

Exercise 1

Getting Set Up for Web Design

What?

Before we start building web pages, we must set up the tools we will use. Then we will spend time using them.

1. Install Software

The following applications are free. Install the following:

Atom (code editor)

<https://atom.io>

Firefox (web browser)

<https://www.mozilla.org/en-US/firefox/new/>

GitHub Desktop (version control, file transfer)

<https://desktop.github.com/>

Adobe Photoshop (graphics editor)

<https://www.it.miami.edu/a-z-listing/adobe-creative-cloud/index.html>

Adobe Creative Cloud is available at *no cost* to University of Miami students. Only Photoshop is required for this class.

If you would like to include original audio, video or vector graphics on a web site, you may need to download Audition, Premiere and/or Illustrator, respectively, to edit these files and prepare them for the web. You can do this at a later date.

2. Create a GitHub Account

GitHub is an online platform which lets programmers share code. One feature is the ability to host simple web pages — like we will be building in this course — for free.

Visit GitHub (<https://github.com/>) and create an account.

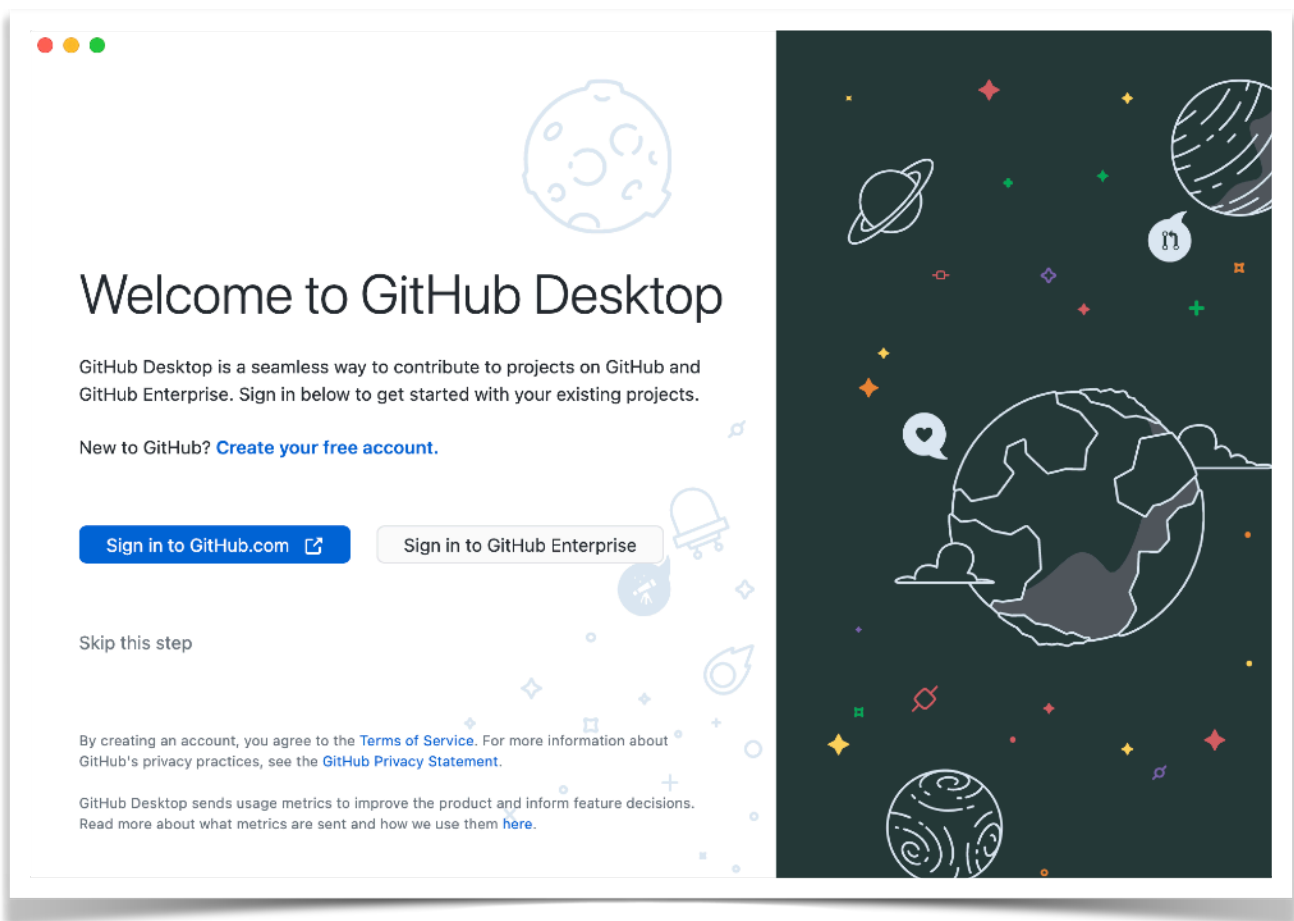
I recommend using your miami.edu address to register since you will be using it for this class. Be sure to **verify your email**; GitHub requires a verified account to host web pages.

