Make a New Repository on GitHub and Edit a Markdown File

Learning Goals:

- Create and use a GitHub repository
- Add files to the repository (without coding)
- Make changes to files
- Push changes to your repository file.

Prerequisites:

- Internet access
- An active GitHub Account

What is a repository?

A GitHub Repository – often called a "repo" – contains all the files and folders involved in a project. In addition, a repository also contains a properly logged history of all changes that have been made to all the files. This entire history is ordinarily kept hidden, so all you generally see is a small but informative history.

The files inside a repository can be text-files, code, images, spreadsheets, etc. One thing that Git and GitHub are not very good at is handling large-binary files, such as large datasets or images. So, in general, large images banks and datasets are generally stored somewhere else. This is not something you will need to worry about for this class, but it's good to know.

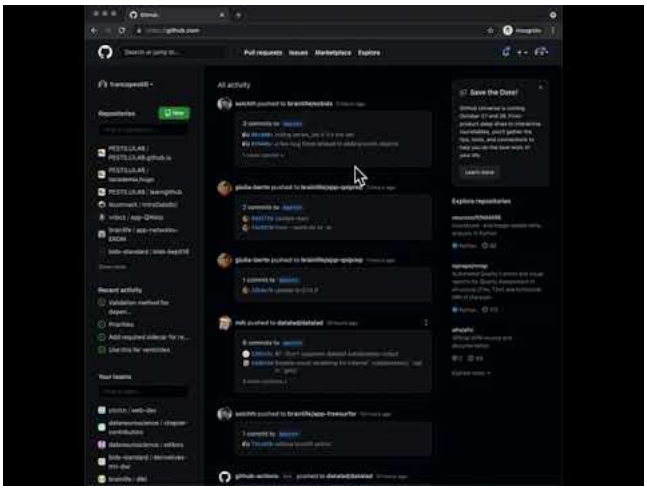
Repositories can be owned by a single user, or by an organization (think about a team of users).

GitHub repositories can be public or private, so you can invite anyone and everyone to work on your project, or you can keep it all to yourself.

Create a repository

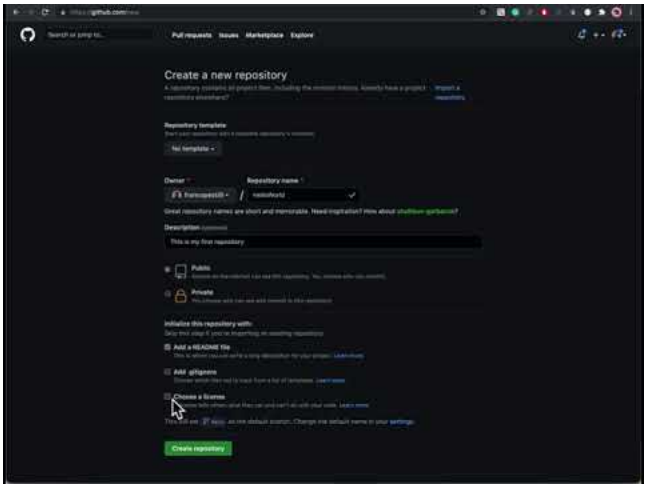
1. Navigate to GitHub.com/yourUserID and Sign in to your account.

First, we will use our prefered web browser and navigate to GitHub.com. More specifically, we will want to navigate to your GitHub account website. This website is automatically generated by GitHub when you create an account. The name of your GitHub website is "GitHub.com/" followed by your user ID, so for example my user account on GitHub.com can be found at https://GitHub.com/francopestilli (remember your User ID was defined in the previous Tutorial). After navigating to our user account, we will *sign in*. That is it! Once logged in your GitHub User Account we will start working with Repositories.



