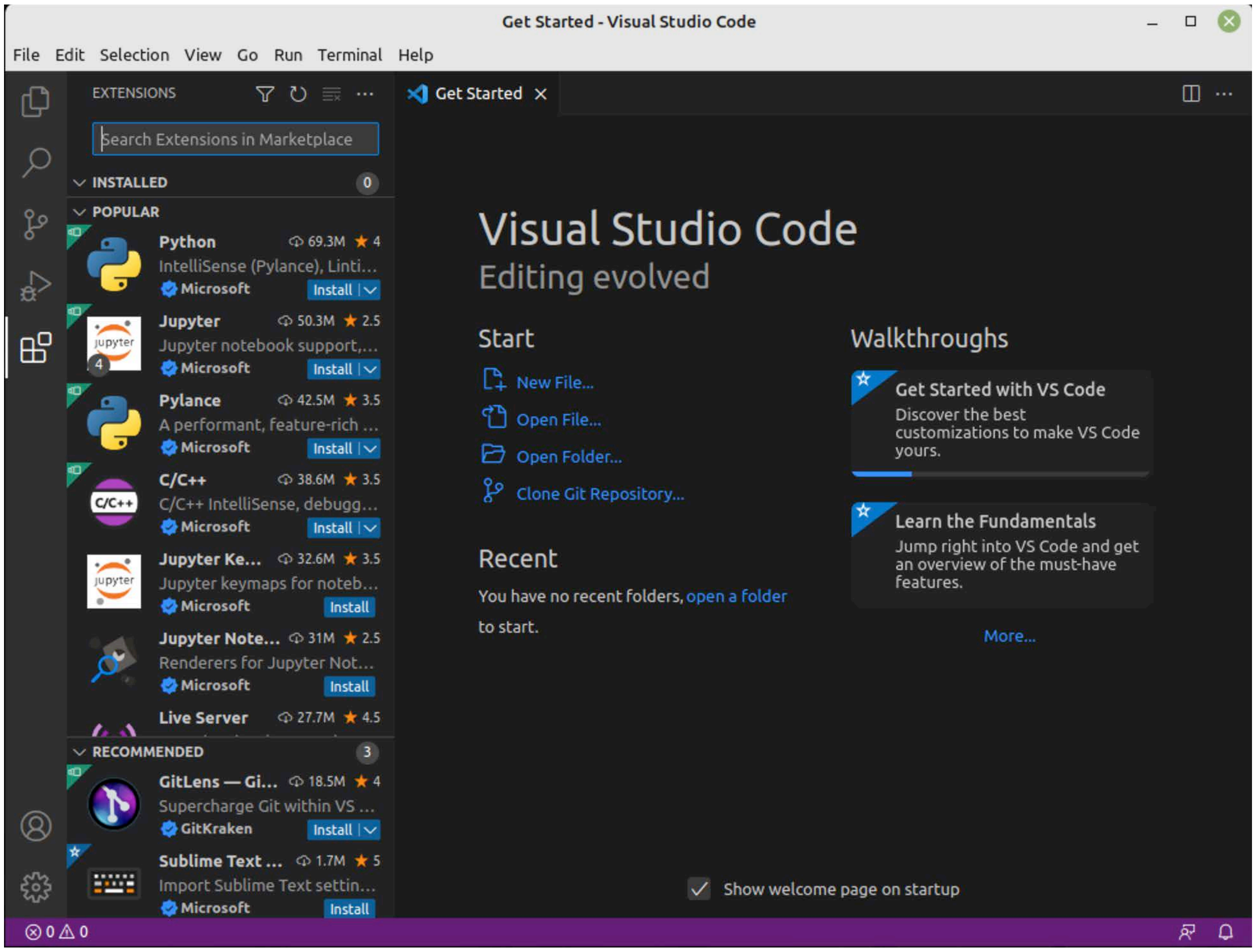

Get Started - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXTENSIONS
Search Extensions in Marketplace
INSTALLED 0
POPULAR
Python 69.3M ★ 4
IntelliSense (Pylance), Linti...
Microsoft Install
Jupyter 50.3M ★ 2.5
Jupyter notebook support,...
Microsoft Install
Pylance 42.5M ★ 3.5
A performant, feature-rich ...
Microsoft Install
C/C++ 38.6M ★ 3.5
C/C++ IntelliSense, debugg...
Microsoft Install
Jupyter Ke... 32.6M ★ 3.5
Jupyter keymaps for noteb...
Microsoft Install
Jupyter Note... 31M ★ 2.5
Renderers for Jupyter Not...
Microsoft Install
Live Server 27.7M ★ 4.5
RECOMMENDED 3
GitLens — Gi... 18.5M ★ 4
Supercharge Git within VS ...
GitKraken Install
Sublime Text ... 1.7M ★ 5
Import Sublime Text settin...
Microsoft Install
Get Started
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Recent
You have no recent folders, open a folder to start.
Walkthroughs
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.
More...
Show welcome page on startup
0 0

Figure 1-51: VS Code Extensions Panel

Quick Review

Visual Studio Code is a powerful Integrated Development Environment (IDE) that runs on Microsoft Windows, macOS, and Linux. It comes out of the box with a ton of helpful features and anything not already built in can be added via extensions.

Visual Studio Code can be launched from the command-line by typing the command `code` followed optionally by the name of a folder or file. This works automatically when installing on Linux Mint with the dpkg command. You'll need to configure the PATH environment variable on Windows and macOS to launch VS Code from the command line.

5 Package Managers

A package manager is an application used to manage software on a computer system as shown in figure 1-52.

Referring to figure 1-52 — A package manager maintains a list of available software packages along with any dependencies required for installation. When you install a package, the package manager compares the software installed on your computer with package metadata and

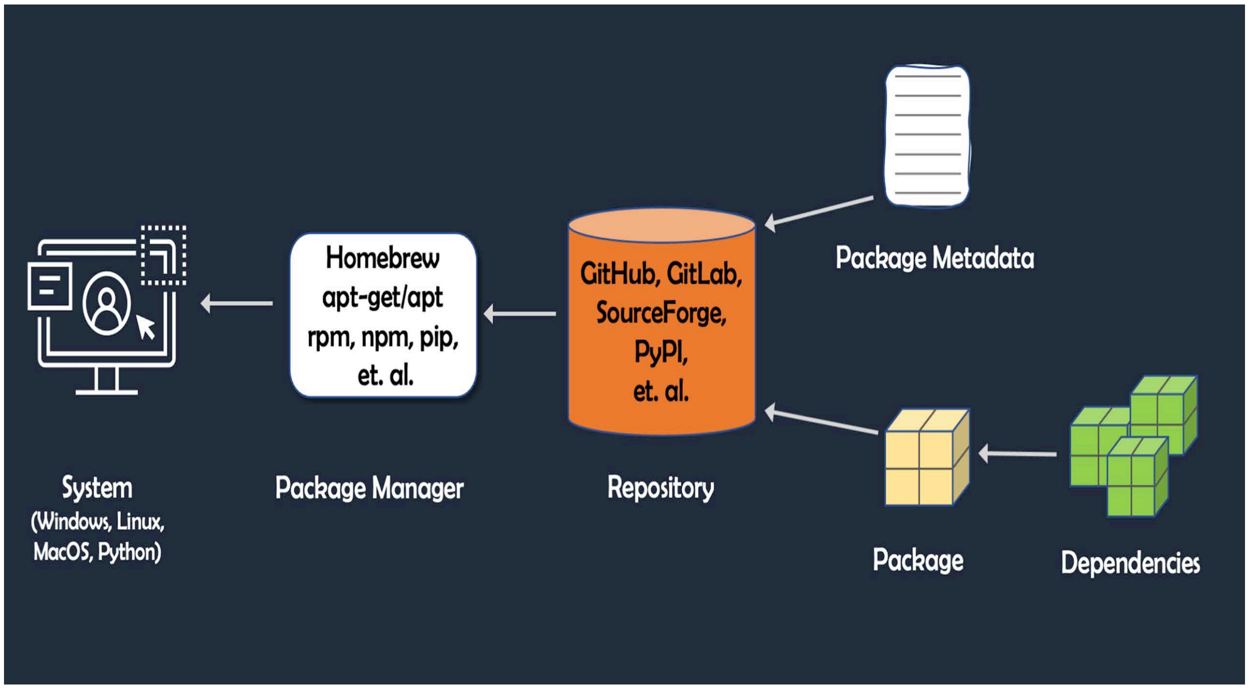


Figure 1-52: Package Manager Architecture

installs or updates related packages required to support the new software. The package manager you'll use depends on your operating system.

Windows and macOS users are rather spoiled in that you generally don't use a package manager to install software. Instead, you usually visit a website that provides a download link to the software you want to install and then install the software manually with some sort of installer. Windows users are generally provided either an installer executable (.exe) or (.msi) file.

For macOS users, most software can be installed via the Apple *App Store* or directly from websites via (.dmg) images, which are automatically unpacked and mounted on the desktop upon download. The new application must then be dragged to the Applications folder. Often times an installer will guide you through the installation process. Sometimes you get a choice of where to install the software and sometimes you don't.

Linux users have different ways of installing software on their machines. There are different package managers for different distributions. For example, *apt* is the package manager for Debian and derivatives (.deb packages) whereas CentOS and RedHat use *rpm* or *yum* (.rpm packages). These are command-line tools but Linux also ships with GUI-based installers.