Spend about 10 minutes getting familiar your GitHub account. Find your account settings by clicking the icon in the top, right corner of the page and select **Settings** from the menu. Most of the options here can be left as-is, but you should fill out your **Profile**. If you are unsure what a setting does, it is best to leave it alone for now.

3. Log in to GitHub Desktop

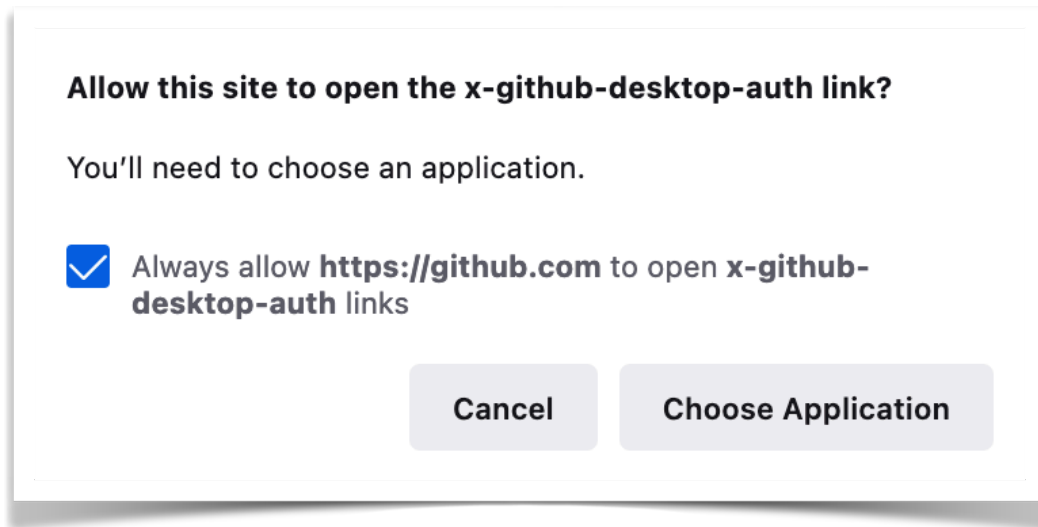
GitHub Desktop is the application we use to keep track of changes to our code and upload our code to GitHub. It is our version control and file transfer software.

Open the GitHub Desktop application you installed in step 1. You should be greeted with the following screen:



Sign in to GitHub.com with the account you just created and verified. This will re-direct you to the web browser which is currently signed in to GitHub. **Authorize GitHub Desktop.**

Your browser will attempt to re-direct you back to the GitHub Desktop application. You will (probably) see an alert in the browser similar to:



It is okay to **Always Allow**. Click **Choose Application**. If you are prompted to choose an application, select **GitHub Desktop**.

Once you are back in GitHub Desktop and see the **Configure Git** screen, click **Finish**.

4. Create a new repository

A **repository** (a.k.a "repo") is a special folder created using GitHub Desktop which keeps track of changes.

Let's practice creating a repository.

Click **Create a New Repository on your Hard Drive**.

Name the repository **practice** (lowercase letters only). (You can call a repository anything you choose, but in this case, we're having it reflect the purpose of this exercise.) Adding a **description** is optional.

This next step is very important. *Please focus!* 😊 Cloud storage (such as iCloud or OneDrive) *does not* play well with GitHub Desktop. We need to make sure this repository is saved to **your computer's physical hard drive** — not the cloud!

