Week 1 + 2

Goals

- [ ] Learn about Kudos!
- [ ] Gain an understanding of some essential developer tools
- [ ] Install git, Node, and VSCode
- [ ] Create and clone your own repository

How will this work?

Weekly session where we will:

- Go over the session's goals
- Code together
- Get support if needed
- Recap

By the end, we will have a finshed project!

Why Kudos?

At Blueprint, recognizing the contributions of our developer members is crucial for fostering a positive and productive project team environment. This tool is specifically designed to highlight the efforts of these members and gather valuable feedback. By acknowledging their hard work, especially amidst their academic and other commitments, we aim to improve their overall project team experience and prevent potential feelings of being overworked or burnt out.

Setup

For any project, it is essential to have certain 'tools' installed...

These tools go by many names:

- libraries
- environments
- packages
- dependencies
- etc.

But they all serve one broad purpose: help developers build.

So, before we begin building our project, lets first install all of the necessary tools we'll need along the way.

Git and GitHub

A key foundation of software development is collaboration.

Across industry, developers collaborate using two essential tools: Git and GitHub

Later on, we will explore these two tools, but for now lets just install them.

Windows install

1. Go to the official Git website at https://git-scm.com/downloads
2. The download for Windows should begin automatically.
3. Once the installer finishes downloading, run the `.exe` file.
4. Follow the prompts in the installation wizard.
5. After installation, open a new terminal and type `git --version` to verify the installation.

macOS install

1. The easiest way to get Git on macOS is in the terminal.
2. Open your Terminal application.
3. Type `git --version` and press enter.
4. If Git is not installed, a pop-up will appear asking you to install the command line developer tools. Click "Install" and follow the on-screen instructions.

Now, lets create and authenticate a GitHub account.

Creating Your GitHub Account

Creating a GitHub account is a simple process.

1. Go to the GitHub Website: Open your web browser and navigate to github.com.
2. Sign Up: On the homepage, you'll see a sign-up form. Enter your **email address**, create a **password**, and choose a **username**.
3. Verification: You'll likely need to solve a quick puzzle to verify you're human. After that, GitHub will send a verification code to the email address you provided. Go to your inbox, find the email, and enter the code on the GitHub site.

That's it! You now have a GitHub account.

Authenticating Your Account

Authentication proves your identity when you want to push (upload) or pull (download) code from your computer. Using your password for this is no longer supported for security reasons. Instead, you should use a **Personal Access Token (PAT)**.

A PAT is like a long, secure password that you use only for accessing GitHub from the command line or apps.

1. Go to Your Settings:
   - Click on your profile picture in the top-right corner and select **Settings**.
   - In the left sidebar, scroll down and click on **Developer settings**.
   - Click on **Personal access tokens**, then select **Tokens (classic)**.
2. Generate a New Token:
   - Click the **Generate new token** button.
   - Set an **Expiration** date. For better security, don't set it to "No expiration". 90 days is fine.
   - Under **Select scopes**, select the `repo` scope.
3. Copy and Save Your Token:
   - Click the **Generate token** button at the bottom.
   - **This is the only time you will see the token!!!** Copy it immediately and save it in a secure place, like a password manager. If you lose it, you'll have to generate a new one.
4. **Using Your PAT**:
   - Now, when you perform a Git operation in your terminal (like `git push`) that requires authentication over HTTPS, you'll be prompted for your username and password.
   - Enter your GitHub **username**.
   - When asked for your password, **paste your Personal Access Token**.

You're all set! Your GitHub account is now created and authenticated for use on your computer.

Node

Like git, Node is a tool in the software development world that is a must-have.

Among many other things, it allows us to see the changes in our code live in our browser!

So as we add components of our app, we can ensure they look and function as we want them to.

Windows

1. Go to the official Node.js website: https://nodejs.org/en/download
2. Download correct verion.
3. Run the installer and follow the on-screen instructions.
4. Once the installation is complete, you can open the terminal and verify the installation by typing:

```
node -v
npm -v
```

These commands should return the version numbers for Node.js and npm, confirming a successful installation.

macOS

1. The easiest way is to download the official `.pkg` installer from the Node.js website: https://nodejs.org/en/download
2. Choose the LTS version and run the installer after it downloads.
3. Follow the installation steps. The installer will guide you through the process.
4. After installation, open a new Terminal window and run the same commands as above to verify: `node -v` and `npm -v`.

Visual Studio Code

We will be writing all of our code using Visual Studio.

Visual Studio has a simple install. Simply use this link and follow the steps: https://code.visualstudio.com/download

Troubleshooting

Before we move on, lets address some common issues that you may run into during setup.