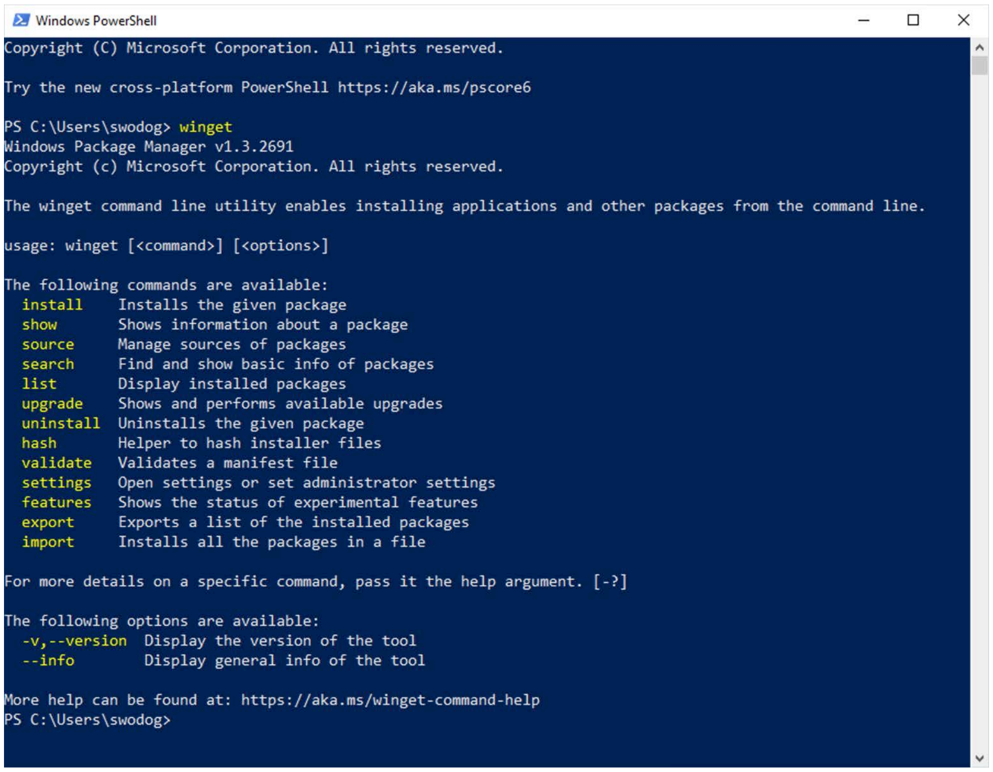
Windows 10/11 users have access to a package manager called *winget*, while macOS users can install software via a third-party package manager called *Homebrew*. These package managers are especially useful for software developers who work daily with command-line tools.

5.1 Windows

If you're a windows user, you will most likely install software directly on your machine by visiting the software maker's website, find and download the installer for your version of Windows (32-bit, 64-bit, x86, arm64, etc.), and install the software. You also have access to a command-line package manager named *winget*.

5.1.1 WINGET

To use *winget* launch a PowerShell terminal and type `winget` at the prompt to display usage and a list of commands as shown in figure 1-53.

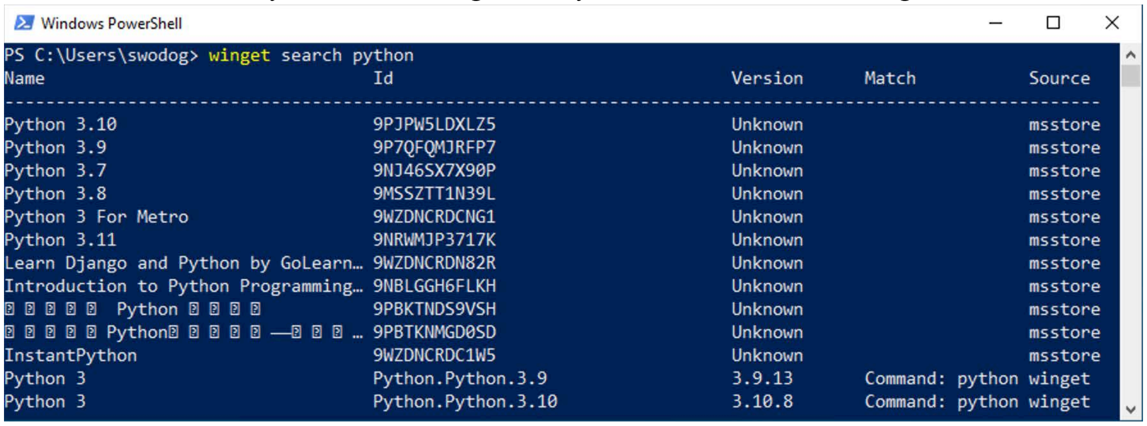


```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\swodog> winget
Windows Package Manager v1.3.2691
Copyright (c) Microsoft Corporation. All rights reserved.

The winget command line utility enables installing applications and other packages from the command line.

usage: winget [<command>] [<options>]

The following commands are available:
  install    Installs the given package
  show       Shows information about a package
  source     Manage sources of packages
  search     Find and show basic info of packages
  list       Display installed packages
  upgrade    Shows and performs available upgrades
  uninstall  Uninstalls the given package
  hash       Helper to hash installer files
  validate   Validates a manifest file
  settings   Open settings or set administrator settings
  features   Shows the status of experimental features
  export     Exports a list of the installed packages
  import     Installs all the packages in a file

For more details on a specific command, pass it the help argument. [-?]

The following options are available:
  -v,--version  Display the version of the tool
  --info        Display general info of the tool

More help can be found at: https://aka.ms/winget-command-help
PS C:\Users\swodog>
```

Figure 1-53: winget Package Manager in Windows PowerShell

Referring to figure 1-53 — Type `winget list` to view installed packages. When asked, *Do you agree to all the source agreements terms?* enter **Y** and hit return. You'll see a slew of installed packages scroll up the terminal. Browse the list. It's interesting to see what's already installed.

Now, search for Python packages by typing `winget search python`. Scroll up to the top of the list. You should see Python 3.10 along with Python 3.11 as shown in figure 1-54.



```
PS C:\Users\swodog> winget search python
Name                                 Id                                   Version      Match            Source
-------------------------------------------------------------------------------------------------------------
Python 3.10                          9PJPW5LDXLZ5                         Unknown                       msstore
Python 3.9                           9P7QFQMJRFP7                         Unknown                       msstore
Python 3.7                           9NJ46SX7X90P                         Unknown                       msstore
Python 3.8                           9MSSZTT1N39L                         Unknown                       msstore
Python 3 For Metro                   9WZDNCRDCNG1                         Unknown                       msstore
Python 3.11                          9NRWMJP3717K                         Unknown                       msstore
Learn Django and Python by GoLearn…  9WZDNCRDN82R                         Unknown                       msstore
Introduction to Python Programming…  9NBLGGH6FLKH                         Unknown                       msstore
⍰ ⍰ ⍰ ⍰ ⍰  Python ⍰ ⍰ ⍰ ⍰             9PBKTNDS9VSH                         Unknown                       msstore
⍰ ⍰ ⍰ ⍰ ⍰ Python⍰ ⍰ ⍰ ⍰ ⍰ —⍰ ⍰ ⍰ …    9PBTKNMGD0SD                         Unknown                       msstore
InstantPython                        9WZDNCRDC1W5                         Unknown                       msstore
Python 3                             Python.Python.3.9                    3.9.13       Command: python  winget
Python 3                             Python.Python.3.10                   3.10.8       Command: python  winget
```

Figure 1-54: Searching for Python Packages with winget

Referring to figure 1-54 — There sure are a lot of interesting Python packages. To get more information about the Python 3.11 package type `winget show --name Python 3.11` to list detailed package information. I'll show you how to install and configure Python later in this chapter but if you want to install it now with `winget` please proceed. You'll need to use the following command:

```
winget install --name "Python 3.11" ‘
--version "Python.Python.3.11" ‘
--source "winget"
```

You can enter this command all on one line but if you do you'll need to omit the trailing backtick characters from the first two lines. The trailing backtick character is used in PowerShell to break long commands across multiple lines. If all goes well (*The battle cry of the software engineer!*) you'll see a *Downloading* message. When the download completes, the Python 3.11.0 (64bit) Setup dialog window will launch. Follow the installation prompts. Meet you in the next section. (**NOTE:** *Newer versions of Python may be available.*)

5.2 MacOS

MacOS users are even more spoiled than their Microsoft Windows colleagues. One generally installs software from Apple's App Store. The App Store usually has the latest and greatest versions of whatever software you're looking for but not all software you need is located on the App Store and often times you will download and install an application directly from its website.

For many software developer tools it's best to install them using a package manager but macOS does not ship with a package manager, however, there is a excellent third-party package manager available for macOS called *Homebrew*.

5.2.1 Homebrew (a. k. a. brew)

Homebrew, or simply *brew*, is, according to its website, "The missing package manager for macOS (or Linux)". To install Homebrew, navigate on over to their website *https://brew.sh* and copy the install script off the homepage. (*Click the clipboard icon to the right of the install script's text box.*)

