Finding the Terminal application on a Mac is quite straightforward. Here's a step-by-step guide:

1. **Using Spotlight Search:**
   * Press **Command (⌘)** + **Spacebar** to open Spotlight Search.
   * Type "Terminal" in the search bar.
   * Click on the Terminal application from the search results.
2. **Through the Finder:**
   * Open a Finder window. (The little spot glass at the top right)
   * Go to the "Applications" folder.
   * Inside "Applications," open the "Utilities" folder.
   * Find and double-click on the Terminal application.

**Type PWD (The map)**

Think of pwd in the terminal like asking "Where am I?" in a huge library. Just like you might want to know which section or aisle you're in, pwd tells you where you are in the computer's file system. It's like looking at a map that shows you exactly where you're standing.

**LS PWD (The list)**

The `ls` command in the terminal is like looking around a room to see what's there. When you type `ls` and press enter, it shows you a list of all the files and folders that are in the current directory (or "room") you are in on your computer. It's a quick way to see what's stored in that part of your computer.

**cd (Moving around)**

The **cd** command in the terminal is like moving from one room to another in a house. When you type **cd** followed by the name of a directory, it takes you into that directory. Just like walking into a different room, **cd** changes your current location in the computer's file system to the specified directory. (Use this to change from folder to folder)

**git init (Start a new git repository to keep track of changes)**

`git init` is a command used in the terminal to start a new Git repository. It's like creating a new project folder where you can keep track of all the changes made to the files in that folder. When you run `git init`, it sets up a hidden .git directory within the folder you're in. This .git directory is where Git stores all the information it needs to keep track of the changes you make to your project.

**Top of Form**

**Bottom of Form**

**Configure Your Name and Email:**  
In Terminal, type the following command to set your name for every Git commit:

git config --global user.name "Your Name"

Replace "Your Name" with your Github username  
  
**Adding a folder to add your content in and so you can push to git.**

1. Make a folder name it whatever you like
2. Bottom left, use the “finder”,
3. Open up documents
4. Click the title “Documents” at the top bar and then right click (two fingers)
5. Change it to the folder that is named after your user usually it’d be name23 or some number
6. You can check if the folder has been added using the “ls” command

**Git add keep track of changes**

`git add .` is a command used in the terminal when working with Git, a version control system. Think of it as telling Git, "Hey, please keep track of all the changes I've made in this folder."

When you use `git add .`, the `.` represents everything in your current directory. This command tells Git to start tracking any new or changed files in this directory. It's like gathering all your work and getting it ready to be permanently recorded in your project's history. However, at this stage, the changes are not yet recorded; they are just staged and ready for the next step, which is committing them with `git commit`.

**Save your changes!**

**git commit -m** is a command used in the terminal with Git, which is a system for tracking changes in computer files. When you use this command, you're basically saving the changes you've made to your files.

Here's a breakdown:

* **git commit**: This part of the command tells Git you want to save (or "commit") your changes.
* **-m**: This stands for "message." It's a way of adding a note to your commit so you (or others) can understand what changes were made and why.
* After **-m**, you write your message in quotes, like **git commit -m "Fixed the bug in the login feature"**.

So, when you use **git commit -m "Your message here"**, you're not only saving your changes, but also attaching a note to them for future reference. It's like putting a label on a file in a folder, describing what's inside or what you did to it.

Go to your repository the one you want to track your changes in. This is found at the github website.

**git remote add origin [URL]** is like setting up that shared online space for your project folder:

**git remote add origin [URL]** <https://github.com/Sleepyashofsloth/lavendartown>

**git remote add origin** <https://github.com/Sleepyashofsloth/lavendartown>

The **git commit --all** command is a variation of the standard **git commit** command used in Git, a version control system. It's particularly useful for making your workflow more efficient. Here's how it works:

* **git commit**: This is the basic command used to save changes you've made to your tracked files in Git.
* **--all** (or **-a** for short): This option tells Git to automatically stage any changes to files that it's already tracking (i.e., files that you have added with **git add** in the past) before doing the commit.

In simpler terms, **git commit --all** is like saying, "Hey Git, please take all the changes I made to the files you know about, save them, and note down what I did."

However, there are a couple of things to remember:

* This command doesn't add new files that Git isn't already tracking. For new files, you still need to use **git add**.
* After **--all**, you still use **-m** to add a commit message, like **git commit --all -m "Your message here"**. This message is a brief explanation of what changes you've made.

Using **git commit --all** can help you skip a step if you're updating files that Git is already keeping an eye on, making your work a bit quicker.

1 time verification

Type in your github username, then go to settings, down to developer settings,personal access tokens, classic,generate a classic token, no expiration, select all the squares for scope, then copy and paste that in the green that’s your password, copy it and paste it. Only once you won’t be able to see it so that’s ok.

Next time you want to push your changes to Git, you can follow these simple steps:

1. **Make Your Changes:** First, do whatever changes you need to in your project files.
2. **Check the Status (Optional but Recommended):** Open the terminal, go to your project's directory, and type **git status**. This shows you what changes you've made. It's a good way to double-check what you're about to upload.
3. **Add the Changes:** To tell Git which changes you want to upload, type **git add .** This command stages all your changes for commit. If you only want to stage some changes, you can specify the files instead of using **.**
4. **Commit Your Changes:** Now, save your staged changes with a commit. Type **git commit -m "Your message here"**, replacing **Your message here** with a short description of what you changed. This message helps you and others know what you did in this update.
5. **Pull the Latest Version:** Before you upload (or "push") your changes, it's good practice to first make sure you have the latest version of the project. Type **git pull origin main**. This step reduces the chances of conflicts between your changes and others'.
6. **Push Your Changes:** Finally, upload your changes to GitHub with **git push origin main**. This sends your committed changes to the main branch of your project on GitHub.
7. **Check for Errors:** Watch the terminal's output. If there are no errors, your push was successful. If there's an error, it will tell you what went wrong.