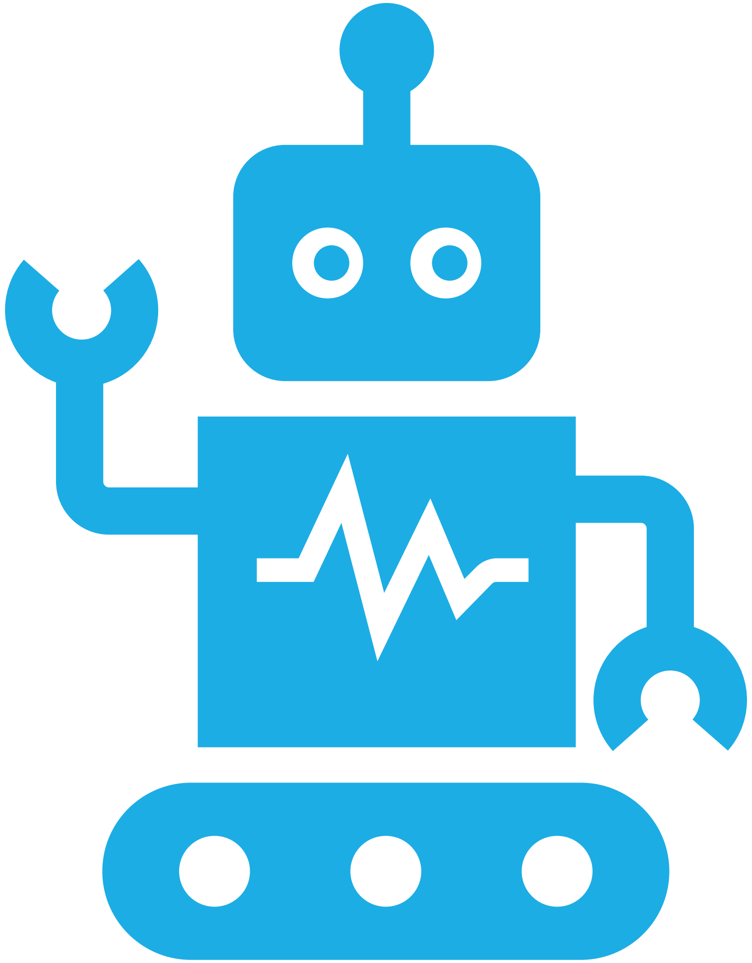
COMMAND LINE INTERFACE

How to impress anyone ever who will watch you interact with a computer for the rest of your life.



THE CLI (COMMAND LINE INTERFACE)

Chances are as a normal computer user, you have been used to interacting with your computer using your mouse and keyboard for almost everything. If you want to open a program, you will either click on the shortcut or you will search for by typing the name of the program.

You have been interacting with your operating systems GUI (graphical user interface). For most users, this interface is plenty. Developers prefer a more efficient and powerful interface interacting with their computer.

To do this, we interact with the CLI. To understand the CLI, we first need to understand the goal of both a CLI and a GUI (really, and interface in general).

INTERFACES

An interface is something designed to help people interact with an underlying system. In our case, the underlying system we want to interact with is our computers operating system. In turn, our operating system wants to interact with our computers hardware.

To give a concrete example related to web development, your internet browser is simply an interface designed to help you interact with the internet. It gives you an easy to learn application that does very complicated interactions under the hood that you don't need to worry about.