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
HEAD/install.sh)"
```

Paste the installation script into a terminal and hit return. When prompted enter your password. Installation may take a few minutes as the Homebrew repository is cloned to your local machine. When installation completes, you'll see three commands listed under a section called Next Steps. Copy and paste, and execute each of those commands, one at a time, in your terminal to add the brew command to your system path. Close the terminal window and launch a new terminal. When you've finished, test it out by installing a helpful command: `tree`. To install the `tree` command using brew type the following at the prompt: `brew install tree`

When installation completes, navigate to your home directory if not already there and type the command `tree`. You should see a tree view of your home directory scroll up the terminal. The iTerm2 terminal may ask permission to access some of your home directory's folders. Click Yes or No depending on your level of comfort. I generally click Yes to everything.

The raw tree view is useless in my opinion for large directory listings and best limited by the `-L` (level) argument. Retry the command like so: `tree -L 2` or target a specific folder as in: `tree Documents` or `tree Downloads`.

The `tree` command comes in super handy when you want to see a graphical layout of your software development project's directory structure. Later in this chapter you'll use `brew` to install Python.

5.3 Linux Mint (Debian/Ubuntu)

As you've already learned in this section, if you're a Linux user, the package manager you'll be using is dictated by the distribution of Linux. In this book I've chosen to focus on Linux Mint, which is a derivative of Debian and Ubuntu. The package manager is `apt`.

5.3.1 APT

To get help on apt simply type `apt` at a terminal prompt. To search for a package type `apt search package_name`, where `package_name` can be a full or partial package name. Type `apt search python` and see how many packages related to Python scroll by.

To install Python 3.11 type `apt install python3.11:any`. When prompted enter your password. When installation completes, close and relaunch your terminal and type `python3` to launch the Python interpreter or **R**ead **E**val **P**rint **L**oop (REPL). I'll talk more about the Python REPL later in the book.

Quick Review

Use a package manager when possible to install and manage software development tools and related applications. Microsoft Windows 10 and 11 ship with *winget*, Linux Mint uses *apt*, and macOS can use Homebrew (*brew*).

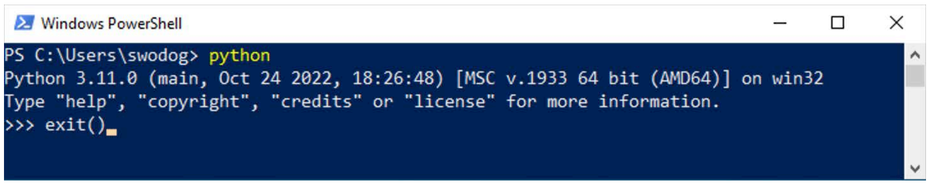
6 Install, Configure, and Run Python

In this section you'll install and configure Python so you can run it from the command line.

6.1 Windows

You can install Python several ways on Windows 10 & 11. You can either use the `winget` package manager discussed in the previous section or visit the Python website, *https://python.org* download the installer and install it manually.

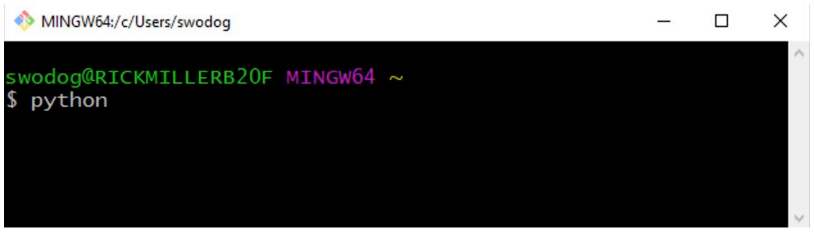
If you installed Python using `winget` you should be able to run it in a PowerShell terminal. Try that now. Launch a PowerShell terminal and type `python` at the command prompt. That should launch the python interpreter or REPL as shown in figure 1-55.



```
PS C:\Users\swodog> python
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
```

Figure 1-55: Python REPL Running in PowerShell

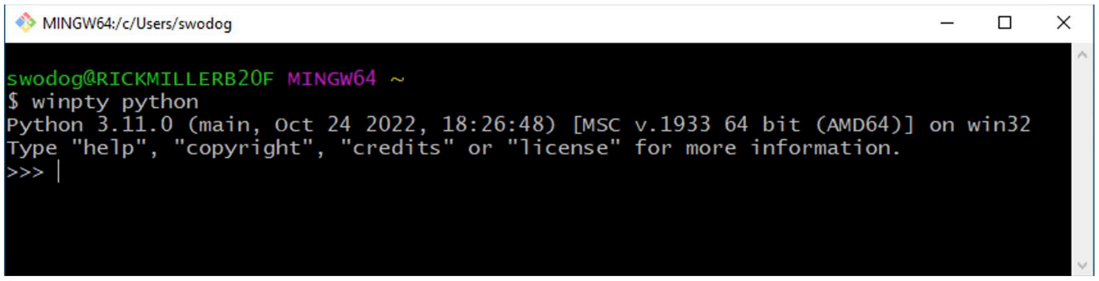
Referring to figure 1-55 — That means the environment variables have been set properly so the operating system can find the Python 3.11 executable. Type `exit()` to exit the REPL. Next, launch the Git Bash terminal and type python. It may appear as though something is happening but the terminal hangs as is shown in figure 1-56.



```
swodog@RICKMILLERB20F MINGW64 ~
$ python
```

Figure 1-56: Python Not Working Properly in Git Bash Terminal

Referring to figure 1-56 — If you encounter this issue, close the terminal and click OK on the warning dialog pop-up. Launch the Git Bash terminal again but this time type `winpty python` as shown in figure 1-57.



```
swodog@RICKMILLERB20F MINGW64 ~
$ winpty python
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Figure 1-57: Python Running Property by Typing winpty python

Referring to figure 1-57 — The `winpty` command comes with Git for Windows. It enables software written to interact with a Microsoft Windows console to work in the Git Bash (mintty) terminal. You can learn more about the details of the `winpty` command here: *https://github.com/rprichard/winpty* All you really need to know is that if you run software in the Git Bash terminal and expect to see output but see none, try running the command with `winpty`. I will show you later how you can create a command alias in your `.bash_profile` that automatically runs such commands with `winpty`.

6.2 MACOS

You can install Python either by visiting the official Python website, *https://python.org*, downloading the installer and installing manually, or via `brew`. I prefer installing Python using `brew`. First, use `brew` to search for Python 3 packages with the following command:

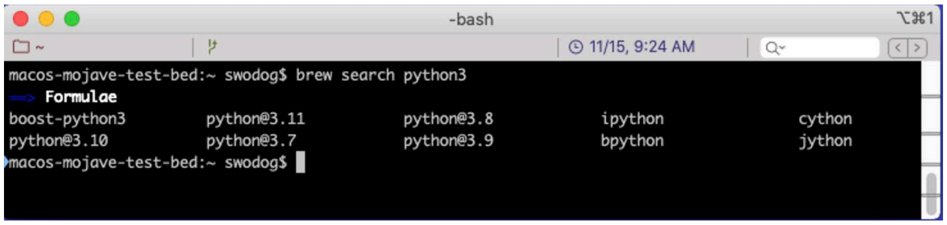
```
brew search python3
```

This will return a list similar to the one shown in figure 1-58.

Referring to figure 1-58 — Note the name of the package you want to install. To install Python 3.11 use the following command:

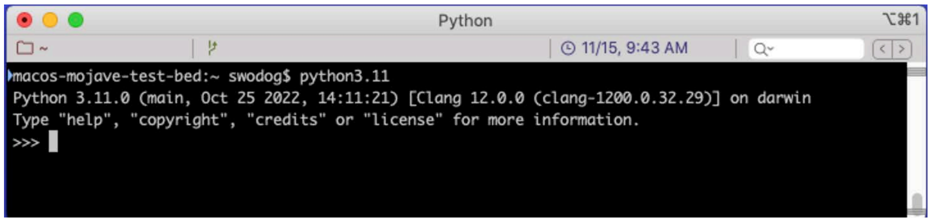
```
brew install python@3.11
```

When installation completes run Python 3.11 with the following command: `python3.11` You should see the Python 3.11 REPL run as shown in figure 1-59.



```
macos-mojave-test-bed:~ swodog$ brew search python3
==> Formulae
boost-python3       python@3.11         python@3.8          ipython             cython
python@3.10         python@3.7          python@3.9          bpython             jython
macos-mojave-test-bed:~ swodog$
```

Figure 1-58: Searching for python3 Packages with brew



```
macos-mojave-test-bed:~ swodog$ python3.11
Python 3.11.0 (main, Oct 25 2022, 14:11:21) [Clang 12.0.0 (clang-1200.0.32.29)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 1-59: Python 3.11 Running on macOS

Referring to figure 1-59 — If you followed the instructions exactly then everything may seem fine and dandy but if you typed the command `python` or `python3` you will see very different versions.

6.2.1 Python vs. Python3

MacOS ships with a version of both Python 2 and Python 3. If you simply type the command `python` it runs Python 2.7.16. If you type `python3` it runs whatever version of Python 3 shipped with the operating system. For macOS Catalina up to Monterey if was most likely Python 3.8. To install the latest stable version of python3 with brew use the following command:

```
brew install python3
```

Now when you run `python3` you'll actually be running Python 3.10.8, which at the time I write this is the latest stable version installed by `brew`. This may change in the very near future. You shouldn't need anything greater than Python 3.10 for this book anyway. If you need Python 3.11, you can always run it with the `python3.11` command.

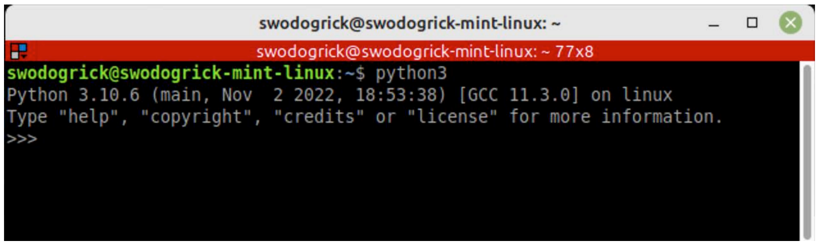
6.3 Linux

To install Python 3.11 on Linux Mint using the `apt` package manager use the following command:

```
apt install python3.11:any
```

When prompted enter your password. When installation completes close and relaunch your terminal window. Type `python3` to launch the Python REPL as shown in figure 1-60.

Referring to figure 1-60 — If you're able to run the Python 3.11 REPL then you're all set. To exit the REPL type `exit()`.


swodogrick@swodogrick-mint-linux: ~
swodogrick@swodogrick-mint-linux: ~ 77x8

```
swodogrick@swodogrick-mint-linux:~$ python3
Python 3.10.6 (main, Nov  2 2022, 18:53:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 1-60: Python 3.11 REPL Running in Terminator on Linux Mint

Quick Review

Unless you have a compelling reason otherwise you should always install the latest stable version of Python on your system. Windows, macOS, and Linux users have options regarding how to install Python. You can either download the installer directly from the Python.org website or use a package manager. You should be able to run the latest version of Python 3.x with the command `python3`.