Checking Node Versions (Windows Users)

If you recieve an error when running the `node - v` and `npm -v` commands:

1. Search 'Terminal' in your taskbar start menu.
2. Hover over the Terminal Icon, then right-click.

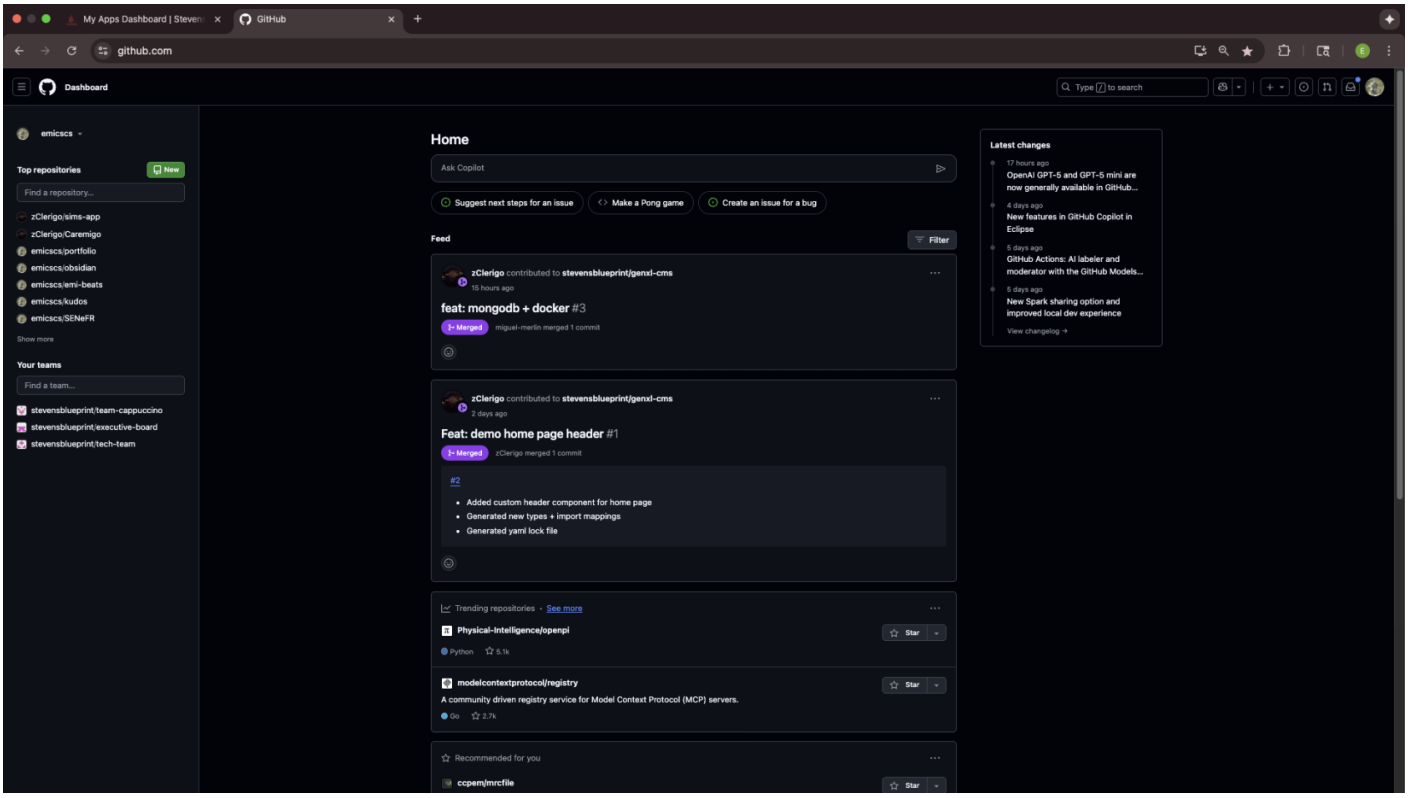
3. A menu should appear. Click on the option to 'Run as Administrator'. This should open a new terminal that indicates you are an Administrator in the title bar.
4. Run this command: `Set-ExecutionPolicy RemoteSigned`.
5. Now, try running `node - v` and `npm -v` once again, and you should be able to see their versions, confirming your Node has installed correctly.

