Introduction to the Terminal (Mac/Linux) and Powershell (Windows) and standard file system

Learning Goals:

- Learn about the Unix/Linux Terminal or Windows Powershell
- Navigate the file system using the terminal/powershell (“terminal” henceforth)
- Create folders and files with the terminal
- Learn about the standard filesystem on a computer (UNIX/Linux, macOS and Windows)

Prerequisites:

- Internet access
- A computer with a terminal application. Basically, any computer (even a Chromebook!)
- Windows users – most linux commands have direct equivalents in powershell. Here is a cheatsheet of powershell commands: pwsh cheatsheet

Reading overview

This short reading is a very simple introduction to your computer's terminal and the system of files and folders on your computer.
We will not attempt to cover the terminal exhaustively, instead, we will focus on the basics needed to get you started and help you to use Git and GitHub in the future. We'll follow this reading up with an in-class exercise to build up our terminal muscles.

The Terminal

We are used to looking at and operating our personal computers using graphical user interfaces (GUIs). This means that we routinely open files or folders by first finding them using the "finder" (Mac) or "browser" (Windows) graphical interface and then clicking on them. We perform a search using a search bar. We navigate the internet using a browser such as Chrome, Firefox or Safari, we write documents using Google Docs or Microsoft Word. All these are examples of software that uses graphical user interfaces or GUIs. GUIs have revolutionized computers effectively making them accessible to anyone. Before GUIs, however, there was the `Terminal`.

The terminal is the command line (text) interface to your computer file system and operative system. Nowadays it is referred to interchangeably as the terminal, console, shell, or command line. They were called "terminals" because they were literally the end of the line - information from a centralized computer (a "main frame" - generally in another room or even another building) came out as far as the terminus or terminal, and no further. The earliest terminal was the "teletypewriter" or "tty", literally a modified typewriter.

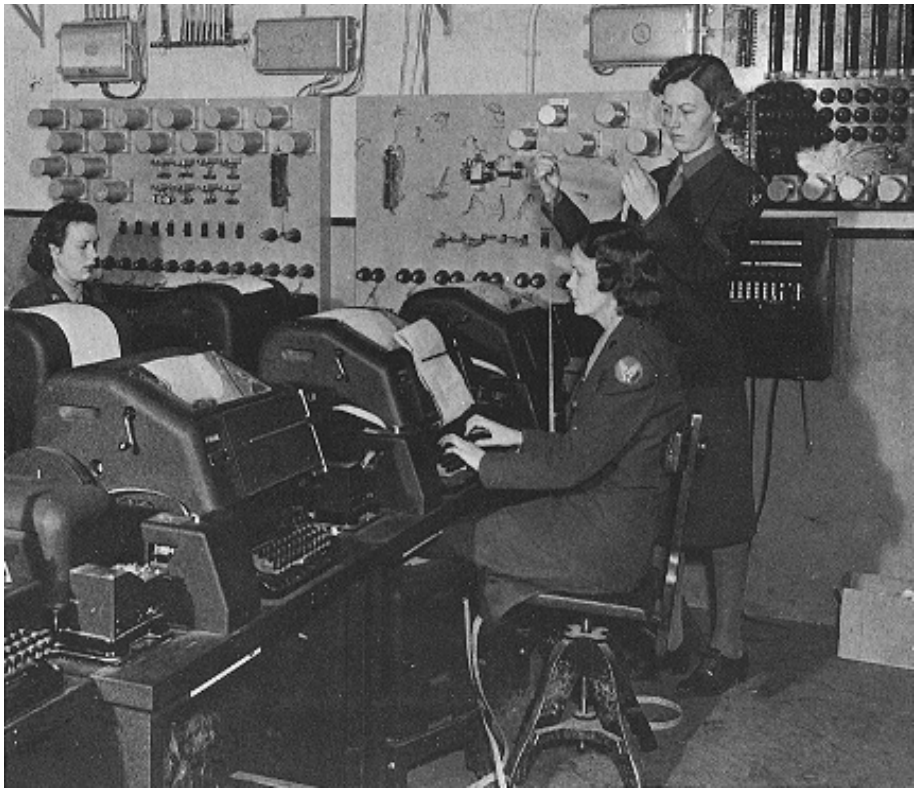


Image source Wikipedia

Later a one-piece keyboard and monitor combination called a "console" took its place but, of course, these were also called terminals. The "shell" is the software that allows you to interact with the computer's operating system using terminal/console. It is mainly used to run other programs by typing the name of the program plus any options at the "command line", the place where your typing appears (read here for more information).