Newer versions of macOS ship with Python 2 and 3 already installed. To execute Python 2.x use the `python` command. You won't need Python 2 for this book but you do need to be aware it's installed. To run Python 3.x use the `python3` command. To install the latest stable version of Python 3.x on macOS use the command `brew install python3`. If you install Python 3.11 with `brew` you can run it by using the `python3.11` command. To run Python 3.11 on Linux Mint use the command `python3`. **NOTE:** *All examples in this book will run on Python 3.10 and above.*

7 Configure Environment Variables And Shell Profiles

There's one last critical topic to discuss regarding development environment configuration before calling this chapter complete and that's how to configure environment variables and shell profiles.

Regardless of what operating system you'll be using, you'll be running most of the examples, especially those involving Bash scripts, in a Bash shell. This includes Microsoft Windows users who will be using the Git Bash (mintty) terminal.

To further customize your terminal experience you'll be adding commands to a file in your home directory called *.bash_profile*. Linux and macOS users can export shell environment variables via their .bashrc or .bash_profile files, but Windows users will still need to set environment variables the Windows way.

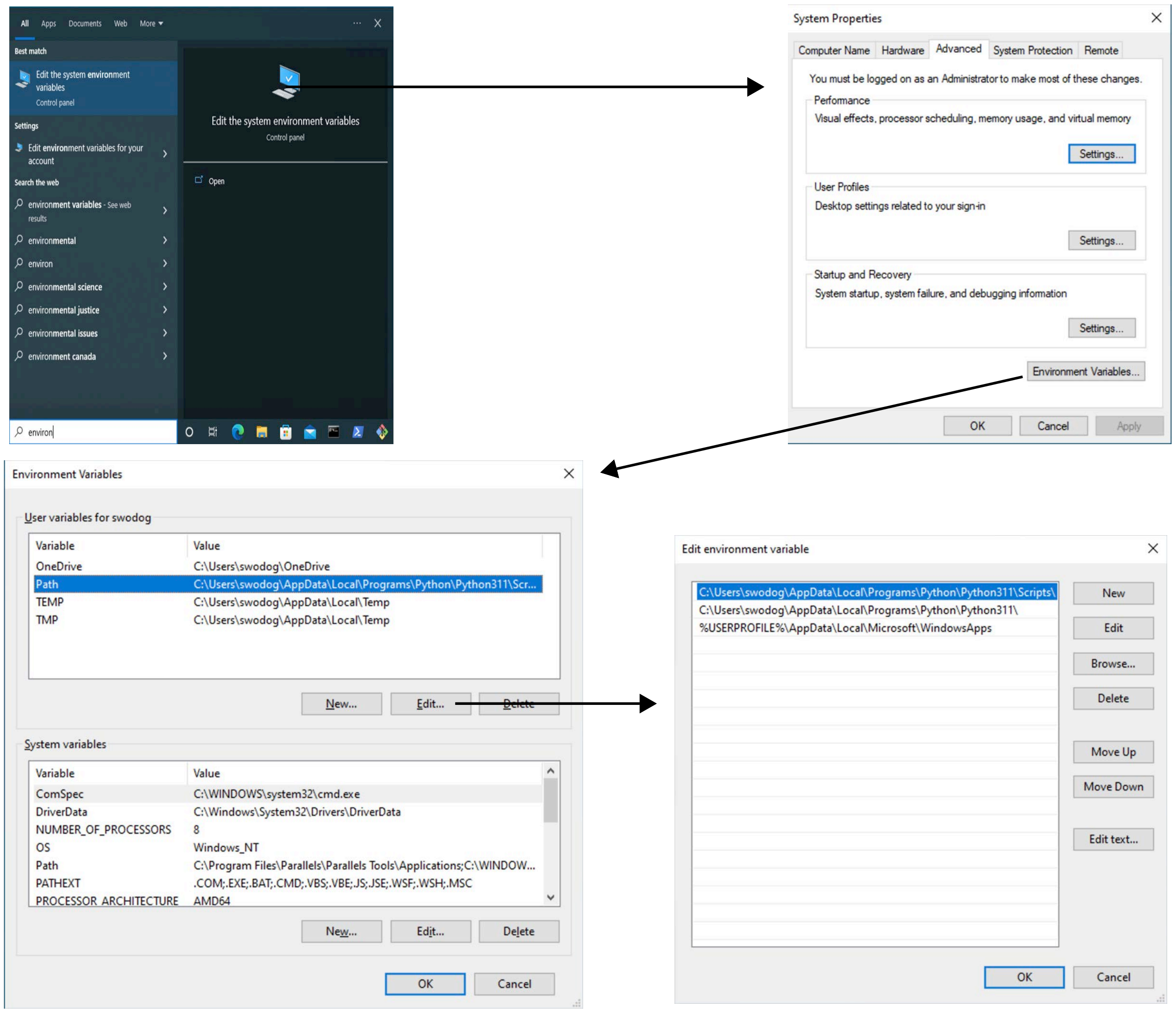
7.1 What Is An Environment Variable?

An environment variable is a named memory location used by the operating system to store a configuration string or value. One ubiquitous environment variable found on all operating systems is the PATH variable, which tells the OS where to search for executable files. There are generally two sets of environment variables: *System environment variables* and *User environment variables*. System environment variables apply to the operating system as a whole and to every user who logs into the system. User environment variables apply only to a particular user when they are logged in.

If you open a terminal, type a command, and get a *Command not found* error, then the operating system is saying either the application or tool is not installed or it does not know where to find it, or in other words, the path to its executable is not listed in the PATH environment variable.

7.2 Windows

Some shell settings can be configured in a .bash_profile but environment variables should be set using the Environment Variables dialog. The easiest way to open the environment variables dialog is to search for it in the taskbar search box as shown in figure 1-61.



System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

User Profiles
Desktop settings related to your sign-in
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK Cancel Apply

Environment Variables

User variables for swodog

Variable	Value
OneDrive	C:\Users\swodog\OneDrive
Path	C:\Users\swodog\AppData\Local\Programs\Python\Python311\Scr...
TEMP	C:\Users\swodog\AppData\Local\Temp
TMP	C:\Users\swodog\AppData\Local\Temp

New... Edit... Delete

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files\Parallels\Parallels Tools\Applications;C:\WINDOW...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

New... Edit... Delete

OK Cancel

Edit environment variable

C:\Users\swodog\AppData\Local\Programs\Python\Python311\Scripts\
C:\Users\swodog\AppData\Local\Programs\Python\Python311\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps

New Edit Browse... Delete Move Up Move Down Edit text...

OK Cancel

Figure 1-61: Setting/Editing Environment Variables in Microsoft Windows

Referring to figure 1-61 — From the search results click "**Edit the system environment variables**" to open the *System Settings* dialog. Click the **Environment Variables** button to open the *Environment Variables* dialog. (*Why Microsoft Windows doesn't directly open that dialog is beyond me.*) Click the New... button to add a new variable or select an existing variable and click the Edit... button. Take the time now to explore what User variables have been set. You may see more or less variables listed depending on how many applications are installed on your machine. As a rule you generally **do not dork around with the System variables**. Add or edit the User

0000001

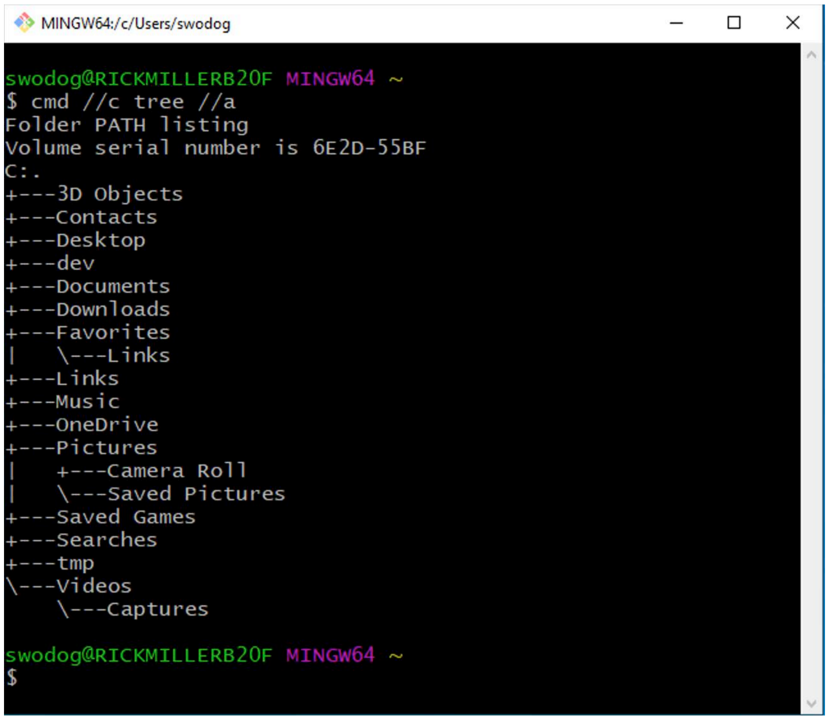
variables only. For more details on setting Windows environment variables watch this video: *https://www.youtube.com/watch?v=7LcDke9rLd0*.