Creating a Repository

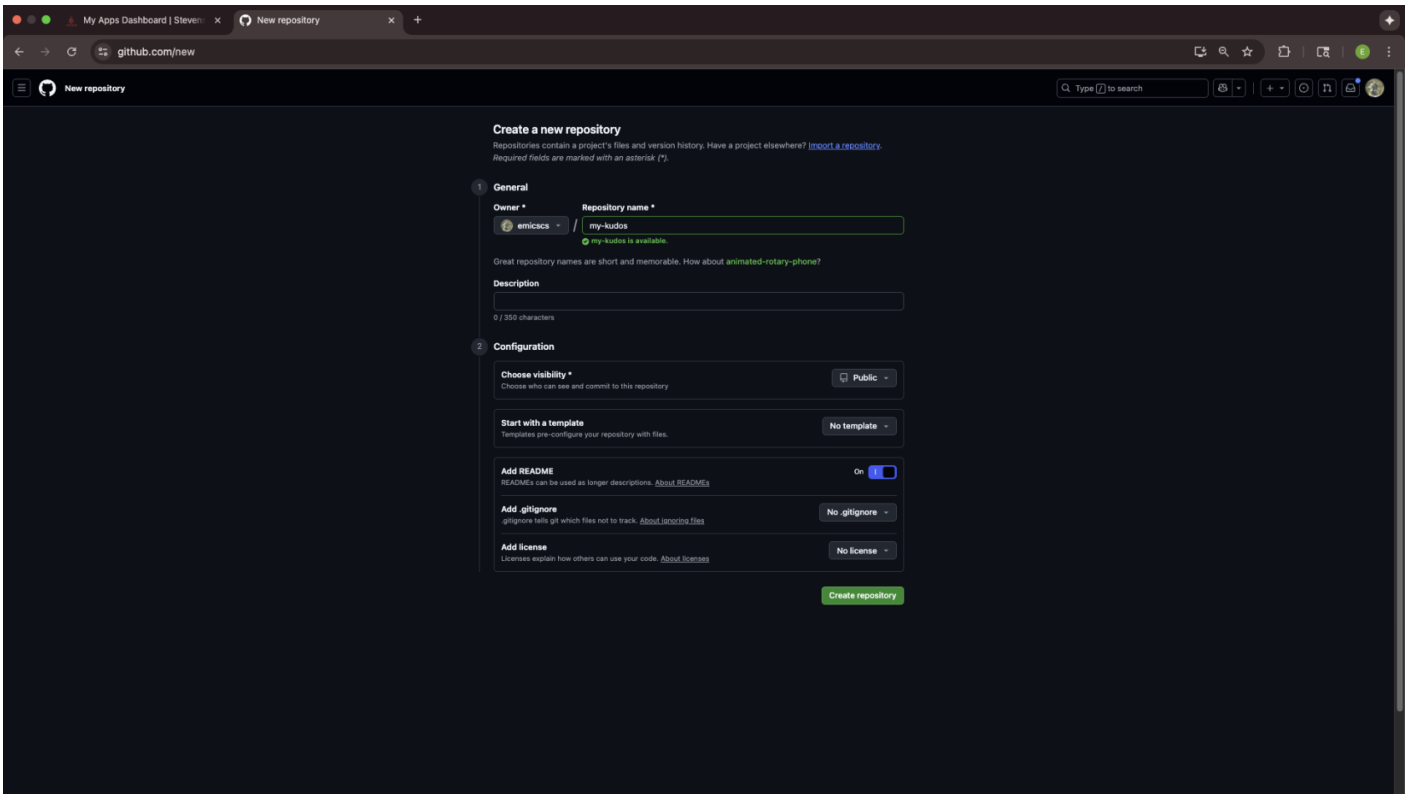
You now know the basics of git and GitHub!

The next step is to create your own repository where you'll store all of your code for the project.

On the main page of GitHub's site, you will see a green box in the top left corner that says 'New'.