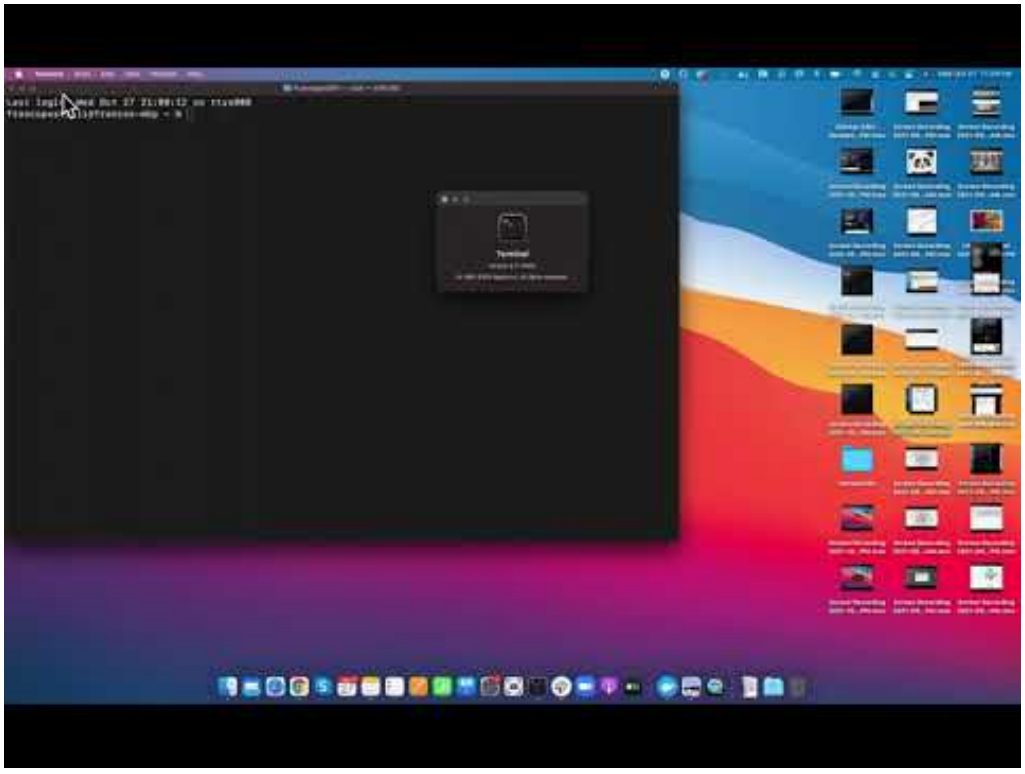
Today's "terminal" is not an antique thing you plug into your computer, but rather just a window on your desktop that looks and acts like the terminals of yore. But why would we want to seemingly step back in time like this? Because, despite the ease of use of the GUI interface to your computer, the terminal is powerful and often provides faster and easier ways of doing things once you get a little good at using it. In fact, there are still certain things that you can do using the terminal that you cannot do using the GUI.

Most of the following examples will use Mac OSX to demonstrate how to access the Terminal and use basic shell commands (because we use Mac OSX). If you are using Linux (Ubuntu for example) here is a nice tutorial on how to access and use the terminal. If you are using Windows, the terminal application is called PowerShell – most or all of the commands we will cover should work "as is" in PowerShell. (If a command doesn't seem to be working in powershell, check out the pwsh cheatsheet – it even has a little AI chatbot that can answer questions like "How do I create a new file?")

Opening a terminal

In Mac OSX you can find the terminal using the search bar.

If you wish, you can add the Terminal Icon to your Dock.



In Windows, locate and run the "Windows Powershell". Open the **Start** menu, type **Windows PowerShell**, select **Windows PowerShell**, then select **Open**.

The Unix/Linux and Windows Filesystems

https://en.wikipedia.org/wiki/Unix_filesystem

On a mac or linux, filesystem appears as one rooted tree of directories.[1] such volumes can be mounted on a directory, causing the volume's file system tree to appear as that directory in the larger tree.[1] The “root” (but more like the trunk) of the entire tree is denoted / (the front slash). Everything – everything – on a mac or linux computer is under this root. For example, Larry’s home directory is “/Users/lkcormack”, which can be read right to left as “Larry’s home folder is in the Users directory under root.”

Windows is a little different. In windows, each physical storage device is denoted by a letter, and then files on that device are denoted by a *file path* starting with that letter. So, for example, Larry’s directory on a windows machine might be “C:\Users\lkcormack” (note the backslashes). If you’re on a windows computer, you might be wondering what happened to “A” and “B”. These letters denote the two floppy disc drives that most early personal computers came with. While floppy disks aren’t used anymore, their legacy lives on with the letter “C” denoting your computers hard disk (which, er, really isn’t a disk anymore).