7.2.1 Create Command Aliases In `.bash_profile`

Open a Command-Prompt terminal and type the command `tree`. You see that it works. The `tree` command ships with Microsoft Windows. Now, open a Git Bash terminal and run the `tree` command again. You'll see a message that says: "*bash: tree: command not found*" Here's how to fix it. While still in the Git Bash terminal type the following command:

```
cmd //c tree //a
```

This command is running the *Windows command interpreter* (cmd.exe) with the `/c` switch, which means run the specified command and then immediately exit. The first '`/`' of the "`//c`" sequence is an escape character. Next comes the command to execute, which is `tree`, followed by the escaped `/a` switch "`//a`", which tells the `tree` command to use ASCII characters in the output. You should see results similar to that shown in figure 1-62.



```
swodog@RICKMILLERB20F MINGW64 ~
$ cmd //c tree //a
Folder PATH listing
Volume serial number is 6E2D-55BF
C:.
+---3D Objects
+---Contacts
+---Desktop
+---dev
+---Documents
+---Downloads
+---Favorites
|   \---Links
+---Links
+---Music
+---OneDrive
+---Pictures
|   +---Camera Roll
|   \---Saved Pictures
+---Saved Games
+---Searches
+---tmp
\---Videos
    \---Captures

swodog@RICKMILLERB20F MINGW64 ~
$
```

Figure 1-62: Running tree Command in Git Bash Terminal

Referring to figure 1-62 — Try running the `tree` command without the `//a` switch if you're curious. Of course, running the `tree` command like this in Git Bash is a pain, so let me show you how to create a command alias in your *.bash_profile*.

Type the command `ls -al` to see if you already have a *.bash_profile* file in your home directory. If not, create it with Vim like so:

```
vi .bash_profile
```

Switch to Insert mode and type in the following lines:

```
alias tree='cmd //c tree //a'
alias dir='ls -al'
alias cls='clear'
```

When you've finished, hit `esc` to switch back to Normal mode then "`:wq`" to save the file and exit Vim. Close and reopen the Git Bash window for the *.bash_profile* settings to take effect. I figured while you were at it you may as well add aliases for common Windows Command-Prompt commands. When you're ready test the `tree` command. This time just type `tree`. If it doesn't work you screwed something up and you'll need to edit your *.bash_profile* file and fix the mistake.

To clear the terminal window you can now type the MS-DOS command `cls` or the UNIX command `clear`, the preference is yours. Same for getting a detailed listing of a directory. You can type either `dir` or `ls -al`. I was an MS-DOS command user way before I learned UNIX and aliasing familiar commands makes transitioning to the UNIX/Linux terminal a smoother experience. Another popular alias for `ls -al` is simply `ll`. You can add that alias to your *.bash_profile* if you so desire.

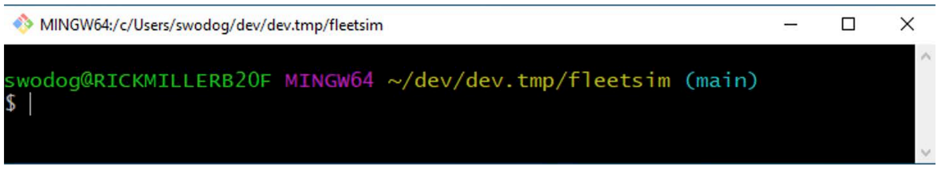
Now, one more important alias to add. Once again edit your *.bash_profile* and add the following line:

```
alias python='winpty python'
```

Save the file, close the terminal and relaunch. Type the command: `python` You should see the REPL work just fine now. To exit the REPL type exit().

7.2.2 Customize Bash Prompt

Git for Windows installs everything you need to start using Git. The Git Bash terminal's shell prompt is already configured to display the active Git branch as shown in figure 1-63.



```
swodog@RICKMILLERB20F MINGW64 ~/dev/dev.tmp/fleetsim (main)
$ |
```

Figure 1-63: Git Bash Shell Prompt Already Shows Git Branch

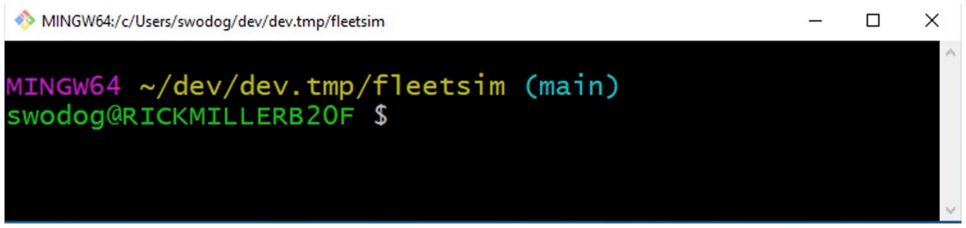
Referring to figure 1-63 — About the only critique I can make regarding the prompt configuration is that the first line's length grows too long for my taste when the path of the current working directory is long. To fix this I like to move the *usersname@host* section to the second line of the prompt.

To do this you'll need to edit the */etc/profile.d/git-prompt.sh* file, and if you want to edit the file with Vim then you'll need to launch the Git Bash terminal as Administrator. To do this right-click the Git Bash desktop icon and select **Run as administrator**. Click **Yes** on the pop-up. You will now be able to edit the file via the terminal using either Vim or Nano. First, however, make a copy of the *git-prompt.sh* file just in case you dork something up. This is good advice anytime you're tweaking important configuration settings. I recommend naming the copy *git-prompt.sh.original*.

Another way to edit the file is to use Visual Studio Code. The absolute path to the *git-prompt.sh* file is: *C:\Program Files\Git\etc\profile.d\git-prompt.sh* Note this path is in Windows format by the use of backslash characters in the path.

If you edit the file with Visual Studio Code you will also need to right-click the VS Code desktop icon and select **Run as administrator**.

To move the *username@host* section of the prompt to the second line, copy lines 14 and 15 of the original file and copy the code (or cut the code if you're ballsy) then add a new line below line 35 and paste the code. If you haven't already done so delete the original lines 14 and 15. Next, cut line 32, add a new line below line 35 and paste. Save the file and open a new Git Bash terminal to test your changes. The new prompt configuration will look similar to figure 1-64.



```
MINGW64 ~/dev/dev.tmp/fleetsim (main)
swodog@RICKMILLERB20F $
```

Figure 1-64: Modified Git Bash Shell Prompt

Referring to figure 1-64 — I've also increased the terminal font size from 12 to 16 points to make it easier to read here. I personally like this configuration but you may have another opinion. Feel free to tweak the *git-prompt.sh* file as you see fit. Just to be safe I recommend making another copy of the *git-prompt.sh* file and name it *git-prompt.sh.current* in case it gets clobbered by a Git for Windows update.

7.3 Linux

Linux uses one of two files to configure your shell depending on how you launch the terminal. These include *.bashrc* and *.bash_profile*.

When you launch a terminal emulator from the panel or desktop it will read shell configuration settings from the **.bashrc** file. This is referred to as a interactive *non-login shell*. On the other hand, if you log in to a Linux terminal locally or remotely the **.bash_profile** is used instead. This is referred to as an interactive *login shell*.

Perhaps the best way to remember which file is being used to configure the shell on Linux is that if you are already logged in to the machine and are using a GUI and you launch a terminal application by clicking an icon then the terminal will launch a **non-login shell**, which reads from **.bashrc**. If you're logging into a Linux machine remotely, like an AWS EC2 instance, then that session is running a **login shell**, which reads from **.bash_profile**.

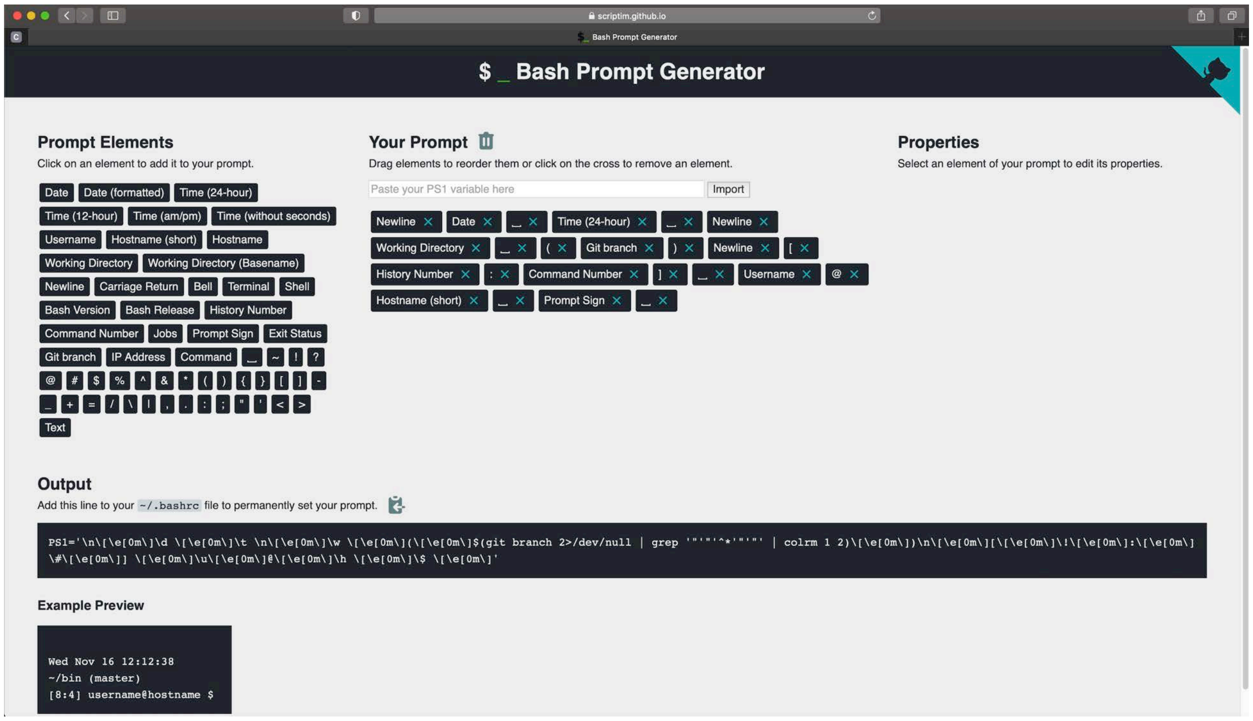
7.3.1 Create Aliases In `.bash_aliases`

On Linux Mint a *.bashrc* file already exists in your home directory. I recommend you open the file and study what it's doing. You'll see a section of code that checks for the existence of a ~/**.bash_aliases** file, which you must create if you want to add custom aliases. Also in the *.bashrc* file you'll find a handful of predefined aliases, some active, others commented out. You can uncomment the ones you might like to use or add them to your .bash_aliases. If you plan to login to your Linux machine remotely then you'll need to create a ~/**.bash_profile**.