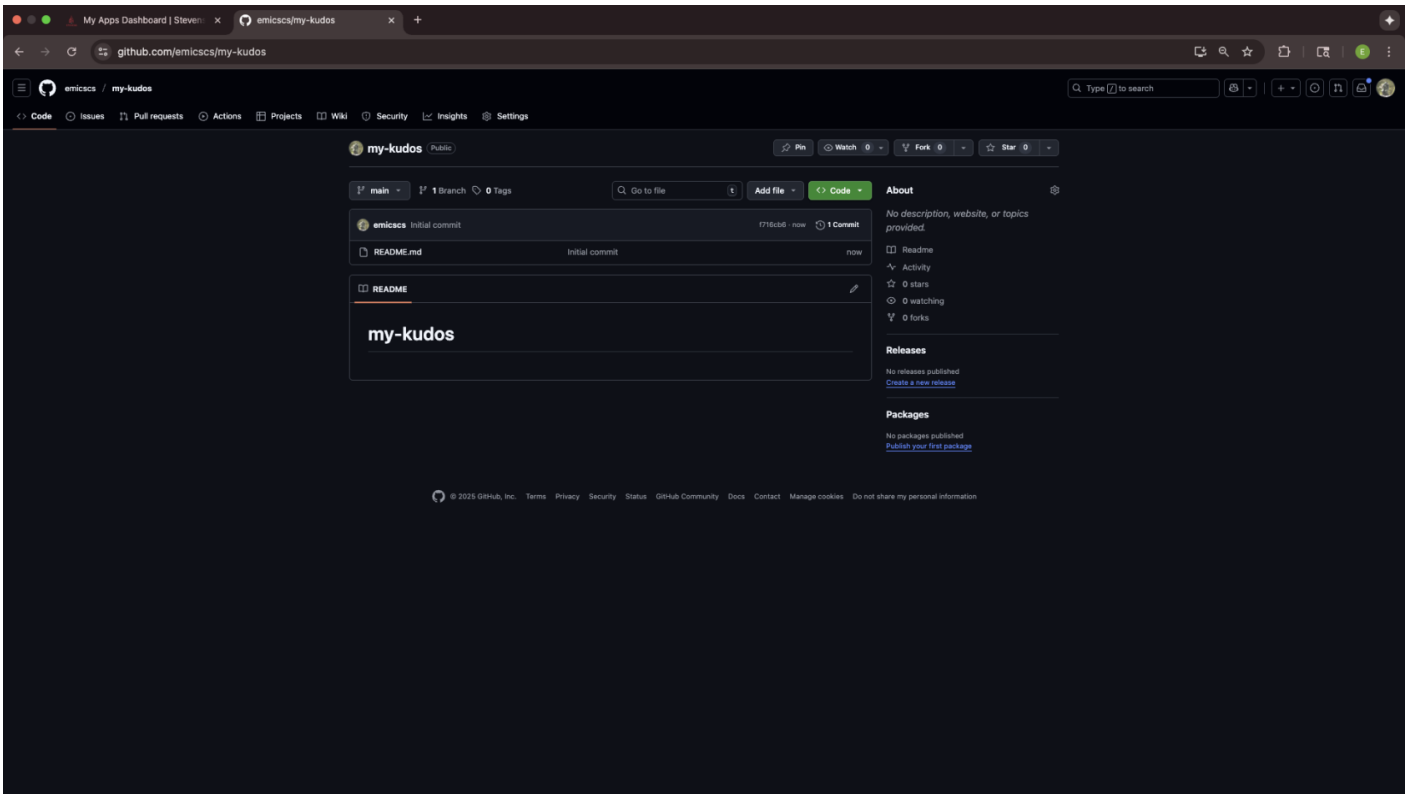


Click it, and you will be redirected to a new page that looks like this:



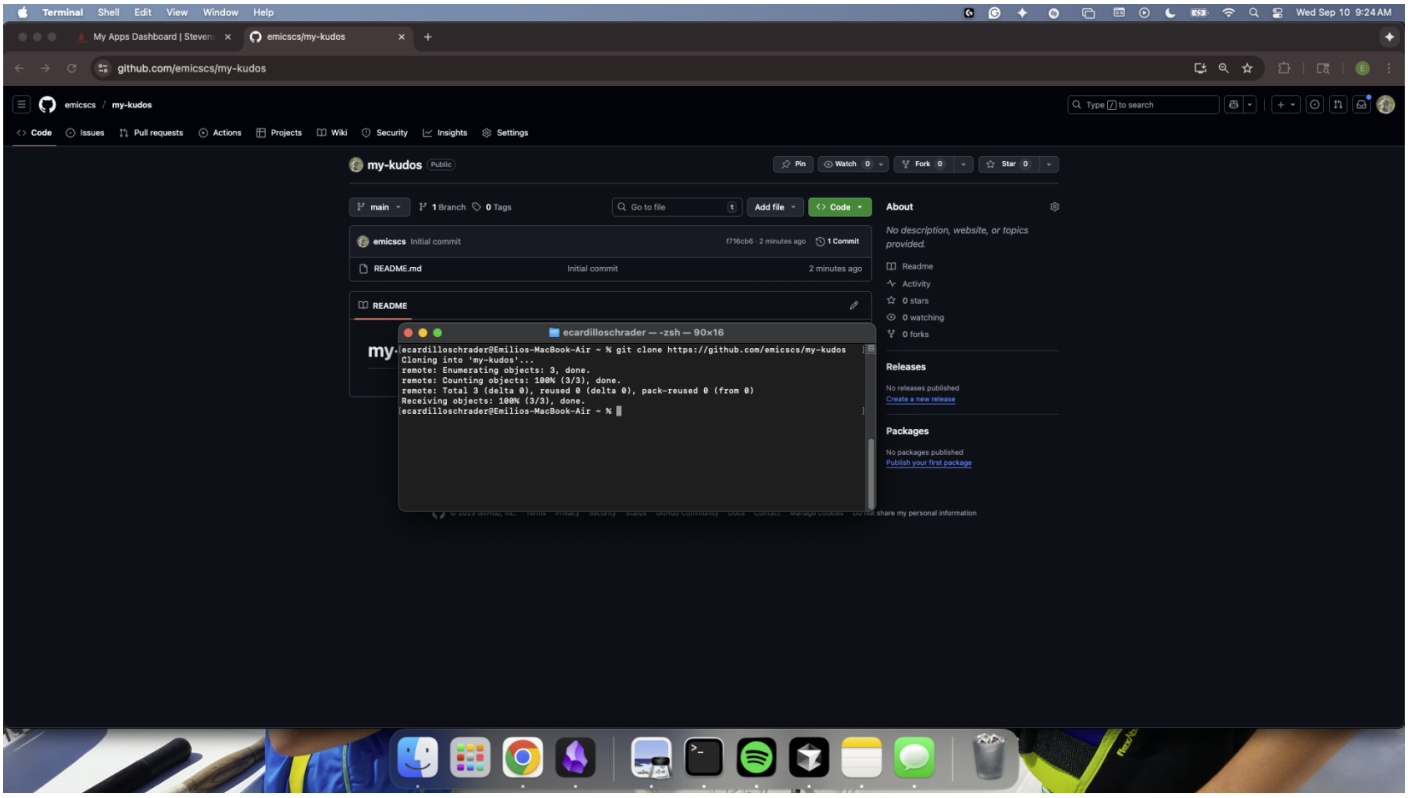
On this page, you will enter the details of your repository. In our case, this includes a specific set of options that oyu should copy from the screenshot above. Your repository name should be `my-kudos`, visibility should be public, and add README option shoud be toggled on.

