Introduction to the Command-line

The *command-line* is a way of using a computer by entering textual commands one after another which cause programs to run and do useful work.

It dates back to the 1960s and is still widely used today because it's much, *much* easier to write and maintain programs that use text than ones that use graphics.

This handout summarizes the concepts and context that are most helpful to know as you begin learning to use the command-line.

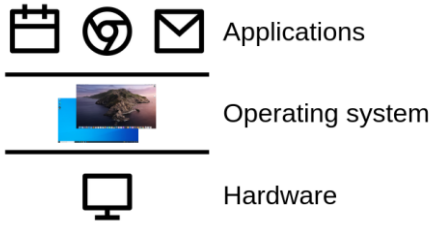
Where the command-line fits in

Picking up the command-line requires understanding it's place in the whole computer system.

Every computer comes with an operating system. The *operating system*, or OS, is a complex collection of cooperating software programs, some of them with special functions and properties.

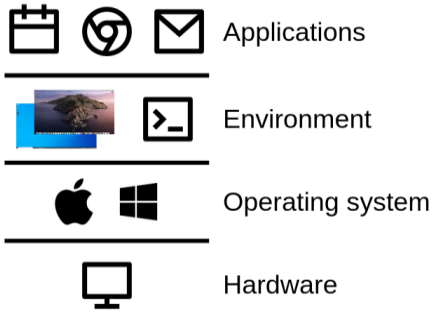
The purpose of the OS is to make life easier both on the user (you) and on software programmers by simplifying the technical details of the computer's hardware. For example, when you attach a USB thumb drive to your computer, the OS automatically takes care of recognizing and opening it. And when a program asks for a file from the thumb drive, the OS takes care of shepherding the data from the thumb drive, without the program needing any thumb drive-specific instructions to cope. The OS takes care of *operating* the computer hardware, so that you and your programs can take care of more interesting tasks.

Two OSes dominate the market as far as most people are concerned: Microsoft Windows and Apple macOS, although there are other options as well for both everyday and special-purpose computing.



Most people understand their computer as a system of three layers, of which the OS is one. The bottom layer is the computer hardware. On top of this foundation rests the OS, which must be compatible with the hardware. Finally, on top of the OS rest the *applications*, the programs which people use to get their work done. Just like the OS needs to be compatible with the underlying hardware, so the applications need to be compatible with the OS.

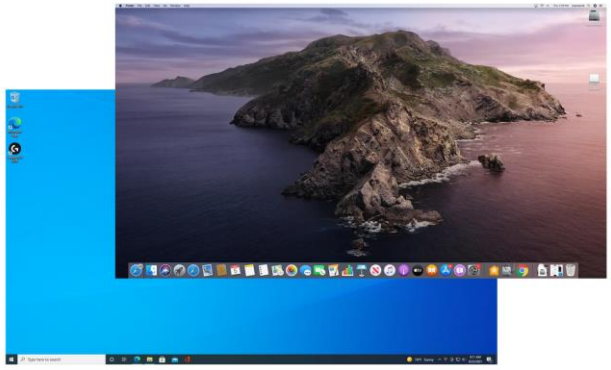
But that isn't the whole truth. **A slightly more accurate picture adds a fourth layer called the environment:**



The environment is a software system that provides an overarching user interface to the computer and the operating system.

It's so called because it offers the human user an "environment" inside which they can launch, manage, switch between, and close applications. **The environment is separate from the OS**, but it's

absolutely essential for users because it is the "face" of the OS, through which users can interact with it.



When you think of Windows or macOS, it's actually the environment you're thinking of. Both OSes provide a *graphical environment*, where applications are given windows inside which they may draw their user interfaces.

Besides giving applications their windows, these graphical environments also provide overarching features, like the taskbar and start menu in Windows or the menu bar and dock in macOS.

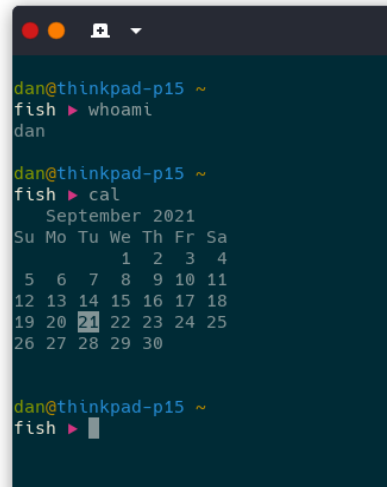
The point of explaining all this is to make the following statement:

The command-line is an environment.

Just like the graphical environments of Windows and macOS provide ways for you to control your computer and places for applications to create user interfaces, so does the command-line.

The main difference is that, instead of a graphical environment, the command-line is a *textual environment*.

What the command-line is like



The command-line is much like an infinite sheet of paper, on which you, your operating system, and your applications can take turns writing. This is because, decades ago, that's how things actually worked: a person would sit at a keyboard and enter a command, and the results would be printed on a long roll of paper.

Of course nowadays we use monitors, which open up exciting new possibilities like erasing and rewriting text that has already been written, but many important programs never do this specifically so that their output can be printed on paper or saved to a file.