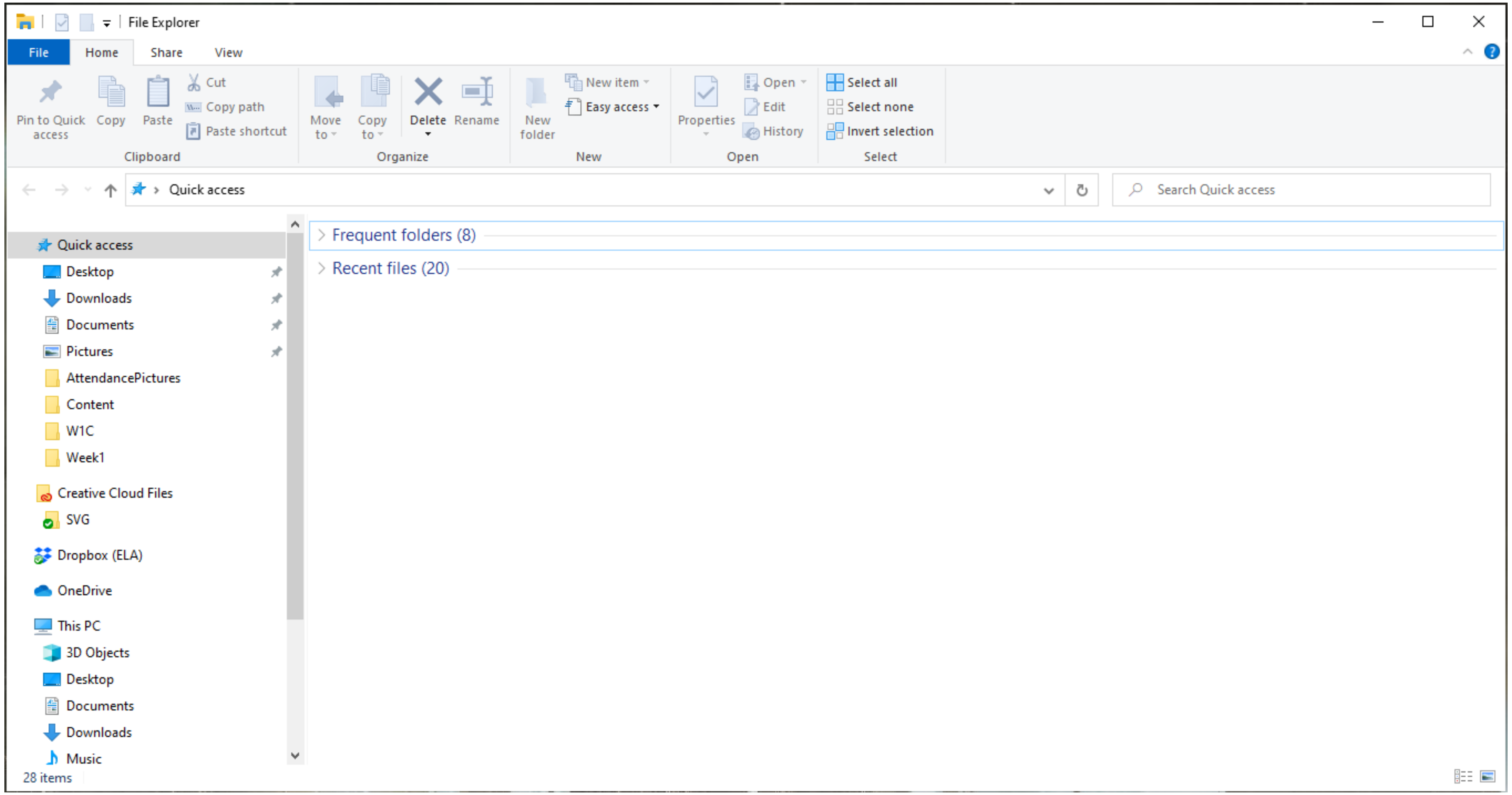
So on your computer, your GUI makes interacting with your operating system and achieving goals very simple. Most of the time you click what you need, maybe type a couple options and you're done.

A CLI is just another way to get the same things done on your computer, but instead of the interactions being moving the mouse and clicking, we are restricted to only typing commands.