Click create, and you will again be redirected.

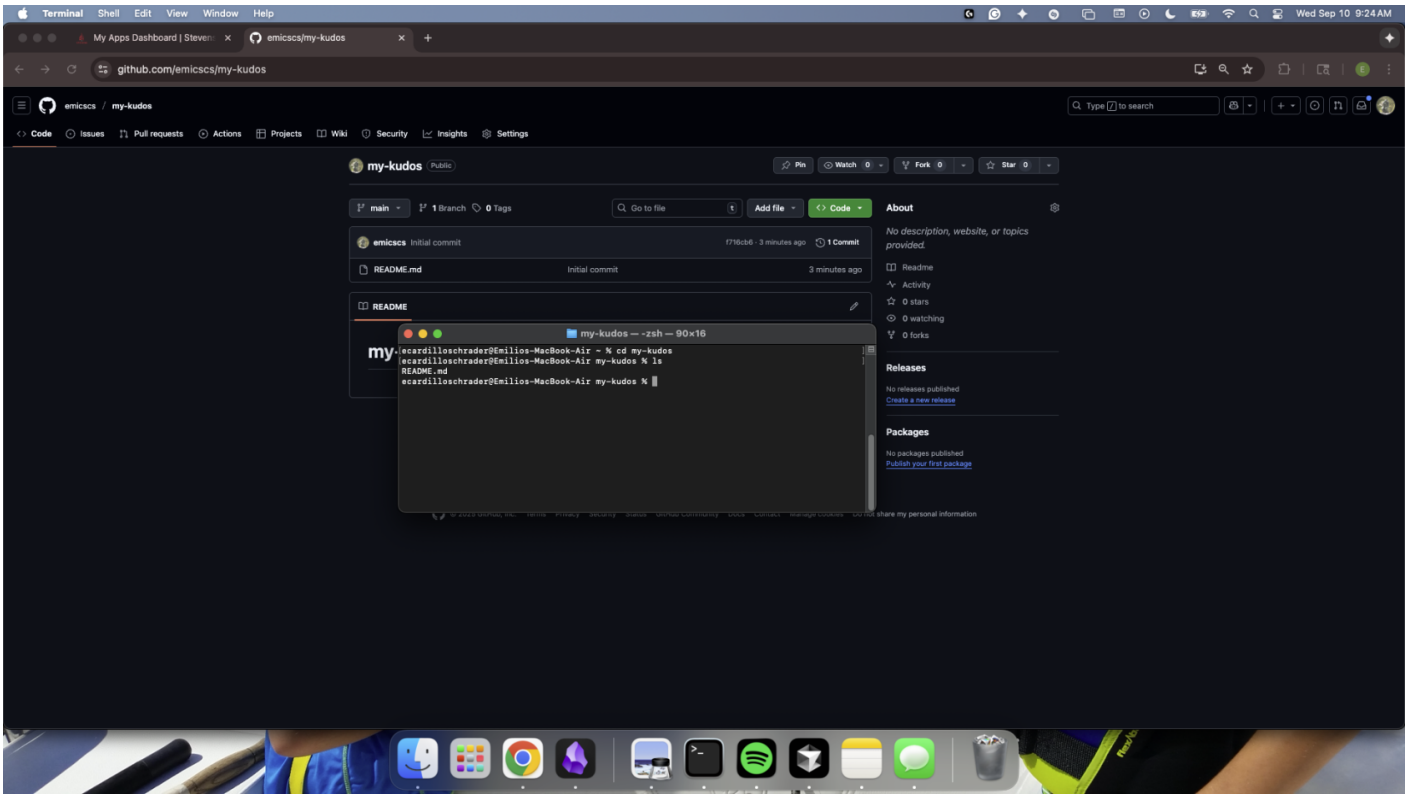


This is your repo! Now, all you have to do is follow some of the steps from the prior section.

The first is cloning your repo with `git clone`.



Like before, to check that the clone was successful, `cd` into your new directory.



Note: In this example, I also used the `ls` command, which lists the contents for your directory. Since we just created our repo and initialized it with a `README.md` file, the only thing in our project folder should be that file - and it is.