In the **Local Path** field look for something similar to:

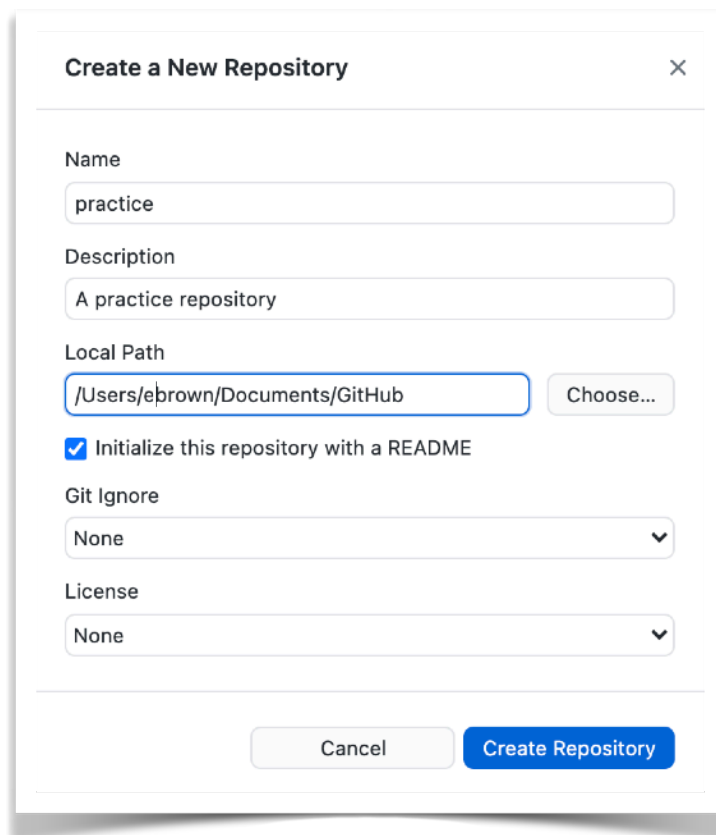
Mac: /Users/<your username>/Documents/GitHub

Windows: C:\Users\<your username>\Documents\GitHub

If Local Path is pointing to a different path, click **Choose** and select a new location *on your hard drive* to save the repository.

Select the checkbox next to **Initialize this repository with a README**. (We'll come back to this in a moment!) You can ignore settings for Git Ignore and License.

If your screen looks like the one below (with Local Path being unique to your computer), click **Create Repository!**



The screenshot shows the 'Create a New Repository' dialog box. It has a title bar with a close button (X). The form contains the following fields and options:

- Name:** A text input field containing 'practice'.
- Description:** A text input field containing 'A practice repository'.
- Local Path:** A text input field containing '/Users/ebrown/Documents/GitHub'. To the right of this field is a 'Choose...' button.
- Initialize this repository with a README:** A checkbox that is checked.
- Git Ignore:** A dropdown menu with 'None' selected.
- License:** A dropdown menu with 'None' selected.
- Buttons:** At the bottom, there are two buttons: 'Cancel' and 'Create Repository'.

This will create the repository on your computer and display GitHub Desktop's main screen.

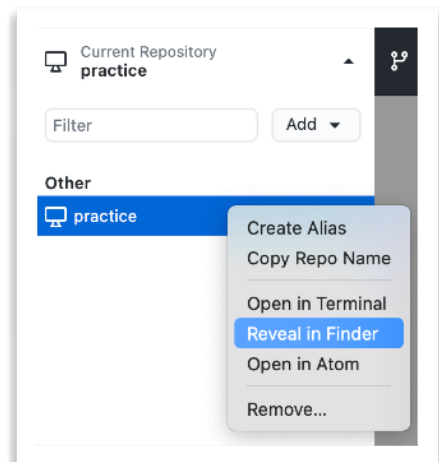
Let's find the repository we just created on our computer.

We can do this by opening up a Finder (Mac) or File Explorer (Windows) window and navigating to the directory path we provided in Local Path. Within the GitHub folder, we should find our **practice** repository, which looks like a normal folder to Mac or Windows. (The special settings which tell GitHub Desktop this is a repository are hidden from our view.)

If you cannot find the repository on your own, GitHub Desktop can help.