GUI VS CLI

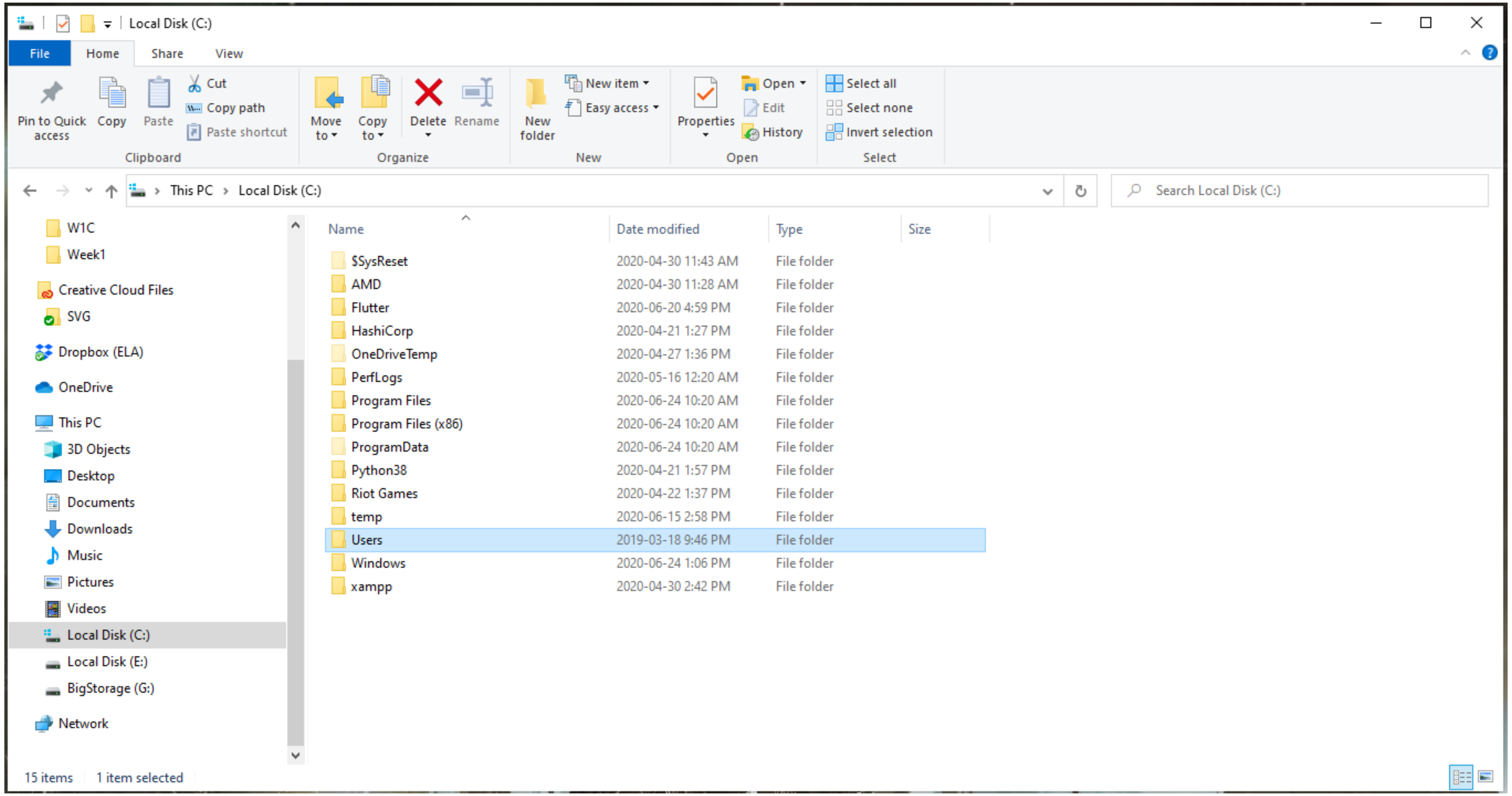
Let’s take a look at a real world comparison of trying to achieve the same goal of creating a folder in your home directory called “test” and then create a file