In the original Bell Labs Unix, a two-disk setup was customary, where the first disk contained startup programs, while the second contained users' files and programs. This second disk was mounted at the empty directory named usr on the first disk, causing the two disks to appear as one filesystem, with the second disk’s contents viewable at /usr. Confusingly, /usr is now used for completely different things that you don't need to worry about, and your stuff is in your home directory in the "Users" folder.

https://linuxfoundation.org/blog/classic-sysadmin-the-linux-filesystem-explained/

Navigating the filesystem

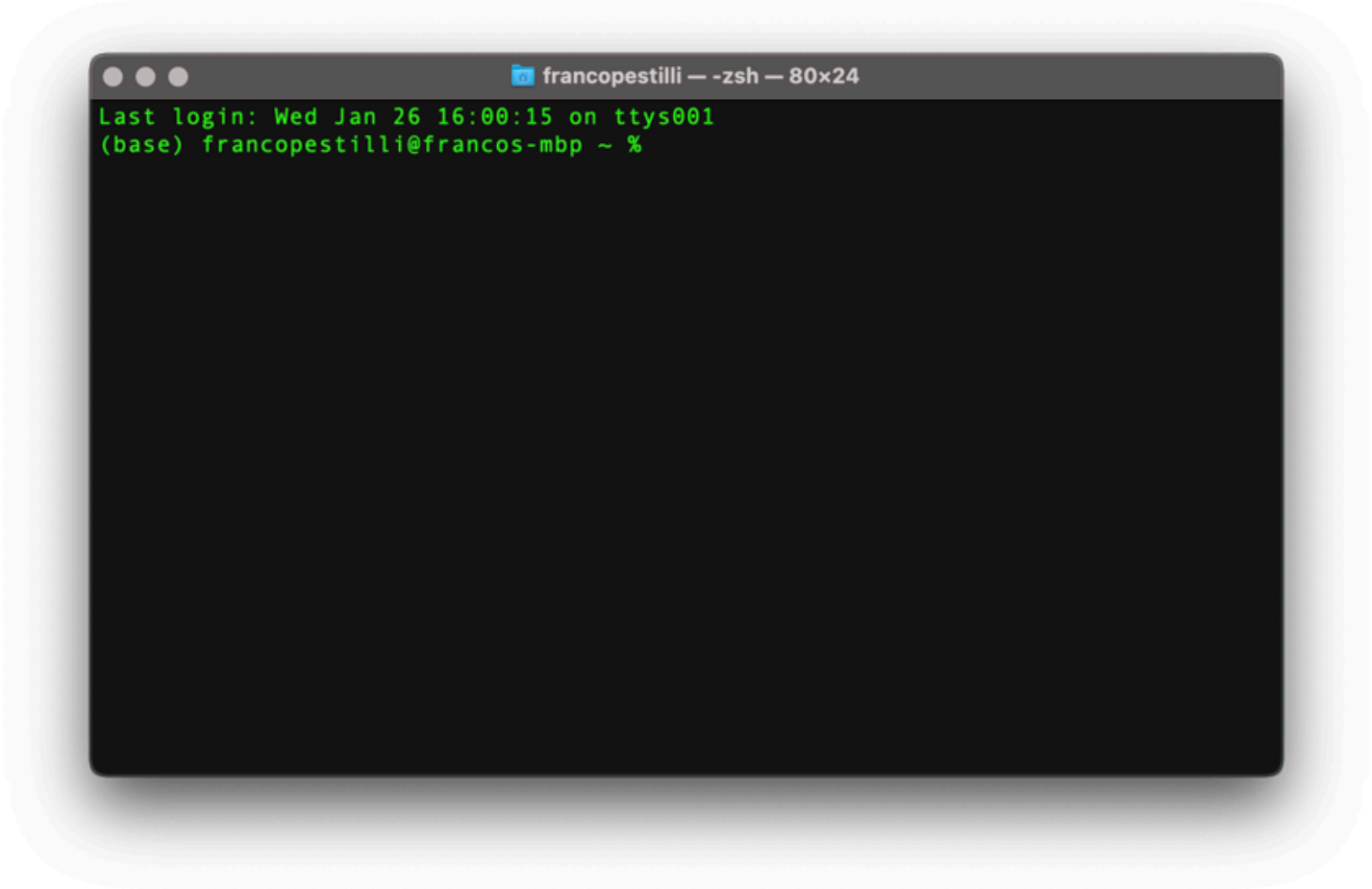
By default Unix/Linux Terminals will show some basic information to help identify the user who opened the terminal and the system the terminal was opened on. This information can seem trivial on a PC because well we know who we are and on which computer we are running. Yet remember that terminals were initially used by multiple users on different shared mainframe computers.

In the video above you can see that my user name (`francopestilli`) and computer names (`francos-mbp`) are reported:

```
Last login: Wed Oct 27 21:00:12 on ttys000
```