How we access the command-line

We do most of our work in a graphical environment that we don't want to leave, and so we access the command-line by running a program called a terminal emulator.

A **terminal** is a combination keyboard and printer or monitor from the days when a building or even a campus would share a single, mainframe computer. Terminals in peoples' offices would be connected to the mainframe by wires running throughout the building or campus, and had no computational abilities of their own.

The **terminal emulator** is a normal graphical program that pretends to be a terminal, so that the OS can provide the command-line *through* the emulator without caring that it's actually a graphical program.

There's more than one command-line

Windows and macOS both give you only one graphical environment. It's a major part of their product and their business model. But there is no technical reason why a single computer can't have more than one environment of any kind, so that users can choose what they like.

Windows and macOS provide default command-lines and terminal emulators, but it's possible to install more of both. You may grow to be a power user, and find that third-party terminal emulators offer you features that you can't get in your OS's default emulator. You may also one day decide to try different command-lines for the same reason.

To prepare you for this likely future there's a handful of command-lines, or *shells*, you need to be aware of.

A note on terminology. The term *environment* isn't actually used all that often because it's a pretty abstract concept. In practice most people talk about the desktop when they mean a graphical environment and the command-line when they mean a textual environment, even though desktops and command-lines are actually only one kind of graphical and textual environment, respectively.

For command-lines the figurative term **shell** is used much more often. Strictly speaking, it's a shorter synonym of *environment*. We could say that Windows and macOS both offer graphical shells. **But for historical reasons, people actually use the term *shell* to refer to a command-line environment.**

Windows comes with two shells and two corresponding emulators, one called cmd.exe and the other called PowerShell.exe. If you use Windows you should focus on learning the newer PowerShell while being aware that you may find information on the Web about using cmd as well.

macOS comes with several shells and one emulator. Unlike on Windows, they don't share a name. The default shell is called Zsh and the emulator is called Terminal.app. macOS also includes the bash, ksh, and tcsh shells, but except for bash you'll very probably never use them.

There's some extra stuff you should know about the relationship between the shells and the OSes we've discussed. While cmd and PowerShell are both Microsoft products, none of the shells included with macOS belong to macOS or even Apple.
Zsh, bash, ksh, and tcsh are all independent projects. They are free and open source software, and can be used on Windows as well as on macOS.

bash comes preinstalled on macOS and Linux, and can be installed on Windows. Linux is a family of free and open source operating systems that you're likely to come into contact with sooner or later. bash is an older shell, but also so popular and reliable that it serves as a de facto common denominator across systems and organizations.

The last shell you should know about is called fish, short for Friendly Interactive SHell. fish is a newer shell with modern features and conveniences that you won't find in, say, bash. Further, as the name implies, fish emphasizes user-friendliness, giving it a more consistent and gentler learning curve. Neither Windows nor macOS come with it, but it's not difficult to install. It is this writer's choice, and my recommendation is somewhat biased by my personal preferences.

Which command-line to learn first

So which should you learn first? That's difficult to say. No matter what you choose, you're making some kind of trade-off. One might be easier to use, but you need to install it first. Another might be already installed, but not on every OS you use. And a third, while it is installed in many places, may be more difficult to learn.

This handout takes the approach of recommending books instead of shells--one each for Windows and macOS--on the hope that the quality of your learning resources is easier to maximize then the optimality of your choice of first shell. The books cover the PowerShell and Zsh shells, respectively.

But besides books, YouTube is a first-rate resource for finding tutorials of widely varying styles (and degrees of quality). I couldn't find a book that teaches you how to install bash on Windows 10, but "bash shell windows tutorial" turned up tons of results.

Windows users should pick up *PowerShell for Beginners* by Ian Waters, published by Apress. The book takes a gradual and methodical approach that focuses on giving you a strong foundation on which to later build more practical skills. The book reflects PowerShell's intended use as a scripting and automation tool more so than a shell.

Windows users should also know a bit more about the history of their OS and how their command-line has been shaped.

Windows started in the mid-80s as a graphical shell for the MS-DOS operating system. Where previously Microsoft's command-line had been a critical part of their most important product, Windows' success caused the command-line to be pushed to the background. The big idea was that everything should be able to be done graphically, and so the command-line became a legacy tool.

PowerShell, which is Microsoft's newer command-line, is not neglected like the old one. But this is because Microsoft has a particular use case in mind: that PowerShell is a tool for scripting and automation--for making it easy and quick to write tailored programs that perform custom tasks useful for a variety of business and other purposes. *Not* as a way of *using* your computer.

This is in stark contrast to shells for macOS and Linux, which do still care about being a human user interface. So when you learn PowerShell, you *are* going to learn how to use a command-line, but the focus will be on writing scripts. You'll learn transferable knowledge, but when you come to pick up bash on Linux or Zsh on macOS--or if you install one of those on Windows--be ready to encounter a very different style of doing things.

macOS users should pick up *Tweak Your Mac Terminal* by Daniel Platt, published by Apress. The book takes a practical approach that focuses on giving you skills and knowledge that can be composed as needed into more complex solutions. The book reflects Zsh's strengths in interactive use, rather than automation.