inside that folder called “index.html”.

First we will look at the steps taken on the GUI.



GUI

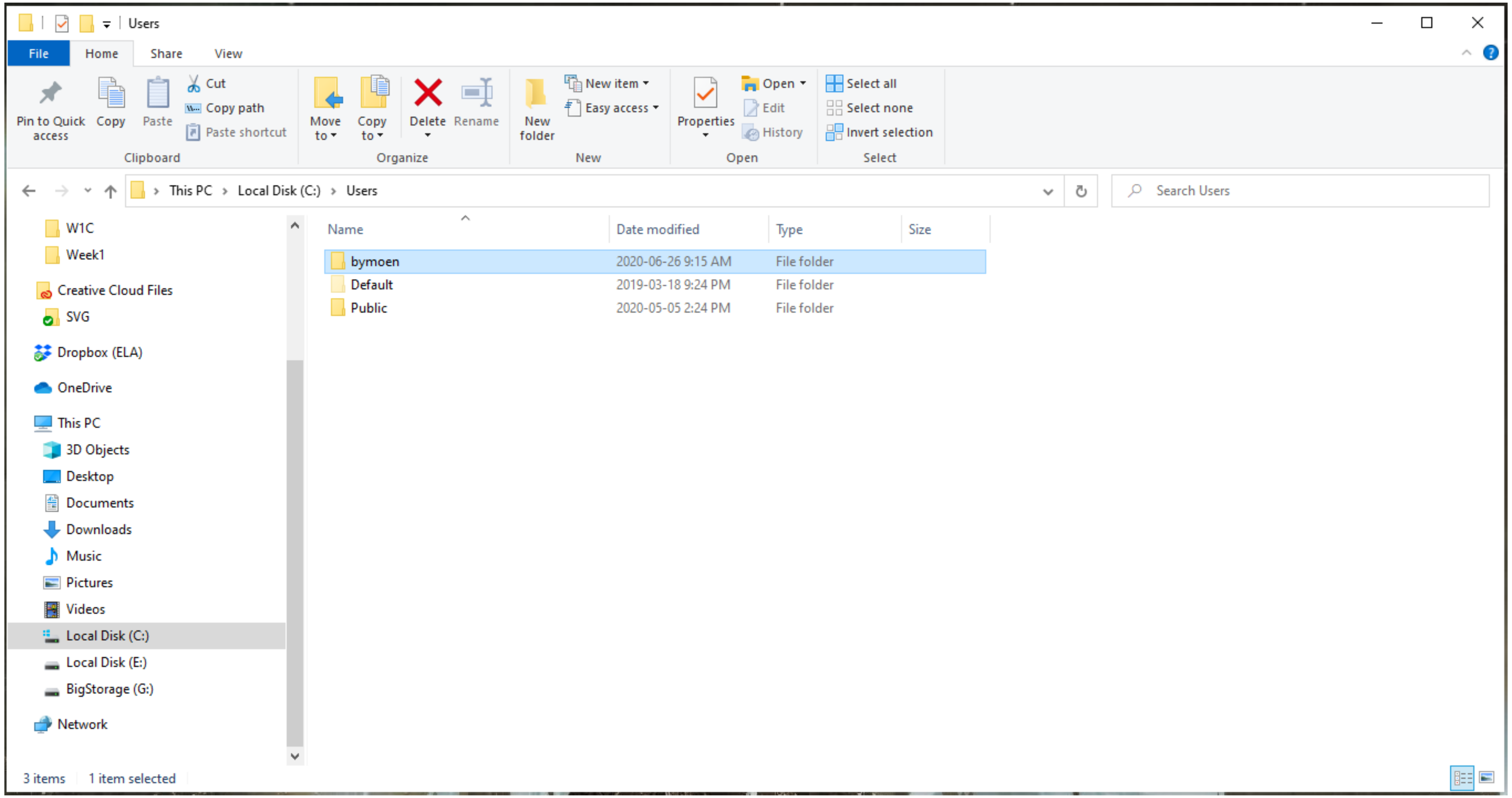
First, we open what ever application on our OS handles file exploration.