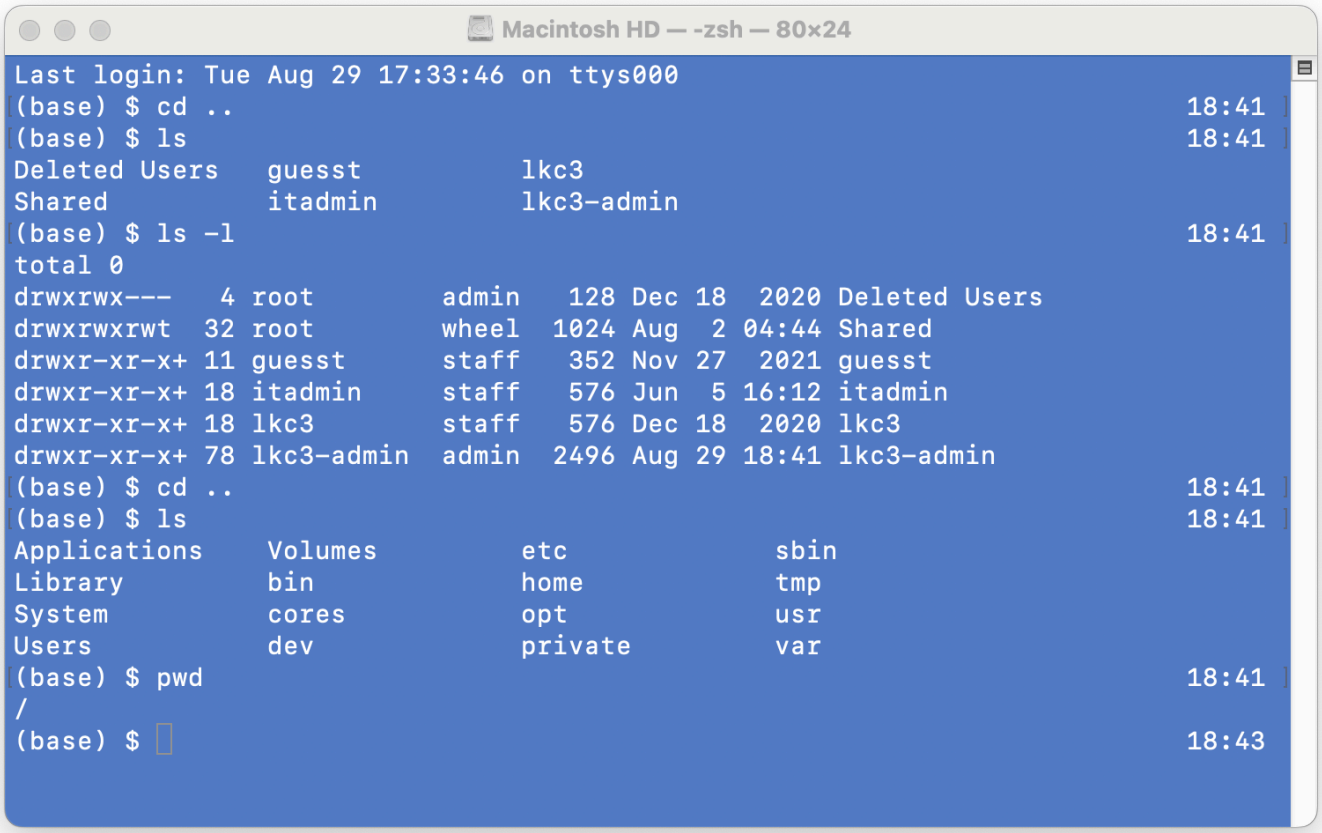
```
francopestilli@francos-mbp ~ %
```

In addition to that, the date of last login (`Last login: Wed Oct 27 21:00:12`) is reported.



Sidenote: you can customize your prompt

For example, here is Larry's prompt, which is short and has the current time on the right.



Useful commands that can be used in the terminal

We can type commands inside the Terminal that will allow us to perform operations on files and folders in the hard drive of the computer. There are many many commands in a Unix/Linux shell, see here for a few.

In a terminal type:
`pwd` - a.k.a. `P`rint `W`orking `D`irectory. The command will show you the current location on your file system. This is the folder where your terminal would read or write files when requested.

`ls` - a.k.a. LiSt. The command will show "list" the files and directories in your current directory.

`cd` - a.k.a. Change Directory. The command will allow navigating away from the current directory. For example `cd ..` will move up one directory, or `cd ~` will return to your home directory (the home directory is the one where the terminal generally starts when it first opened.

`touch` - a.k.a. Touch a file to create the file. This command will create an empty file with the specified file lane for example: `touch franco.md` will create a file in the current directory with my name and the suffix `.md`. This is a strange name; if used for an existing file, it updates the time the file was modified, i.e. the time it was last "touched". But if the file doesn't exist, it creates one.