7.3.2 Customize Prompt

You'll find the default shell prompt definition in the *.bashrc* file as well. The default prompt configuration is adequate and straightforward but considered by most experienced terminal users to be boring and insufficient. Shell prompt customization is a highly personal endeavor. I tend to favor multi-line prompts with a separate line for the date/time, another for the current working directory path, and some type of Git integration that shows me what branch I'm working on.

Like Vim configurations, you can find ready-made shell prompts online. Here's a link to a handy list of Bash prompt repositories on GitHub: *https://github.com/topics/bash-prompt* Try out the **$_ Bash Prompt Generator**: *https://scriptim.github.io/bash-prompt-generator/* Figure 1-65 shows a typical $ _ Bash Prompt Generator design session.



```
PS1='\n\[\e[0m\]\d \[\e[0m\]\t \n\[\e[0m\]\w \[\e[0m\](\[\e[0m\]$(git branch 2>/dev/null | grep '"'"'^*'"'"' | colrm 1 2)\[\e[0m\])\n\[\e[0m\][\[\e[0m\]\!\[\e[0m\]:\[\e[0m\]
\#\[\e[0m\]] \[\e[0m\]\u\[\e[0m\]@\[\e[0m\]\h \[\e[0m\]\$ \[\e[0m\]'
```

```
Wed Nov 16 12:12:38
~/bin (master)
[8:4] username@hostname $
```

Figure 1-65: $ _ Bash Prompt Generator Design Session

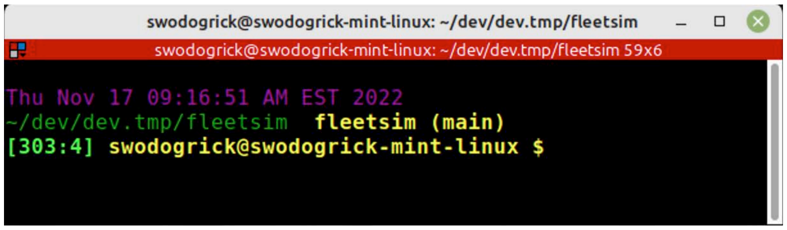
Referring to figure 1-65 — Designing Bash prompts has never been more fun! Just select and drag one or more Prompt elements to the Your Prompt section. You can reorder elements to get things just right. It's also a helpful learning tool, especially for someone who's never configured a Bash prompt. Copy and paste the prompt configuration from the Output section. You can see what the prompt will look like in the Example Preview box. The only critique I have with this excellent tool is that it doesn't let you add color codes. In the following section, I'll show you how to display the current Git branch in the prompt and give you the prompt code configured in color.

7.3.3 Display Git Branch In The Shell Prompt

Linux Mint comes with everything you need to use Git out of the box but you'll need to configure the Bash prompt to display the current Git branch. To do this you'll need to modify the prompt definition located in the *.bashrc* file. Open the *.bashrc* file with your editor of choice, navigate to line 60, comment out that line by adding a hashtag '#' to the beginning, create a new line and paste the following prompt configuration:

```
PS1='\n\[\033[35m\]$(/usr/bin/date)\n\[\033[32m\]\w \[\033[1;33m\]
\W$(__git_ps1 " (%s)") \n\[\033[1;32m\][\!:\#]\[\033[1;33m\] \u@\h $
\[\e[0m\]'
```

Note this is all one line of code. Breaking it down — `PS1` is the prompt variable. It is initialized here with a string containing a mix of *terminal codes*, *prompt variables*, and *command expansions*. It starts with a newline character `'\n'` to separate the prompt from the results of the previous command. Next is the color code for the color purple followed by an expansion of the current date followed by another new line character. The `'\w'` prompt variable prints the current working directory path `($PWD)` followed by a color change to yellow. The `'\W'` variable prints the basename of `($PWD)` or simply the current directory name followed by the expansion `$(__git_ps1 "(%s)"` which inserts the current Git branch. This is followed again by a new line character and the color code for bright green and the sequence `[\!:\#]` which denotes the shell command history number and the current command number. This is followed by another change to the color yellow followed by the *username@hostname* `'\u@\h'` and finally the prompt symbol `'$'` followed by a space and the final escape sequence `'\[\e[0m\]'`, which removes all formatting and attributes before the cursor is displayed. Figure 1-66 shows what this prompt looks like on Linux Mint running in a Terminator terminal window.



```
swodogrick@swodogrick-mint-linux: ~/dev/dev.tmp/fleetsim
swodogrick@swodogrick-mint-linux: ~/dev/dev.tmp/fleetsim 59x6

Thu Nov 17 09:16:51 AM EST 2022
~/dev/dev.tmp/fleetsim  fleetsim (main)
[303:4] swodogrick@swodogrick-mint-linux $
```

Figure 1-66: Linux Mint Shell Prompt with Git Branch

Referring to figure 1-66 — The purple date text is hard to read, I know. I look at this now and think it might be a good idea to remove *username@hostname* from the prompt because it's displayed in the Terminator title bar. That would make it just about perfect IMHO but I'll leave that to your discretion.

7.3.4 Export Environment Variables

Linux Mint installs software into the */usr/bin* directory, which is already part of the PATH environment variable. To see the value of the PATH variable type `echo $PATH`. If you need to add a path to your PATH environment variable you'll need to add a line to the *.bashrc* file. Let's say you want to add the *~/tmp* directory to your PATH. You'd need to add the following line to your *.bashrc* file:

```
export PATH="$HOME/tmp:$PATH"
```

This will add the path to the *tmp* directory located in your home directory to the existing PATH shell variable. Essentially, PATH is a variable name. To access the value a Bash shell variable contains prefix a dollar sign '$' to the variable name like so: `$PATH`

`HOME` is another existing shell variable. It contains the absolute path to your home directory. To see this value type `echo $HOME`. The `$HOME` will expand to */home/username*, where *username* is your home directory name. To this path the string “*/tmp*” is appended to give the absolute path to the *tmp* directory. To separate path strings use a colon character ‘`:`’. Finally, add the original path `$PATH` or commands will mysteriously stop working.

7.4 macOS

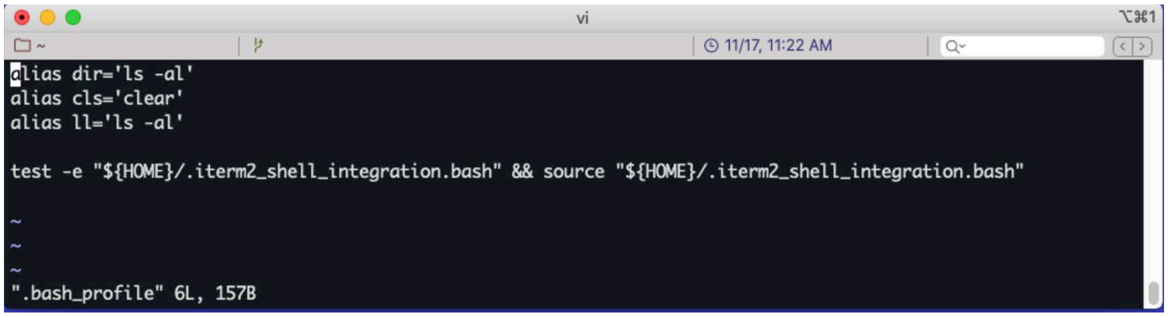
Terminals behave differently in macOS in that by default **launching a terminal launches a login shell**. This means you can place most of your shell configuration settings in your *.bash_profile*. This is the default behavior and unless you have a compelling reason to change it, I recommend leaving it as is. If the *.bash_profile* file does not exist then shell configuration settings will be read from the *.profile* file.

7.4.1 Create Aliases In `.bash_profile`

Being a long-time Microsoft Windows user before embracing macOS for software development, I like to alias the MS-DOS commands `dir` and `cls`. Open or create the *.bash_profile* file in your home directory and add the following aliases:

```
alias dir='ls -al'
alias ll='ls -al'
alias cls='clear'
```

Save the file and relaunch the terminal to test. While you’re at it, if you installed iTerm2 Shell Integration before creating the *.bash_profile* file, cat the *.profile* file and copy the line that begins with the word `test` and paste it into your *.bash_profile*. Your *.bash_profile* will look like figure 1-67.



```
alias dir='ls -al'
alias cls='clear'
alias ll='ls -al'

test -e "${HOME}/.iterm2_shell_integration.bash" && source "${HOME}/.iterm2_shell_integration.bash"

~
~
~
".bash_profile" 6L, 157B
```

Figure 1-67: macOS .bash_profile Settings - So Far...

Referring to figure 1-67 — OK, save the file, close any open terminal windows and relaunch. Type the command `it2attention fireworks` to make sure everything works as it should.