Name	Date modified	Type	Size
$SysReset	2020-04-30 11:43 AM	File folder	
AMD	2020-04-30 11:28 AM	File folder	
Flutter	2020-06-20 4:59 PM	File folder	
HashiCorp	2020-04-21 1:27 PM	File folder	
OneDriveTemp	2020-04-27 1:36 PM	File folder	
PerfLogs	2020-05-16 12:20 AM	File folder	
Program Files	2020-06-24 10:20 AM	File folder	
Program Files (x86)	2020-06-24 10:20 AM	File folder	
ProgramData	2020-06-24 10:20 AM	File folder	
Python38	2020-04-21 1:57 PM	File folder	
Riot Games	2020-04-22 1:37 PM	File folder	
temp	2020-06-15 2:58 PM	File folder	
Users	2019-03-18 9:46 PM	File folder	
Windows	2020-06-24 1:06 PM	File folder	
xampp	2020-04-30 2:42 PM	File folder	

GUI

We then must find our users home directory (this is in a different place for Mac and Windows, but the idea is the same).



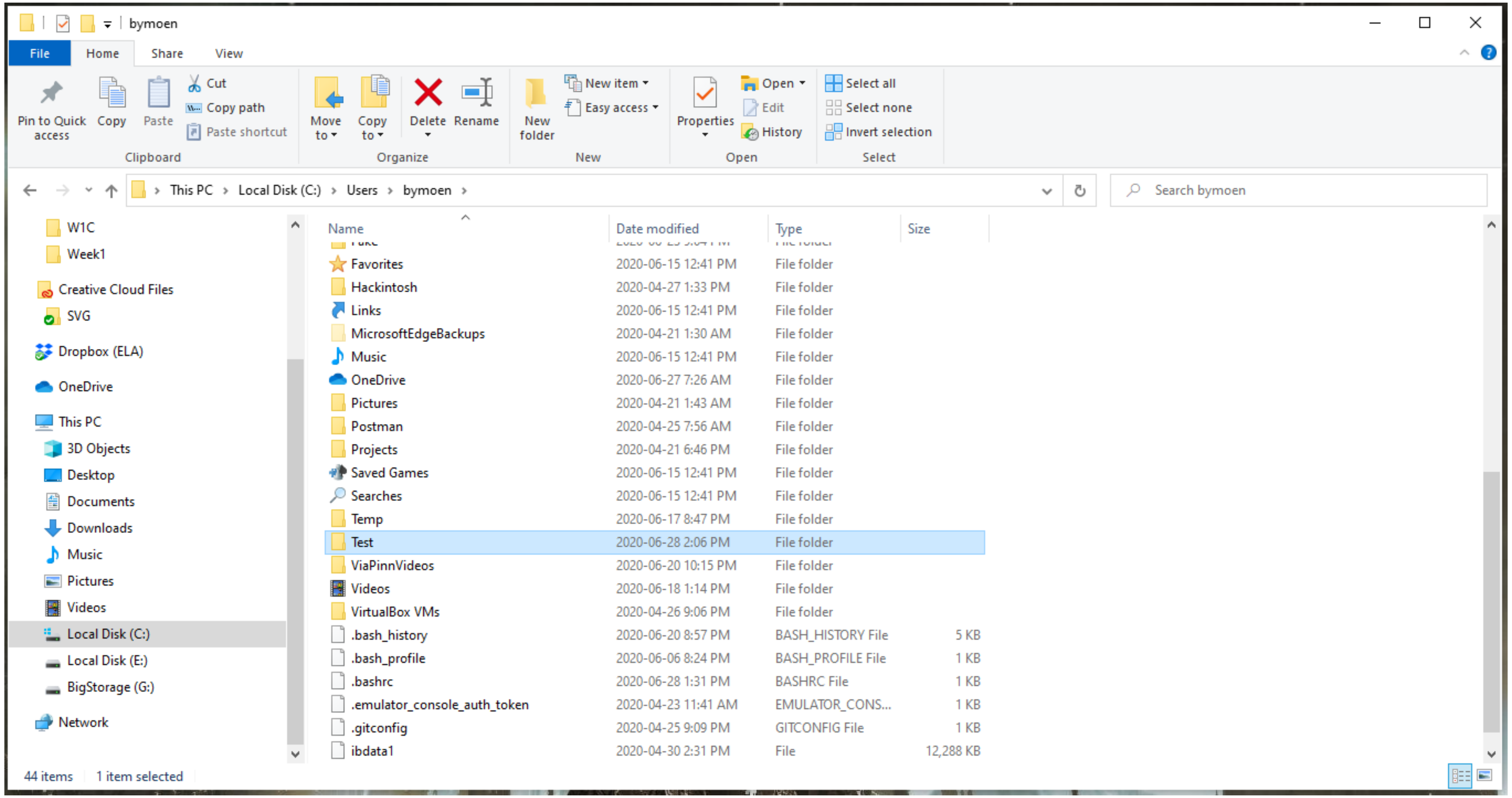
This PC › Local Disk (C:) › Users

Name	Date modified	Type	Size
bymoen	2020-06-26 9:15 AM	File folder	
Default	2019-03-18 9:24 PM	File folder	
Public	2020-05-05 2:24 PM	File folder	

3 items 1 item selected

GUI

Continue clicking through the folders until the home directory is found.