Making a Change

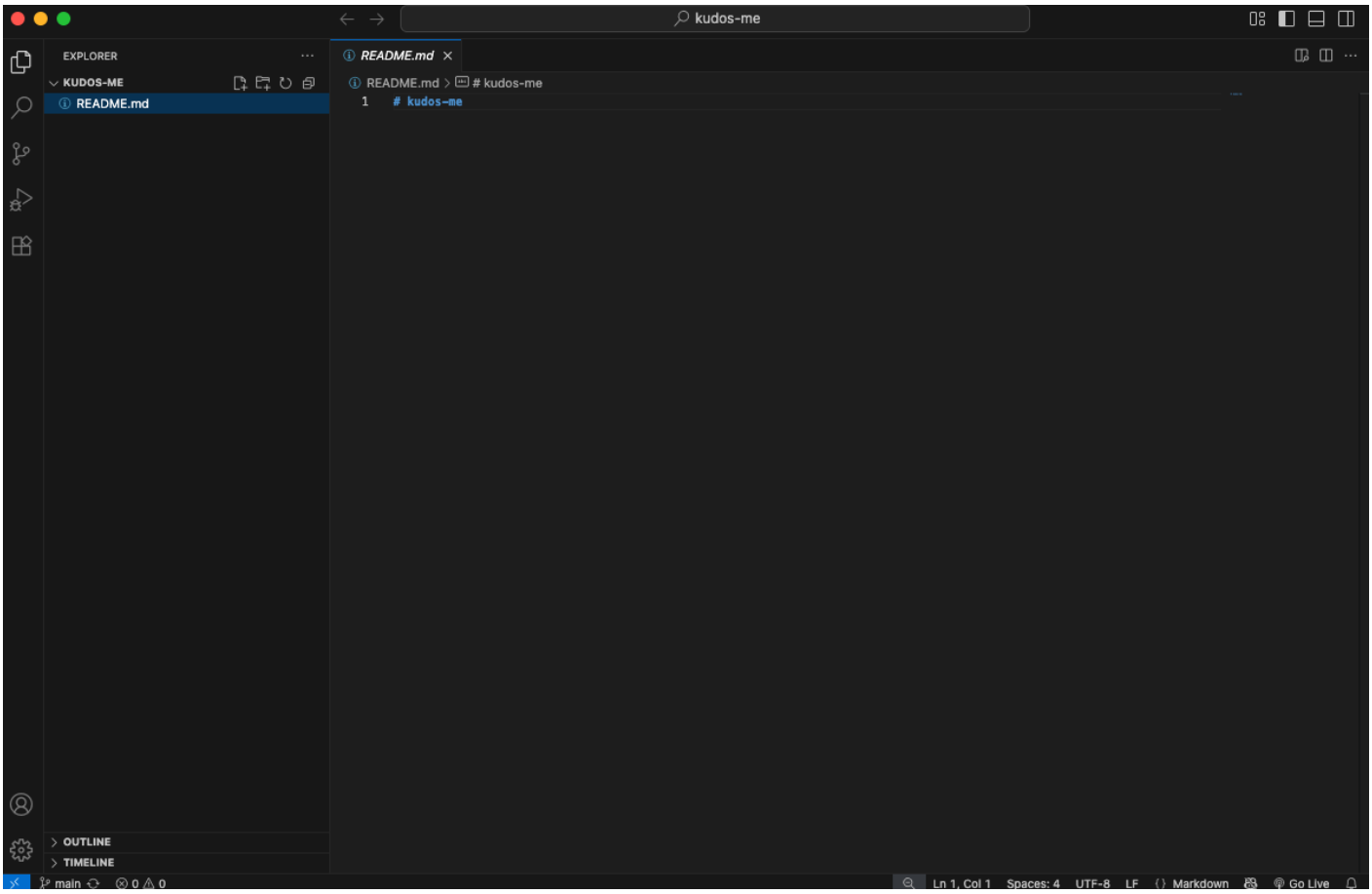
As a way to test what we've done so far, lets try actually making changes to file and seeing them reflected in our repository.

Our goal will be to edit a file in our project - then upload the changes to our GitHub repo using the commands we just learned.

Editing a file

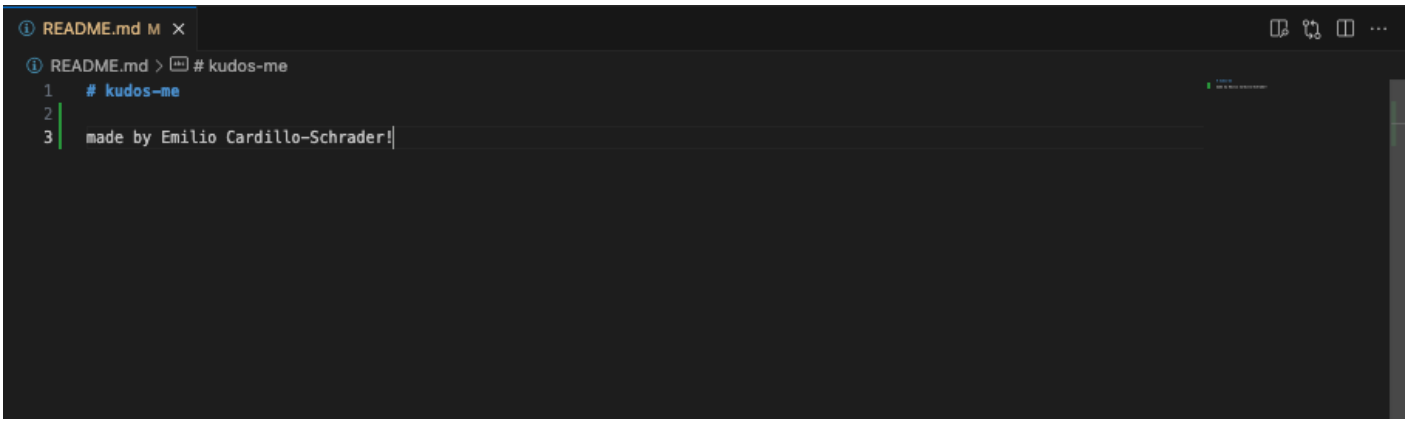
In VSCode, open your `README.md` file.