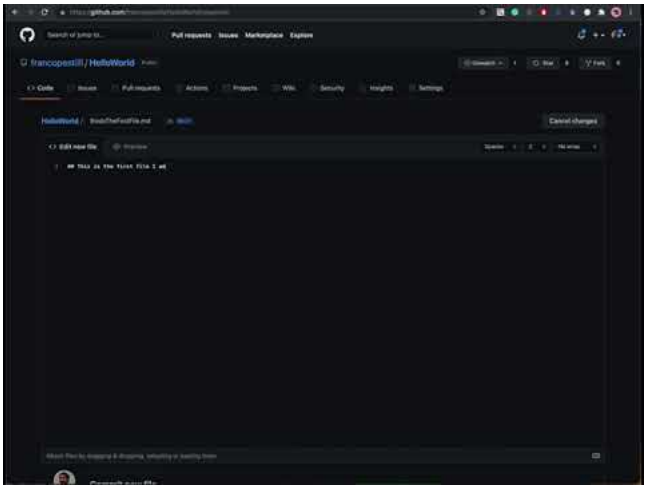
2. Create a repository: Using the Web User Interface (Web UI).

We will next learn how to use the GitHub.com Web User Interface (WebUI) to create an empty git repository. The first thing we will do is create a ReadMe file and pick a license (MIT 2.0). This is a nice article on why it is important to add Licenses to GitHub.com Repositories. Also ReadMe files have become critical to GitHub repositories. They provide a "landing page" for your project and there are multiple ways to customize them. You can read more about ReadMe Files here.



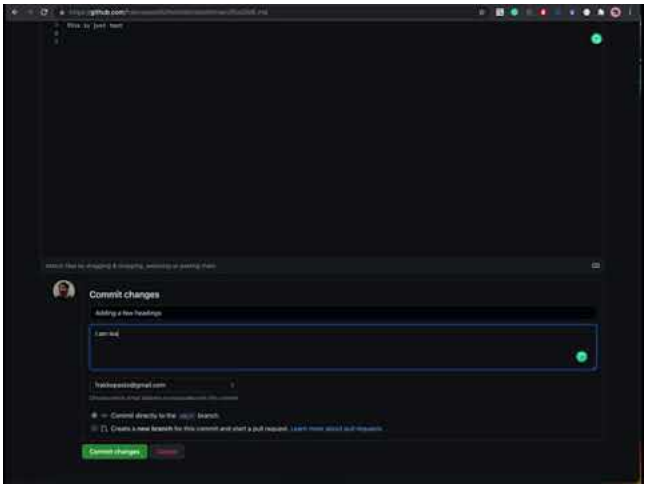
3. Edit a repository: Add a file to it and then edit the file.

Once a repository is created the next thing we will want to do is to add and edit files. This is indeed why we have created the repository in first place! We will want to use GitHub and get help with content management and version control as we develop data analysis skills. Although this might sound confusing at first (we have not yet used the repository), please bare with us for the moment and let's experiment a little bit on how to add and edit the file all via the Web UI.



4. Edit a repository: Edit an existing file.

We have now created a repository, added a ReadMe file, a license, and an additional file. Let's practice opening one of the files and editing the content of the files using the Web UI. We will edit the ReadMe.md file, this will give us the opportunity to understand little bit more about MarkDown (or .md files) and have fun with the landing page for the repository. The video below will show you how to edit a ReadMe file using the MarkDown Syntax. MarkDown is an easy way to make a formatted text file – files that can have *italics* or **boldface** for example, like the one you are reading now. In our example, we show how to add headings (I called them headers in the video, oh well...) and a list of items. You can think of Markdown is a simple version of html, the formatting scheme used to create webpages.



Additional readings about repositoroes can be found here.

A few notes about MarkDown. Think about markdown as a programming language. You type, you add a few extra symbols or 'syntax' and that will allow render the text with special formatting.