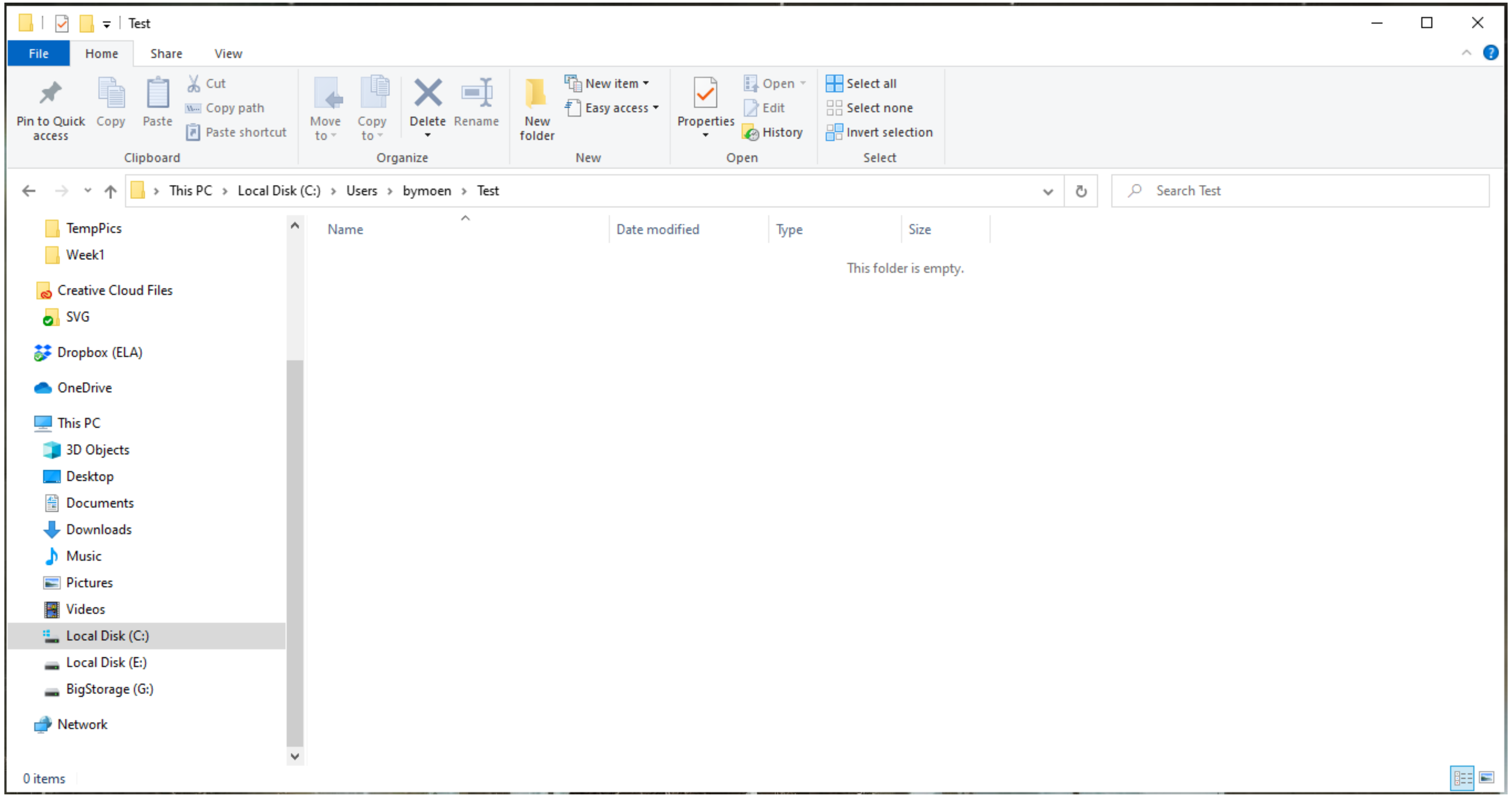


Name	Date modified	Type	Size
Favorites	2020-06-15 12:41 PM	File folder	
Hackintosh	2020-04-27 1:33 PM	File folder	
Links	2020-06-15 12:41 PM	File folder	
MicrosoftEdgeBackups	2020-04-21 1:30 AM	File folder	
Music	2020-06-15 12:41 PM	File folder	
OneDrive	2020-06-27 7:26 AM	File folder	
Pictures	2020-04-21 1:43 AM	File folder	
Postman	2020-04-25 7:56 AM	File folder	
Projects	2020-04-21 6:46 PM	File folder	
Saved Games	2020-06-15 12:41 PM	File folder	
Searches	2020-06-15 12:41 PM	File folder	
Temp	2020-06-17 8:47 PM	File folder	
Test	2020-06-28 2:06 PM	File folder	
ViaPinnVideos	2020-06-20 10:15 PM	File folder	
Videos	2020-06-18 1:14 PM	File folder	
VirtualBox VMs	2020-04-26 9:06 PM	File folder	
.bash_history	2020-06-20 8:57 PM	BASH_HISTORY File	5 KB
.bash_profile	2020-06-06 8:24 PM	BASH_PROFILE File	1 KB
.bashrc	2020-06-28 1:31 PM	BASHRC File	1 KB
.emulator_console_auth_token	2020-04-23 11:41 AM	EMULATOR_CONS...	1 KB
.gitconfig	2020-04-25 9:09 PM	GITCONFIG File	1 KB
ibdata1	2020-04-30 2:31 PM	File	12,288 KB

44 items   1 item selected

GUI

We then make a new folder named “Test”