In GitHub Desktop, click the **Current Repository** tab in the top left. This will reveal repositories GitHub Desktop is aware of; right now, it just knows **practice**. If you right-click (ctrl-click on Mac) **practice**, it will reveal a menu.

Selecting **Reveal in Finder** (or File Explorer) will open the repository.



5. Tracking changes

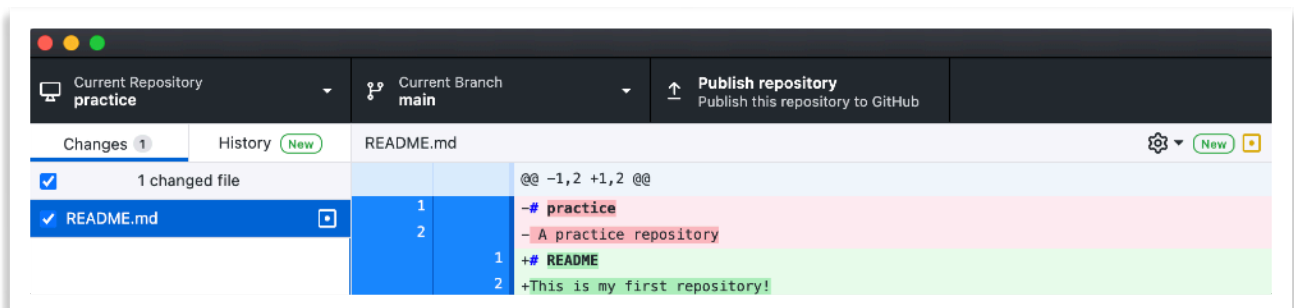
Now that we found our **practice** repository and the **README.md** file within it, let's see how version control works.

Open README.md in Atom, the code editor we installed in step 1. (*Do not double-click the file.* Your computer may try to open it in an application other than Atom.) The file, which is written in a format called markdown, will contain a hashtag (#) followed by the repository name, and the description, if you provided one.

Delete these lines and include the following:

```
# README
This is my first repository!
```

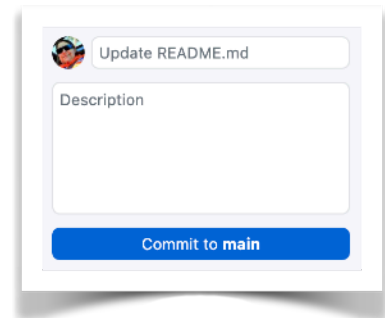
Save these changes. Return to GitHub Desktop.



Notice GitHub Desktop knows what the file originally contained and sees we have made changes. In the left column, we see **1 changed file**, with **README.md** listed below. In the right panel, we see the **deleted text highlighted in pink** and our **new text highlighted in green**.

For GitHub Desktop to keep track of these changes, we need to **commit** them. A **commit** locks down our changes as being the newest, most up-to-date version of our work.

Toward the bottom of the left column, there is a field to the right of your GitHub profile icon. Here we enter a (required) **summary** of our changes. It currently suggests we summarize them as **Update README.md**.



A summary should always describe what is different between our previous and new versions. We can write anything we want. In this case, it is an accurate representation of our changes, so let's go with it. The optional **Description** field is used to include more details if needed.

After adding a summary, click **Commit to main**. The window will refresh. Where did those committed changes go? Notice the very bottom of the left column says we **Committed X minutes ago** and shows our summary, **Update README.md**. Next, click **History** in the left column. This lists all the commits we have made in the history of the repository.

It is important to **remember to commit changes**. Saving a file is not the same as committing it! There is a saying among version control users worth remembering:

"Commit, or it didn't happen."

6. Reverting to a previous commit

A benefit of using version control is the ability to revert to a previous version if something goes awry. Let's take a moment to see this in action.

Open README.md in Atom. Delete the content and replace it with:

```
# This is a mistake!  
Something broke! Oh no!
```

Save these changes. Return to GitHub Desktop. The application is aware README.md has changed since its last committed version.

But what if the changes we saved are mistakes? How do we get back the last committed version that was correct?

In the **Branch** menu, select **Discard All Changes**. Take note of the alert window, then click **Discard All Changes**.