7.4.2 Customize Prompt

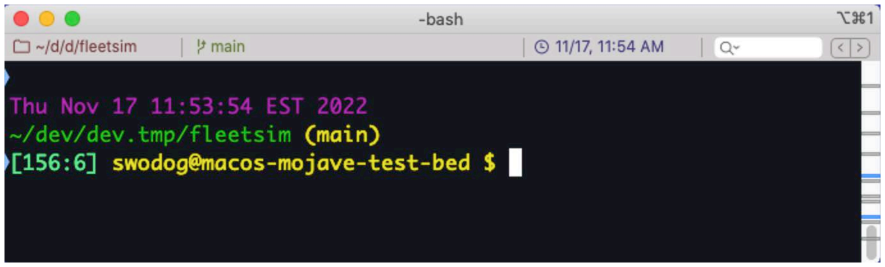
To customize the shell prompt you’ll need to load the git-prompt.sh file into your *.bash_profile*, which is installed with the Xcode developer command-line tools. If you haven’t installed Xcode or the developer command-line tools please do so now before proceeding. When you’re ready, edit your *.bash_profile* and add the following line:

```
source /Library/Developer/CommandLineTools/usr/share/git-core/git-prompt.sh
```

This will load and execute the *git-prompt.sh* file. Now, copy and paste the following prompt configuration into your *.bash_profile* below the line you just copied:

```
PS1="\n\[\033[35m\]\$(/bin/date)\n\[\033[32m\]\w \[\033[1;33m\]\$(__git_ps1
'(%s)')\n\[$(iterm2_prompt_mark)\]\[\033[1;32m\][\!:\#]\[\033[1;33m\] \u@\h $
"
```

Note this is one line of code. This will give you a shell prompt that looks like figure 1-68.



```
Thu Nov 17 11:53:54 EST 2022
~/dev/dev.tmp/fleetsim (main)
[156:6] swodog@macos-mojave-test-bed $
```

Figure 1-68: macOS Shell Prompt showing Git Branch

Referring to figure 1-68 — The color purple is hard to decipher but I personally like the subdued date and time readout at the top of the prompt.

7.4.3 Export Environment Variables

Exporting shell environment variables in macOS works the same as on Linux except you'll place the environment variable definitions and export statements in your *.bash_profile*. You'll need to modify your `PATH` variable to tell the OS where to find the executable for Visual Studio Code. Where these applications reside depends on where you dragged their application files to. Earlier, I recommended you create an Applications folder in your home directory. That's where I install applications if I'm given the choice. Figure 1-69 shows the applications residing in my *~/Applications* folder.

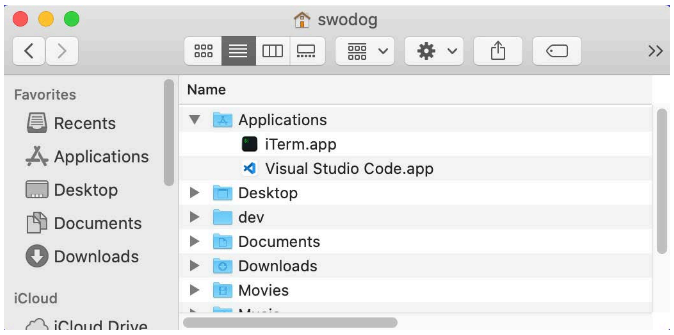


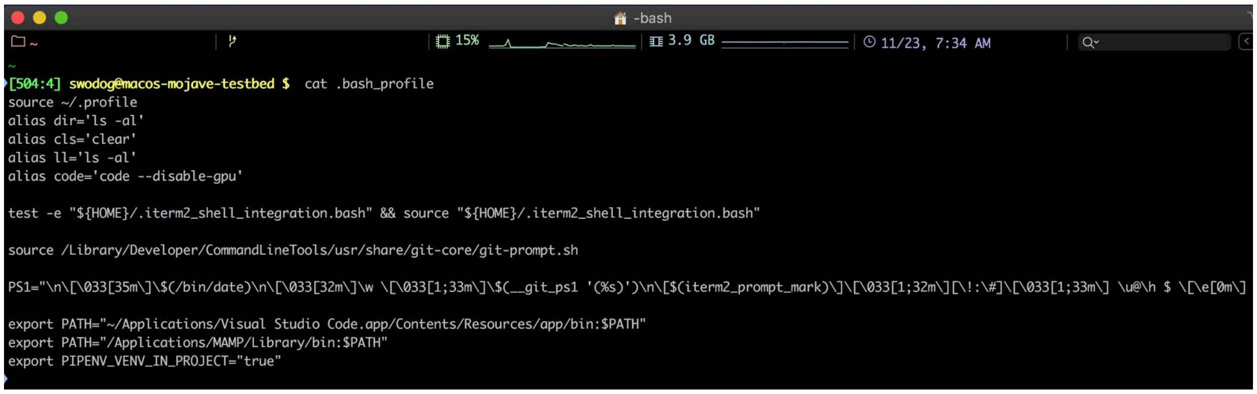
Figure 1-69: My *~/Applications* Folder Containing iTerm and Visual Studio Code

Referring to figure 1-69 — If you installed Visual Studio code to your *~/Applications* folder, fine. If not, it will be located in your */Applications* folder. Either way, to launch it from the command-line, edit your *.bash_profile* and add the following line:

```
export PATH="~/Applications/Visual Studio Code.app/Contents/Resources/app/
bin:$PATH"
```

The path used above assumes the application resides in the *~/Applications* directory. The settings in your *.bash_profile* will look similar to figure 1-70.

Referring to figure 1-70 — The actual settings in your *.bash_profile* may differ depending on whether or not this was the first time you created the file, how many software packages you have installed, existing aliases, and shell-prompt configuration.



```
~
[504:4] swodog@macos-mojave-testbed $  cat .bash_profile
source ~/.profile
alias dir='ls -al'
alias cls='clear'
alias ll='ls -al'
alias code='code --disable-gpu'

test -e "${HOME}/.iterm2_shell_integration.bash" && source "${HOME}/.iterm2_shell_integration.bash"

source /Library/Developer/CommandLineTools/usr/share/git-core/git-prompt.sh

PS1="\n\[\033[35m\]\$(/bin/date)\n\[\033[32m\]\w \[\033[1;33m\]\$(__git_ps1 '(%s)')\n\[$(iterm2_prompt_mark)\]\[\033[1;32m\][\!:\#]\[\033[1;33m\] \u@\h $ \[\e[0m\]

export PATH="~/Applications/Visual Studio Code.app/Contents/Resources/app/bin:$PATH"
export PATH="/Applications/MAMP/Library/bin:$PATH"
export PIPENV_VENV_IN_PROJECT="true"
```

Figure 1-70: macOS `.bash_profile` Settings Recommended So Far

Quick Review

Regardless of which operating system you ultimately use there are a handful of issues you need to understand regarding Bash shell configuration settings:

- What type of shell does your terminal application launch? login vs. non-login
- To which shell configuration file should you add your custom settings
- How to set environment variables
- How to customize the `PATH` environment variable
- How to create aliases
- How to customize your shell prompt (`ps1`)
- How to display the active Git branch in the shell prompt

Git for Windows is not a full-fledged Linux system but it does come fully configured to display the active Git branch at the command prompt. You should set Windows environment variables via the *System Settings* > *Environment Variables* dialog. Take care in modifying the prompt. If you do modify the shell prompt save a copy of the original *git-prompt.sh* file before making your edits.

Linux treats new terminal windows as *non-login* shells. Add shell customization settings to the *.bashrc file*. If you plan to login remotely to your machine then create a *.bash_profile* and add settings there as well.

MacOS launches new terminal windows with *login-shells*. Add shell customizations to your *.bash_profile*. You'll also need to modify your `PATH` variable to include the paths to Sublime Text and Visual Studio Code if you want to launch those applications from the command-line.

8 Configuration Checklist

Table 1-2 summarizes the development environment suggested configurations for each operating system discussed in this chapter.

	Windows	**Linux Mint**	**macOS**
Home Directory Name	No spaces in home directory name (All Operating Systems)		
Initial Housekeeping	View hidden files and folders View file suffixes Add shortcuts to frequently-used applications to desktop and taskbar/dock Add shortcuts to frequently-accessed hard drive(s) and folders Create development project folder Configure folders to open in home directory Configure terminal shortcuts to open in home directory or projects directory		
Terminal & Shell	PowerShell Command Prompt Git for Windows Git Bash (mintty) Terminals open in home directory	Terminal Terminator Bash Terminals open in home directory	Terminal iTerm2 Bash Terminals open in home directory
Text Editor	Vi/Vim - Customize via .vimrc and show line numbers Nano Visual Studio Code		
Integrated Development Environment	Visual Studio Code Add Microsoft Python Extension		
Package Managers	winget	apt	Homebrew (brew) Install *tree* command
Python 3.10 or greater	Install w/winget or manually	Install w/apt or manually w/ dpkg	Install with brew
Environment Variables & Shell Profiles	Set environment variables via System Properties Add aliases to *.bash_profile* Add alias to call python and tree with winpty Customize prompt via */etc/profile.d/git-prompt.sh*	Use *.bashrc* Aliases Shell variables Modify PATH as required	Use *.bash_profile* Aliases Shell variables iTerm2 shell integration Modify PATH to start VS Code and Sublime Text from command line

Table 1-2: Development Environment Configuration Checklist

Summary

The purpose of initial housekeeping configuration is to provide quick access to frequently used files, folders, and applications. Although terminology differs between macOS, Linux, and Microsoft Windows for similar concepts the objectives are the same. You want to see hidden files and folders, especially in your home directory, and you want quick and easy access to frequently used applications, folders, and files on either your desktop, taskbar, or both. The most important configuration you can make is to ensure your home directory name has no spaces.

A terminal supports text based human/computer interaction. The purpose of a terminal is to present user input to the computer and render output from the computer. Most terminal applications are actually terminal emulators and if necessary can be used to connect to older computer systems that require specific terminal types.

A shell is a program launched by the terminal when you log in. The shell dictates what types of commands and scripts can be executed. The Bash shell is a popular shell, which is used throughout this book, but there are others.

Together, the terminal and the shell provide a command-line interface to the computer. The prompt is the line on which you enter shell commands and is usually indicated by a dollar sign ‘`$`’ or hash tag ‘`#`’ character depending on whether you’re logged in as a *user* or as *root*. Cursor position is indicated with an underscore ‘`_`’ or box ‘█’character. You can customize the prompt via terminal preferences and shell profile settings.

The Git Bash (mintty) terminal and its version of the Bash shell will serve as the lowest common denominator for all Bash script examples in this book.