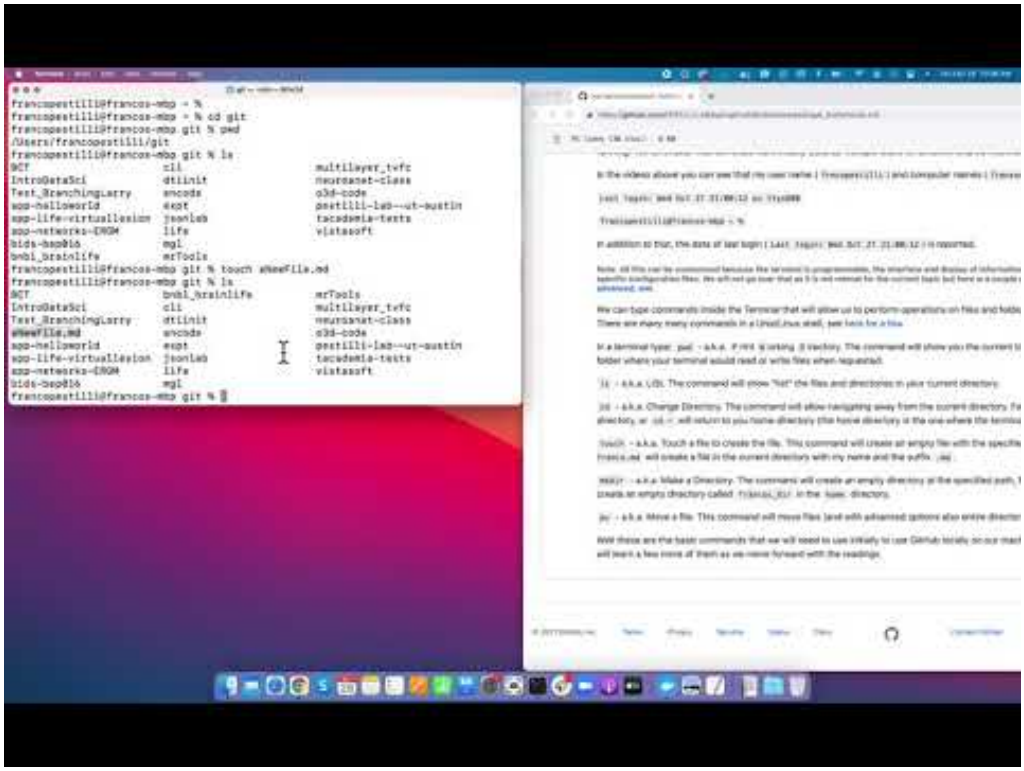
`mkdir` - a.k.a. Make a Directory. The command will create an empty directory at the specified path, for example: `mkdir ~/franco_dir` will create an empty directory called `francos_dir` in the `home` directory.

`mv` - a.k.a. Move a file. This command will move files (and with advanced options as well as entire directories).

`rm` - a.k.a. ReMove a file or a directory. This command will delete files and directories (to remove entire directories and additional parameters, the video below shows an example of how to delete a full folder).

These are the basic commands that we will need to use initially to use GitHub locally on our machines using the Linux/Unix terminal. We will use them in class and learn a few more of them as the course progresses.

Below is a short video that uses the commands above to give you a sense of the outputs.



Let's do something with the terminal

(1) Let's make a new folder

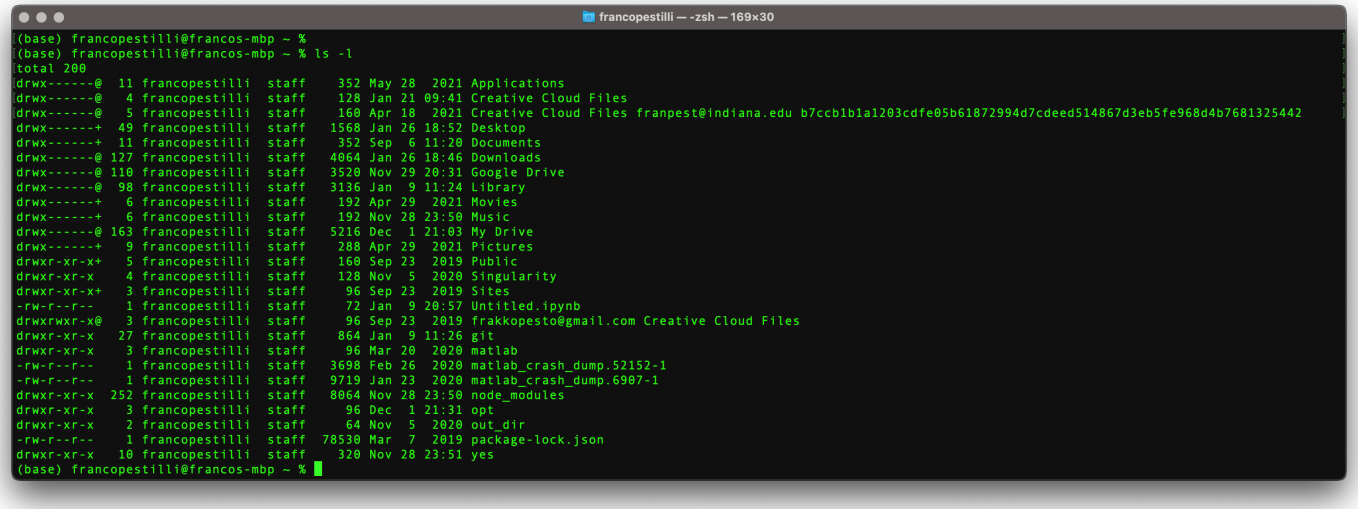
(1.1) Open a terminal.