Return to Atom. If the file with "broken" text is still on screen, close the file and re-open it in Atom. You should now see the "fixed" version. (Always remember to re-open the file after discarding changes!)

Why is this helpful? As more complexity is introduced in code, you may break something and not know how to get it back to a working state. Making frequent commits — but not necessarily every line of code you write — gives you a safety net. Rather than starting over from the beginning, you have can revert to the last commit which you *know* worked.

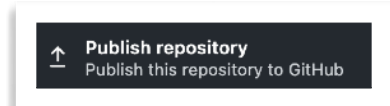
Knowing when to commit takes practice and time to develop. We will try to demonstrate good opportunities for commits during live demos.

7. Publishing

GitHub Desktop is also used to transfer files from our computer to github.com, where our web pages will be hosted.

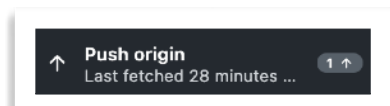
This step may be the easiest. Click **Publish Repository**.

This will display a window which has fields pre-populated with the repository name and description. Make sure **Keep this code private** is **unchecked**. Click **Publish Repository**.



Visit your repositories page on github.com. You will see **practice** listed. Click **practice** to see the files. You will see README.md (and maybe a 1-2 files beginning with a period; you can ignore these). Each file has a commit summary, as well as how long ago that commit was made.

When you make additional commits to a repository, the Publish Repository button will change to read **Push Origin**. "Push" is GitHub's fancy way of saying *upload*. ("Fetch" means to download.)



The Push Origin tab includes a bubble to the right indicating how many commits are to be pushed. It is worth noting you *do not* have to push after every commit. This gives you the luxury of making changes while offline (i.e. on a plane). Once you regain Internet access, all your commits can be pushed at once.

8. Deleting a repository

Ideally, you will not need to delete a repository in this class, but there are rare instances where it may be necessary to start over. There are two steps:

1. Delete the repository on github.com. Open the repository, click **Settings** and scroll to the very bottom. You will see an option to **Delete this repository**.
2. Delete the repository from your computer. In GitHub Desktop, click **Current Repository**, right-click (ctrl-click) on the repository you want to remove and select **Remove**.

9. Your Turn! (Part 1)

Time to apply what you learned! Use the **practice** repo for this part.

The following files are not named well based on the naming conventions discussed in lecture this week:

- Internet Sociology Paper
- HURRICANES FOOTBALL SCHEDULE
- 2022 Travel Plans
- Distraction article - September 2021
- Fall Schedule

Your task is to create these files using **proper naming conventions**, **commit them to your practice repo** and **push those commits** to github.com. (To create the files, start a new file in Atom, write a line of sample text and use proper naming conventions when saving.)

You *could* commit all the files at once. But to develop more familiarity with the process, commit the files one at a time, or group multiple files in one commit. For instance, you might group fun-related documents in one commit and school-related documents in another.

We will discuss acceptable answers in class.

10. Your Turn! (Part 2)

Your first assignment (due at the end of Week 3) involves writing and uploading a basic web page. While you have not yet learned HTML, you *can* get your web-hosting repository set up!

Create a new repository named `username.github.io` — where username is *your* GitHub username. (For instance, if my GitHub account name is `ebrown15`, the repo name `ebrown15.github.io`.)

Create a new file in Atom. Within this, write:

Hello, world!

Save this file as `index.html` to the repo you just created. Commit this file. Push it to GitHub. Visit `github.com` to confirm the file was uploaded.

Guess what? You just published your first web page. Don't believe it? Open a new browser window and type in (replacing "username" with your GitHub username):

`https://username.github.io/`

... and you should see "Hello, world!" 😊

If you completed this step before Monday noon of Week 2, email me the URL of your site for one (1) point of extra credit on the GitHub Setup project.

DONE!

Whew! That was a lot. Good job!

To revisit what we accomplished:

- We set up the **software** needed to build web pages. We created a **GitHub account**, which will eventually serve as our **web host**.
- We learned how to **create a new repository** in GitHub Desktop and how to **commit changes** so the application can keep track of our work.
- We learned how to **push our commits** to GitHub so our work is publicly accessible (as it will be on the web).
- We practiced how to **write proper file names**.
- We **set up our web hosting repository** where all assignments will be pushed to and **published our first web page**!