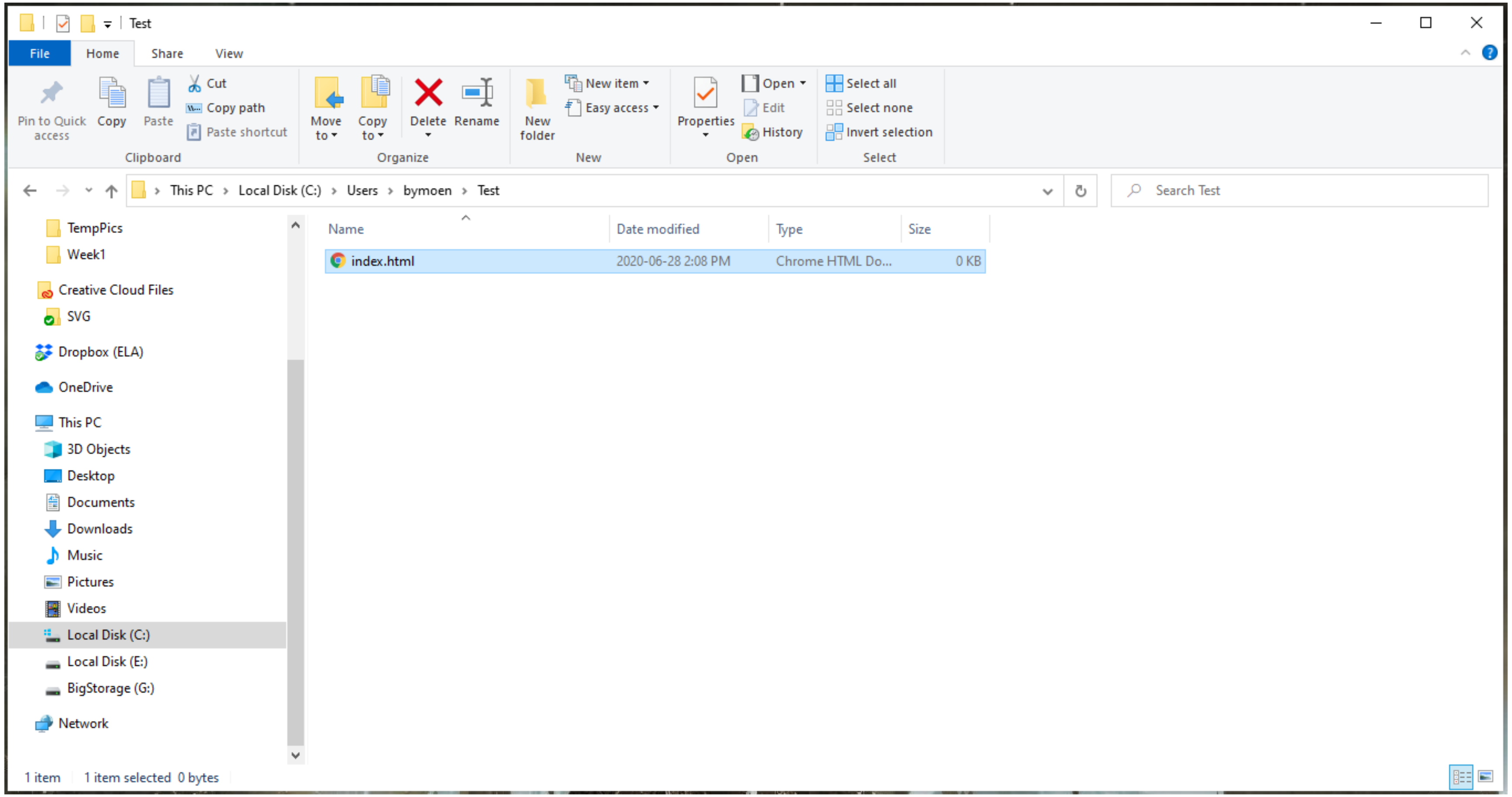


This folder is empty.

0 items

GUI

Click on the new folder to enter it.



Name	Date modified	Type	Size
index.html	2020-06-28 2:08 PM	Chrome HTML Do...	0 KB

GUI

Finally we create our index.html file

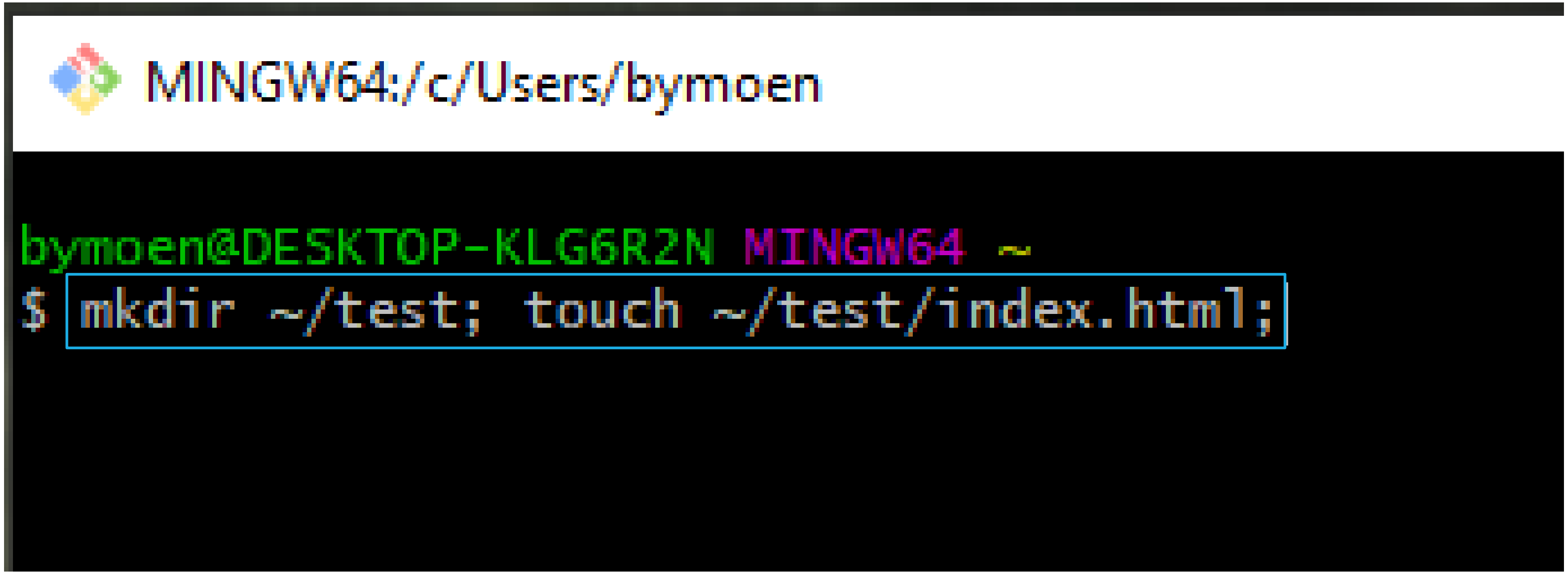
GUI RECAP

So we can see that interacting with the GUI has many clicking steps that are totally unavoidable. This is where we will see the power of the CLI. There are many shortcuts available that make our lives easier (once you learn the commands).