Type `cd ~` to navigate to your home directory (indicated by the symbol `~`).

(1.2) Let's take a look at the folders and files we have in the home directory.

Type `ls -l` This will show all your files and folders. Some of which you might know you have others might be new to you.

The image below shows all the visible files in the home directory of Franco's computer.

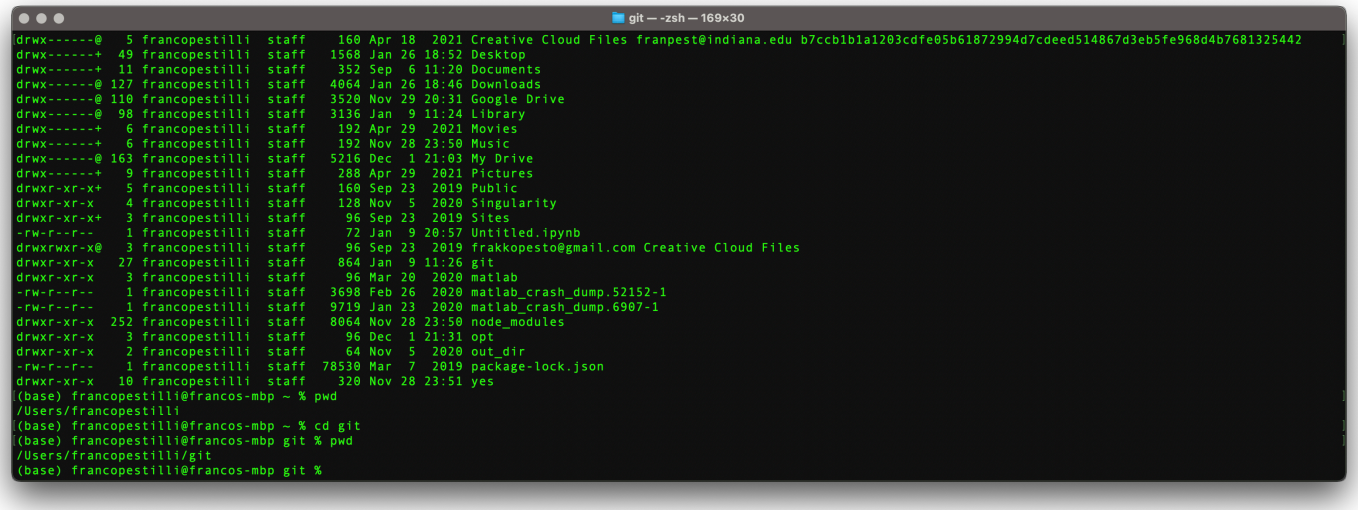


(1.3) Let's now create a folder.

Type `mkdir GitHub` this command will create a folder `GitHub` that will use later on to organize the repositories from GitHub.

(1.4) Navigate inside the folder `GitHub`.

Type `cd GitHub`, and the command will change our current location from the home dir (`~`) to the directory we just created. You can check that by typing `pwd` (print working directory), the command will show you the path to the current (working) directory.



(1.5) Create a file inside the folder `GitHub`

(1.5) Create a file inside the folder `GitHub`

Mac/Linux: Type `touch readme.md` This will create an empty file inside the current directory, that at this point should be the directory `GitHub`.

Windows: This is a bit more cumbersome. Ask the cheat sheet for help. The command should look something like `New-Item -ItemType File -Path "newtextfile.txt"`

(1.6) Edit the file `readme.md` and use MarkDown to make it informative.

Open the file using a basic text editor. On the Mac, you can use TextEdit; on Windows, you can use NotePad. If you're using Linux, you probably don't need advice on this.

Write a short title using the top heading to explain what the goal of the folder `GitHub` will be. For example, "Folder for all my GitHub repositories."
In the second line, write your name, email, and GitHub username using smaller headings.

Save the file.

A couple of notes.

Note 1. The Terminal is customizable.

As we saw above, Larry and Franco have different custom shell prompts. The prompt and other shell behaviors can be customized. We won't cover this, but here are a couple of good resources one, and another, more advanced, one.

Note 2. Commands have options.

In the previous video, we used additional parameters (a.k.a. options) to generate richer outputs from the commands or to automatically delete the folder and the files inside the folder. You can find out more about how to use a command in Unix/Linux and the available options for the command by typing the command `man` (for **man**ual) followed by the command, for example, `man ls`. You can page through the help using the spacebar, and quit by hitting the `q` key.

Note 3. The shell is programmable.

You can actually write scripts (programs) consisting of shell commands to automate tasks like copying hundreds of files of a certain file type from one location to another. This is beyond the scope of this course, but remember that this is possible. You can Google "shell scripting" for more information.

The UNIX File System Hierarchy

Above, we took a shallow dive into our home directory. We have not really talked about the "file system" and how it works. Here we will try to give an introduction to a typical file system. This is not meant to be exhaustive but should give you a big picture of a file system's structure.