It should look something like this:



Lets make a simple change: add a short sentence about yourself!

It should look like something like this:



Note: Those green bars next to each of the lines we just wrote indicate that these are new (uncommitted) lines. This is good! It means that we now have code to upload (commit) to our repo!

Making a Commit

Once the change is made, use the `git add` command in order to include or 'stage' our changes for the next commit. Commits can be thought of as checkpoints; more on that later.

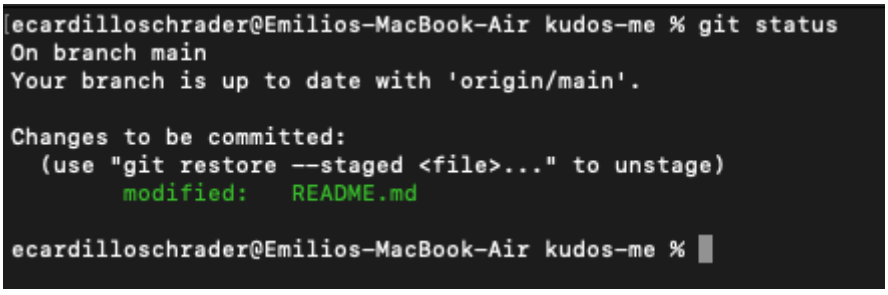
In this case, we made changes to the `README.MD` , so our file path is `/README.MD` and we type in:

```
git add /README.MD
```

Note: You can use `git add` to add as many files as you need, and another common option is to use `git add .` to add all files with changes. Also, The file path is case sensitive.

Checking Changes to be Committed

You can use the `git status` command to check what files have been added or "staged" for a commit to ensure you added the right things. In this case, it's only the one contributors file that we added.



You can see here that our file is under changes to be committed and is in **green** text. Changes not staged for commit will be in **red** (if there are any).

Committing Changes

With our changes made and staged, we can create a **commit** which can be thought of as a checkpoint for our code. We can revert to this point if we make mistakes or need to look at the

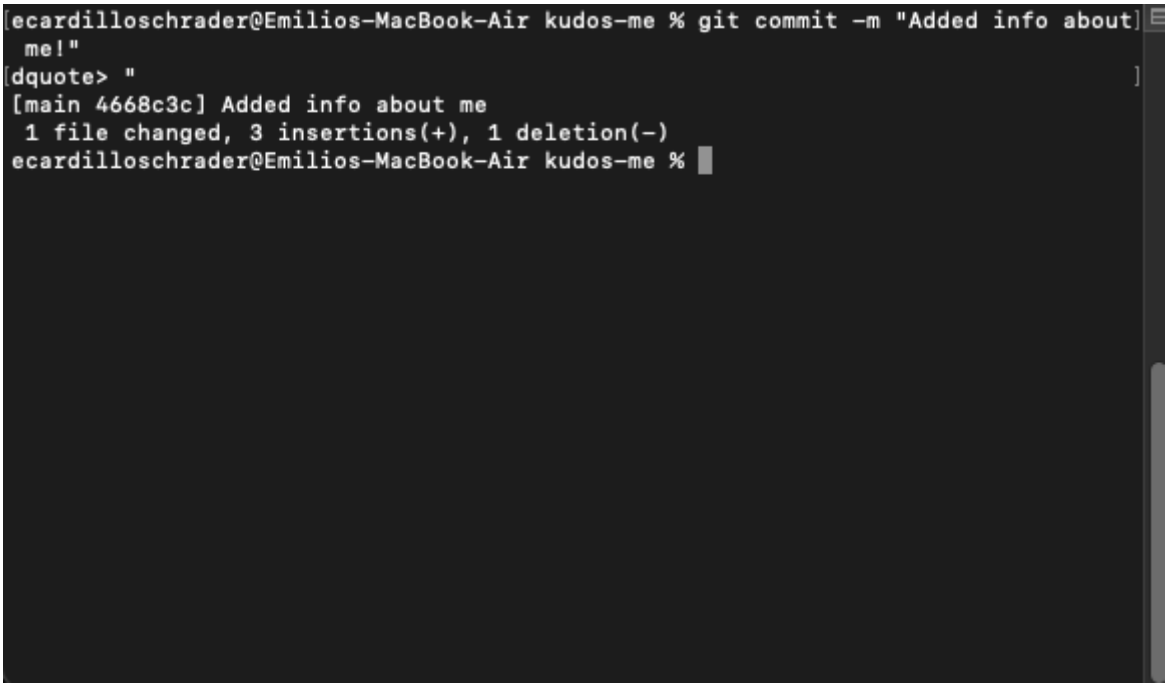
version of the code at this point in time.

It is also good practice to include a commit message by using the `-m` flag to describe what changes were made. For us, we will just say that we are adding a new contributor, like so:

```
git commit -m "Added info about me!"
```

Commit Messages: When writing a commit, it is usually good practice to follow a convention. At Blueprint, we use the Conventional Commits specification: www.conventionalcommits.org, but don't worry about that right now!