CLI

With the CLI, instead of clicking around on our desktop with our mouse we type commands into a **terminal emulator** running a **shell** (don’t worry these terms will be explained later).

To make this more concrete, let’s take a look at doing the exact same thing we did in the GUI but instead in the CLI.



```
bymoen@DESKTOP-KLG6R2N MINGW64 ~
$ mkdir ~/test; touch ~/test/index.html;
```

CLI

Everything inside the blue box is what is typed in the terminal. That's it, one step and done.

CLI RECAP

Although as of right now we don’t know what those typed commands are doing, we can see the major difference between a GUI (clicking around) and the CLI (typing commands).

I simply typed **mkdir ~/test; touch ~/test/index.html;** and the folder and file were both created.

Both interfaces achieved the same thing in creating a folder and a file inside that folder. The way that we achieve the goal is the only thing that changes.

HOW DOES IT WORK

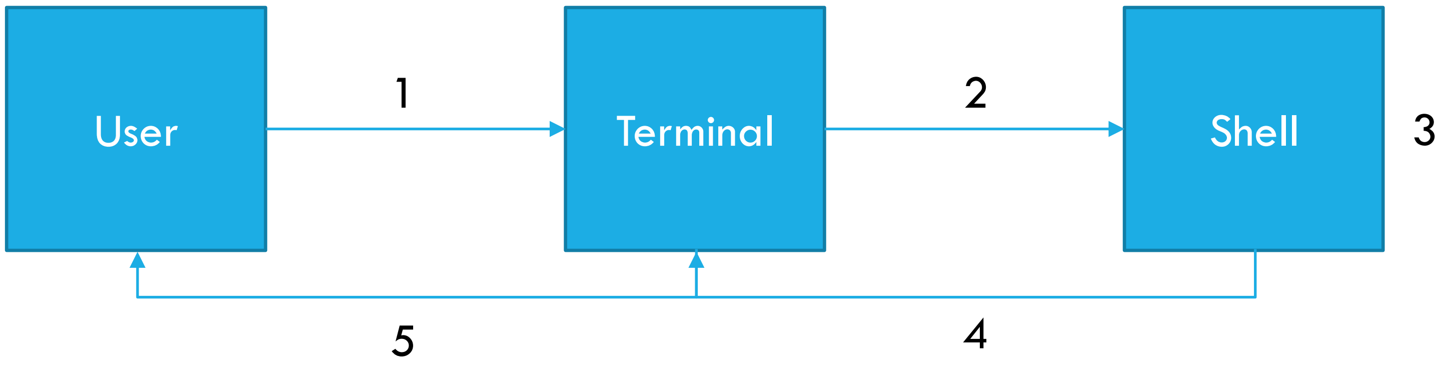
The CLI is one of the most misunderstood topics in the industry. It is very common for developers to have decades of experience as software engineers and have almost no experience using a CLI.

Let’s get an understanding of what is needed to interact with a CLI.

1. The first thing you need is a **terminal emulator**. The terminal emulator (usually just called a terminal or a console) is responsible for:
   1. Allowing you to type commands
   2. Showing you the output of your commands
   3. Initially launching you into a **shell**
2. Second you need a **shell.** The shell is whatever program that sits between you and the operating system. You interact with the shell, the shell takes your commands and executes them.
   - The shell you are running determines what kinds of commands you are able to run
   - There are many different kinds of shells available to us.

HOW DOES IT WORK CONT.

Let's take a look at a graph of what is going on.



If this seems complicated to you, don't worry. In fact this is just as complicated as how your GUI works.

1. The user types a command into the terminal
2. The terminal simply passes that command to the underlying shell
3. The shell runs the command (could be as simple creating a file or as complicated as formatting a storage drive)
4. If there is any, the shell will pass any messages or errors from the command being ran back to the terminal
5. The terminal shows displays these messages or errors to the user

GETTING UP & RUNNING

Depending on the operating system your computer is running, you will either have to install a nice terminal emulator and shell or do nothing.

Mac and Linux users don't have to do anything. Your OS comes with an okay terminal and already has a few different shells installed to use.

Windows users your OS comes with a simple terminal and the **cmd** shell which we will not be using. Instead we will be installing **git bash** which is already installed.

For both OS's, you can actually use a terminal inside your VSCode, you just have to pull it up from the bottom of the window.

We now all have the software required to effectively interact with a CLI and an understanding of how this software is operating under the hood.

All that is required now is for us to learn the commands to type into our terminal to make it do our bidding!