Unix-like operating systems use a File System with one root directory, and every file existing on the system is

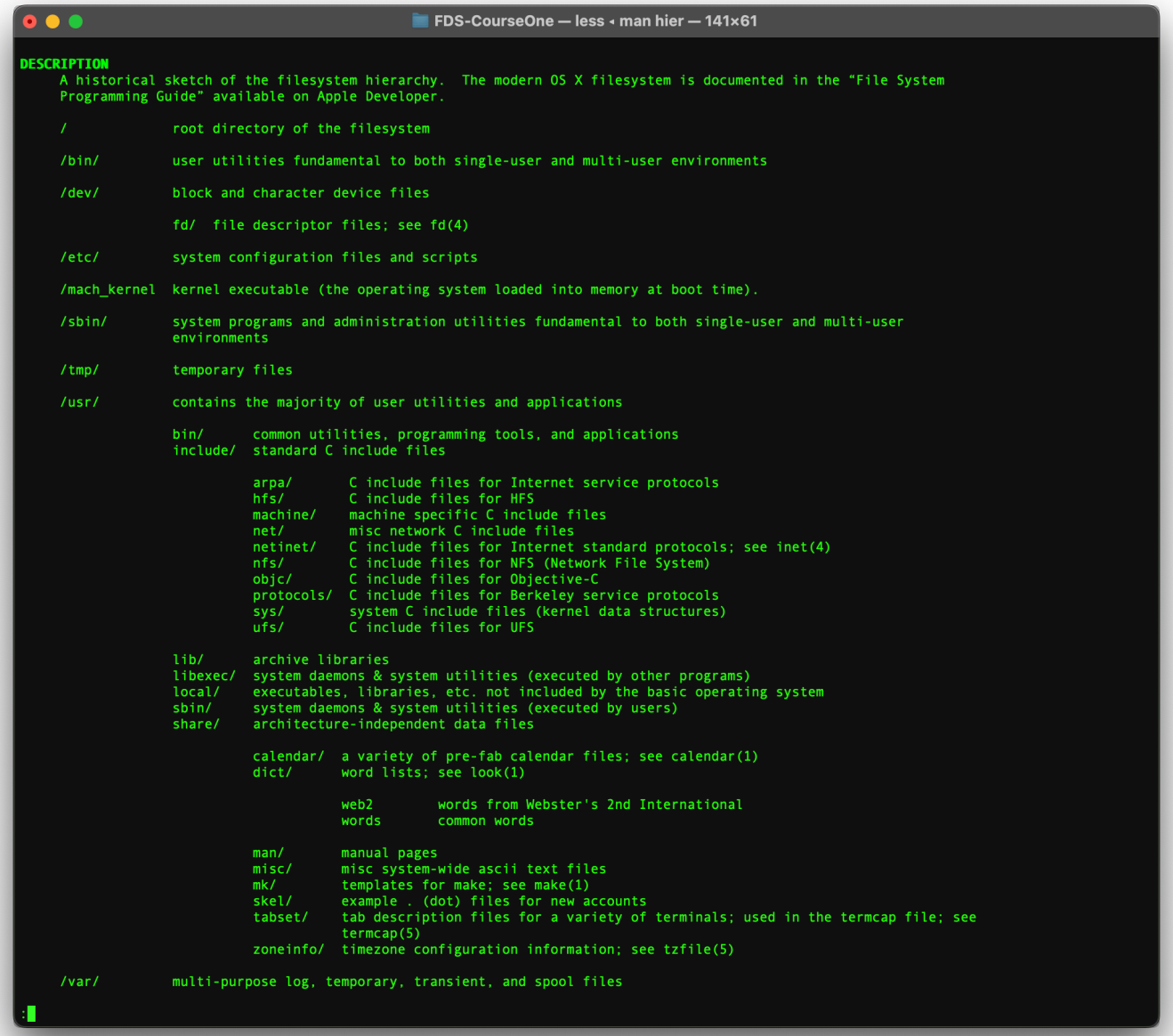
Unix-like operating systems use a File System with one root directory, and every file existing on the system is located under it somewhere. The structure of the UNIX/Linux File system is defined and is called the Filesystem Hierarchy Standard (FHS). The FHS describes the conventions used for the layout of this operative system.

Regardless of the operative system, all files and folders must be addressed by starting in a specific "root" directory. This root directory on UNIX/Linux systems is indicated with the symbol `/`. The "root" or `/` is where all the files originate. All files and folders are organized under the `/` directory. The image below shows the FHS organized in a visual fashion, `/` is where the Filesystem Hierarchy starts.

Wikipedia has a nice, longer article on the FHS.

Apple's macOS File system

Apple's macOS is a Linux-based system, as such the FHS also exists on macOS. The command `man hier` typed in a Terminal on an Apple macOS computer should show something that looks like the following image. The image shows the critical top-level directories living under your `root` or `/` directory.