A software developer needs a good text editor. The editors discussed in this chapter, Vi/Vim and Nano work cross-platform. They work in a terminal and do not require a GUI or a mouse. They are ideal text editors for times when you must remote in to a UNIX-like virtual machine running in the cloud or in a Docker container.

Vim stands for **V**i **IM**proved and ships along with Nano on macOS, Linux, and Git for Windows. Vim has a steep learning curve but is extremely powerful and highly configurable but it’s not everyone’s cup of tea. If the thought of using key-commands to navigate text makes you want to throw a commode out of a window then Nano is a good alternative to Vim.

Don’t make the common novice mistake of editing text files with a word processor application as you may end up adding hidden codes to the file which will dork everything up.

Visual Studio Code is a powerful cross-platform Integrated Development Environment (IDE). It comes out of the box with a ton of helpful features and anything not already built in can be added via extensions. Visual Studio Code can be launched from the command-line by typing the command `code` followed optionally by the name of a folder or file. This works automatically when installing on Linux Mint with the `dpkg` command because the installation directory */usr/bin* is already part of the `PATH` environment variable. You’ll need to configure the `PATH` environment variable on Windows and macOS to launch VS Code from the command line.

Use a package manager when possible to install and manage software. Microsoft Windows 10 and 11 ship with *winget*, Linux Mint uses *apt*, and macOS can use Homebrew (*brew*).

Unless you have a compelling reason otherwise, you should always install the latest stable version of Python on your system. Windows, macOS, and Linux users have options regarding how to install Python. You can either download the installer directly from the Python.org website or use a package manager. You should be able to run the latest stable version of Python 3.x with the command `python3`.

macOS comes with Python 2 and 3 already installed. To execute Python 2.x use the `python` command. You won't need Python 2 for this book but you do need to be aware it's installed. To run Python 3.x use the `python3` command. To install the latest stable version of Python 3.x on macOS use the command `brew install python3`. If you install Python 3.11 with `brew` you can run it by using the `python3.11` command. To run Python 3.11 on Linux Mint use the command `python3`. All examples in this book will run on Python 10.8 and above.

Regardless of which operating system you ultimately use there are a handful of issues you need to understand regarding Bash shell configuration settings:

- What type of shell does your terminal application launch? login vs. non-login
- To which shell configuration file should you add your custom settings
- How to set environment variables
- How to customize the `PATH` environment variable
- How to create aliases
- How to customize your shell prompt
- How to display the active Git branch in the shell prompt

Git for Windows is not a full-fledged Linux system but it does come fully configured to display the active Git branch at the command prompt. You should set environment variables via the *System Settings* > *Environment Variables* dialog. Take care in modifying the prompt. If you do modify the shell prompt save a copy of the original *git-prompt.sh* file before making your edits.

Linux treats new terminal windows as *non-login* shells. Add shell customization settings to the *.bashrc* file. If you plan to login remotely to your machine then create a *.bash_profile* and add settings there as well.

MacOS treats new terminal windows as *login-shells*. Add shell customizations to your *.bash_-profile*. You'll also need to modify your `PATH` variable to include the path to Visual Studio Code if you want to launch it from the command-line.

Skill-Building Exercises

1. **Linux Commands:** Download a copy of *The CLI Commands Handbook* by Flavio Copes *https://flaviocopes.pages.dev/books/cli-commands-handbook.pdf* and research the commands used in this chapter including: `ls`, `ls -al`, `cp`, `mv`, `cd`, `chmod`, `mkdir`, `export`, and `alias`.

2. **Bash Scripting:** Download a copy of Pro Bash Programming: Scripting the GNU/Linux Shell by Chris F.A. Johnson, Apress, *https://doc.lagout.org/programmation/Shell%20/Pro%20Bash%20Programming.pdf* and research what constitutes a Bash script and how to make a script executable with the `chmod` command.

3. **Research `winpty` Command**: For Windows users who are using Git for Windows and the Git Bash terminal, the `winpty` command is used to call Windows commands or applications that are written to interface with a Windows terminal. Conduct some deeper research into the need for the `winpty` command.

4. **Windows Package Manager** `winget`: Windows developers — research the winget package manager and browse available packages to see what other software development tools or applications you may want to install.

5. **Customize Shell Prompt:** Dive deeper into shell prompt configuration for your particular operating system and shell. For the Git Bash terminal you'll need to edit the */etc/profile.d/git-prompt.sh* file. For Linux Mint you'll need to edit your *.bashrc*. For macOS add your prompt configuration to your *.bash_profile*.

Suggested Projects

1. **Windows Configuration:** For Microsoft Windows users — step through the sections in this chapter related to Windows and complete the configuration.

2. **Linux Configuration:** For Linux users — step through the sections in this chapter related to Linux and complete the configuration. Although the chapter focuses on Linux Mint, which is a Debian/Ubuntu derivative, the general steps remain the same.

3. **macOS Configuration:** For macOS users — step through the sections in this chapter related to macOS and complete the configuration. Note that if you are using older versions of macOS you may get warning messages when installing packages with Homebrew. Newer versions of macOS, especially Ventura and beyond, will have a slightly different but similar user interface compared to screenshots of Mojave and Catalina used in this chapter.

4. **Terminal Split Panes and Multiplexing:** For Linux users using Terminator and macOS users using iTerm2 research advanced terminal features such as splitting the terminal window into multiple vertical and horizontal panes. Also, research terminal multiplexing with `tmux`.

5. **Git Bash Terminal Multiplexing:** Windows users — Scour the Internet and search for ways to add terminal multiplexing to the Git Bash terminal. (*Hint:* msys2)

6. **Windows Subsystem for Linux:** Windows users — If you want a more robust implementation of Linux vs. the streamlined set of tools offered by Git for Windows, research the Windows Subsystem for Linux.

7. **The Awesome Vim Configuration:** If you want to use Vim I suggest you install the Awesome vimrc configuration found at the The Ultimate vimrc GitHub repository: *https://github.com/amix/vimrc* Follow the instructions found in the repository's README.md.

8. **Update Bash on macOS:** macOS ships with an older version of Bash, which will work perfectly fine for all the examples in this book, still, it's a good idea to upgrade. To check the current version of Bash type: `bash --version` If it hasn't already been upgraded it will be version 3.2.x or thereabouts. Use the Homebrew package manager to install the latest edition of Bash. While you're at it install the bash-completion package.

9. **Neovim Installation and Configuration:** Vim doesn't have to be brutally minimalistic nor boring. In fact, it can be fun to install different flavors of Vim and scour the Internet for interesting ways to configure and customize the text editor for software development. Download and install Neovim *https://neovim.io* and research customization options. One option is to use a ready-made theme like LunarVim *https://www.lunarvim.org*.

Self-Test Questions

1. In your own words state why you think it's important to have a properly-configured development environment.

2. What command is used to run tools and applications written for the Windows console on the Git Bash (mintty) terminal?

3. What's the difference between a login and non-login shell?

4. Which shell configuration file does a non-login shell use? How about a login shell?

5. On macOS, what type of shell is being launched by default, login or non-login?

6. On macOS, what command is used to run the latest stable edition of Python 3.x?

7. On macOS, assuming you have installed the packages *python3* and *python@3.11* via Homebrew, what's the difference between running `python` vs. `python3` vs. `python3.11`?

References

Official Python Website, *https://python.org*

Bruce Chittenden, James Hyde, and Jeffrey P. Radick. 1982. A scheme for terminal I/O not requiring interrupts. In Proceedings of the 20th annual Southeast regional conference (ACM-SE 20). Association for Computing Machinery, New York, NY, USA, 66–72. https://doi.org/10.1145/503896.503909

The DEC VT100 Terminal, *http://www.columbia.edu/cu/computinghistory/vt100.html*

DEC VT100 Image, Jason Scott, CC BY 2.0 <https://creativecommons.org/licenses/by/2.0>, via Wikimedia Commons

Teletype Image, Arnold Reinhold, CC BY-SA 3.0 <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons

Pro Bash Programming: Scripting the GNU/Linux Shell, Chris F. A. Johnson, Apress, ISBN-13: 978-1-4302-1998-9, Source: *https://doc.lagout.org/programmation/Shell%20/Pro%20Bash%20Programming.pdf*

MinTTY Terminal, *https://mintty.github.io*

MinTTY Terminal Options, *https://mintty.github.io/mintty.1.html#OPTIONS*

Git For Windows, *https://gitforwindows.org*

MSYS2, *https://www.msys2.org*

Git SCM, *https://git-scm.com*

iTerm2, *https://iterm2.com/index.html*

Xcode Releases, *https://xcodereleases.com*

Apple Developer Website, *https://developer.apple.com*

Terminator Linux Terminal, *https://terminator-gtk3.readthedocs.io/en/latest/*

The Ultimate vimrc Repository, *https://github.com/amix/vimrc*

GNU Nano Editor Website, *https://www.nano-editor.org/dist/v2.9/nanorc.5.html*

Top Five Cross-Platform Developer Text Editors, YouTube Video, PulpFreePress, *https://www.youtube.com/watch?v=k3wwDcyN8yA*

Microsoft Visual Studio Code Website, *https://code.visualstudio.com/*

RPM Package Manager, *https://rpm.org*

Yellowdog Updater Modified (yum) Package Manager, *http://yum.baseurl.org*

Advanced Package Tool (apt) Package Manager, *https://wiki.debian.org/Apt*

winpty Command GitHub Repository, *https://github.com/rprichard/winpty*

How-To: Setup Prompt Statement Variables website, *https://ss64.com/bash/syntax-prompt.html*

ANSI Escape Code, Wikipedia, *https://en.wikipedia.org/wiki/ANSI_escape_code#Escape_sequences*

cmd.exe Command Reference, *https://learn.microsoft.com/en-us/windows-server/administra-*

tion/windows-commands/cmd

Notes