We can commit as often as we want, or when we feel it is necessary before making big changes.



Pushing a Commit

In order for our changes to appear on the remote repository (the one on Github, which is online), we need to push our changes using the `git push` command. If we run it as is, however, we will encounter an error:

It is no big deal - the reason it happens is because when we create the branch contributors/johnDoe, we only created it locally and it does not exist on the Github repository.

Therefore, we must run the command shown:

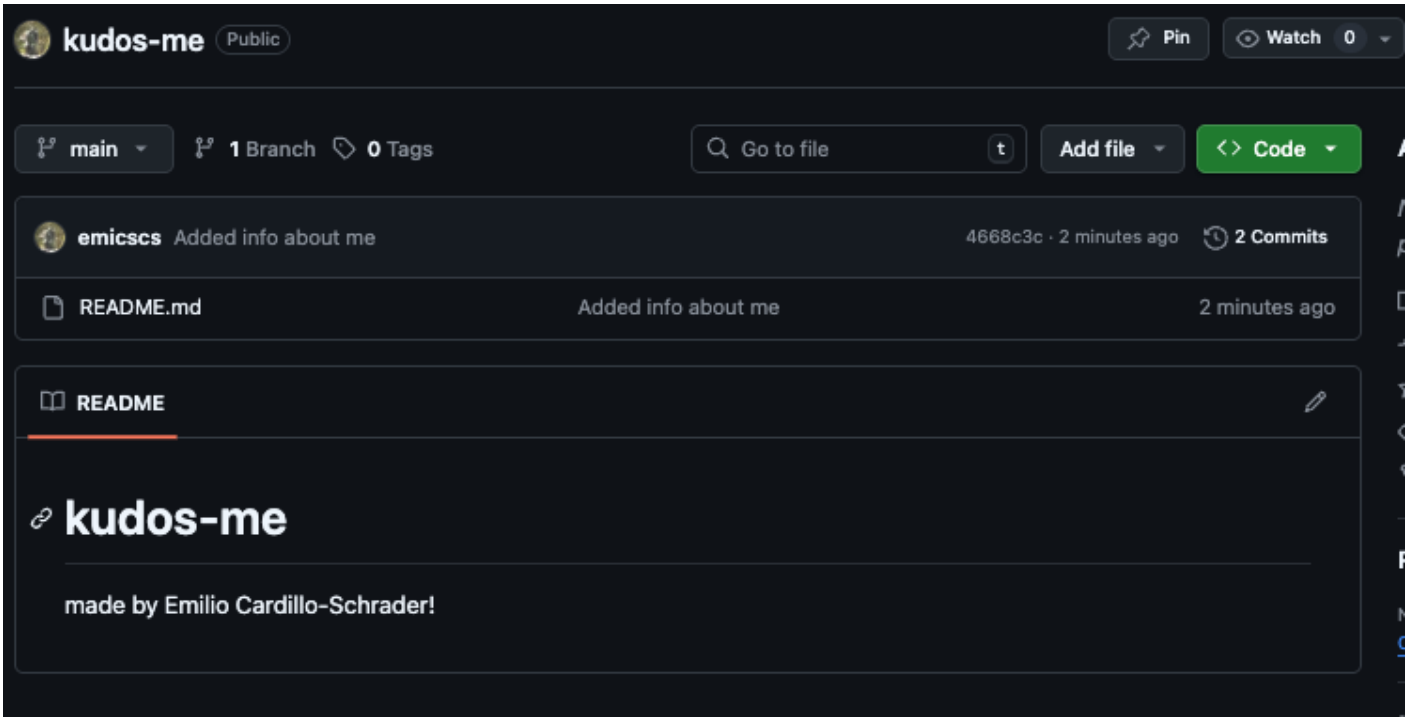
```
git push --set-upstream origin contributors/<your name>
```

Once the branch is set up remotely, or if the branch already existed remotely and was not created locally, we can simply use `git push`.

Seeing our Changes

Finally, we can confirm that all of these steps were done correctly by going over to our repo on GitHub and refreshing the page.

Upon doing so, we should see any changes we made reflected on the web, like this:



Note: Another way to check is by opening VSCode again, and seeing if those green bars from earlier are still there. If not, then that means there are no new lines to stage, and your commit was successful!

Project Outcomes

At the end of our time together, I hope that you all improve on a two key areas of software development:

Technical Skills

- Development Patterns: Hooks, Context, functional components
- State Management: Complex state updates and data relationships
- User Experience Design: Responsive, accessible, intuitive interfaces
- Problem Solving: Handling edge cases and data integrity

Project Management Skills

- Incremental Development: Built feature by feature
- Requirements Analysis: Translated real needs into technical specifications
- Testing Strategy: Systematic verification of functionality
- Documentation: Clear code organization and commenting

Revision #26
Created 10 September 2025 04:27:59 by Emilio Cardillo Schrader
Updated 28 October 2025 00:31:18 by Emilio Cardillo Schrader