The MacOS simplifies the UNIX FHS for the common users. So, in MacOS, the visible directory structure (the one any user can easily have access to from the GUI) is slightly different than the Unix/Linux Filesystem Hierarchy. The table below is an extract from an Article by Apple that covers this more fully.

The table below shows the top-level directories commonly accessed by a user using the GUI. This file system structure hides the underlying, more complex Linux FHS that still exists on macOS. You can navigate the complete filesystem from a terminal using `ls` and `cd`.

Directory	Usage
`/Applications`	The directory where apps and programs the users utilize are installed. The directory is intended for use by all users of a computer (if you have multiple users they will all see, read, and write the programs in this folder. The App Store installs apps purchased by the user in this directory automatically. The directory contains a subdirectory called `Utilities`. This folder contains apps and programs that are intended for use not by the user but by the operative system.
`/Library`	This directory contains components of apps and programs not directly accessed by the user but needed by the Apps or programs that use the `Library` directory to store app/program-specific (or system-specific) resources. Each user on a computer will have a distinct `Library` folder.
`/Network`	This directory will show a list of computers visible when the computer is connected to a network.
`/System`	This directory contains the system resources required by Mac OSX to run. These resources are provided by Apple and must not be modified.
`/Users`	This directory contains one or more user home directories. The user home directory is where user-related files are stored. A typical user's home directory includes the following subdirectories: • `Applications`—Contains user-specific apps. • `Desktop`—Contains the items on the user's desktop. • `Documents`—Contains user documents and files. • `Downloads`—Contains files downloaded from the Internet. • `Library`—Contains user-specific app files (hidden in Mac OS 10.7 and later). • `Movies`—Contains the user's video files. • `Music`—Contains the user's music files. • `Pictures`—Contains the user's photos. • `Public`—Contains content the user wants to share. • `Sites`—Contains web pages used by the user's personal site. The preceding directories are for storing user documents and media only. Apps and programs do not write files to the preceding directories unless explicitly directed to do so by the user. The sole exception to this rule is the Library directory, which apps may use to store data files needed to support the current user. Of the subdirectories, only the `Public` directory is accessible by other users on the system. Access to the other directories is restricted by default.

Microsoft's Windows File System

Microsoft's Windows File System

Microsoft Windows (this short reference is relevant for Windows 10, see the Original Wikipedia Article here) uses a file system structure that does not follow the Linux Hierarchy. In most Windows 10 installations, the following folders may appear.

Folder	Description
`\PerfLogs`	May hold Windows performance logs, but on a default configuration, it is empty.
`\Program Files`	• `32-bit architecture`: All programs (both 16-bit and 32-bit) are installed in this folder. • `64-bit architecture`: 64-bit programs are installed in this folder.
`\Program Files (x86)`	Appears on 64-bit editions of Windows. 32-bit and 16-bit programs are by default installed in this folder, even though 16-bit programs do not run on 64-bit Windows.
`\ProgramData` (hidden)	Contains program data that is expected to be accessed by computer programs regardless of the user account in the context of which they run. For example, a program may store specific information needed to operate DVD recorders or image scanners connected to a computer, because all users use them. Windows itself uses this folder. For example, Windows Defender stores its virus definitions in `\ProgramData\Microsoft\Windows Defender`. Programs do not have permission to store files in this folder, but have permission to create subfolders and store files in them. The organization of the files is at the discretion of the developer.
`\Users`	User profile folders. This folder contains one subfolder for each user who has logged onto the system at least once. In addition, it has two other folders: `Public` and `Default` (hidden).
`\Public`	This folder serves as a buffer for users of a computer to share files. By default, this folder is accessible to all users who can log on to the computer. Also, by default, this folder is shared over the network, although anonymous access (i.e. without a valid password-protected user account) to it is denied. This folder contains user data, not program data, meaning that users are expected to be the sole decider of what is in this folder and how it is organized. It is unethical for a program to store its proprietary data here. (There are other folders dedicated to program data.)
`[username]\AppData` (hidden)	This folder stores per-user application data and settings. The folder contains three subfolders: Roaming, Local, and LocalLow. Roaming is for networked-based logins for roaming profiles. Data saved in Roaming will synchronize to the computer when the user logs in. Local and LocalLow do not sync up with networked computers.

`\Windows`	Windows itself is installed in this folder. • `\System` - stores 16-bit DLLs and is normally empty on 64-bit editions of Windows. • `\System32` - stores either 32-bit or 64-bit DLL files, depending on whether the Windows edition is 32-bit or 64-bit. • `\SysWOW64` - Only appears on 64-bit editions of Windows and stores 32-bit DLLs. These folders store dynamic-link library (DLL) files that implement the core features of Windows and Windows API.
`\WinSxS`	This folder is officially called "Windows component store" and constitutes the majority of Windows. A copy of all Windows components, as well as all Windows updates and service packs is stored in this folder.