For example, a few useful commands can be found here (adapted from the GitHub Guide:

Font Type	MD command	text rendering	text description
Bold	** **	**This is bold text**	This is bold text
Italic	* *	*This text is italicized*	This text is italicized
Strikethrough	~~ ~~	~~This was mistaken text~~	This was mistaken text
Bold and nested italic	** ** and _ _	**This text is *extremely* important**	This text is extremely important
All bold and italic	*** ***	***All this text is important***	All this text is important

You can even make a table in MarkDown but using the symbols `|` and `-` creatively. For example the following commands:

```
| This | is | a | table | header |
| --- | --- | --- | --- | --- |
| this | is | the | table's | content |
| More | content | is | shown | here |
```

will render this table:

This	is	a	table	header
this	is	the	table's	content
More	content	is	shown	here

There are many commands in MarkDown (explore for example you can add Emojis 👽, by starting typing : and selecting your favorite 🐜!). You can even add a URL to your MD file in the following way: `[URL NAME HERE](URL HERE)` for example here is the URL to all the commands you can explore and learn about: MarkDown Syntax.

Note that we will ask you to learn more about it for the next lectures and the first report!

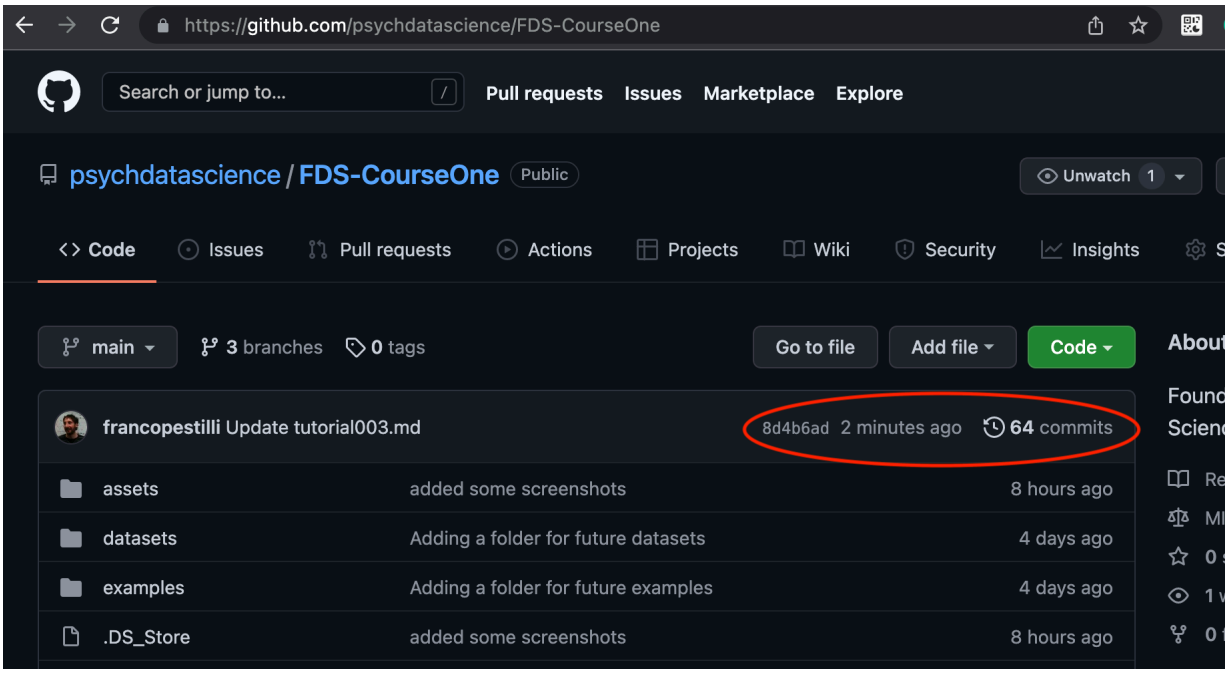
4. Git Commits

What is a `commit`?

A "commit" is how we say to `git` "*I've changed this file and I really want the changes to be a permanent part of my Repo!*" So, it is an important operation as it makes temporary changes, permanent, hence: you are committing to the changes! But remember, the beauty of `git` is that you can

always go back to an earlier version if you need to!

As we edited and saved files and commited changes to the files in the tutorial above, you might have noticed that online, GitHub added information about the files that appeared changed. The information about the changes, and `git commits` appear on top of the repository website. The image below shows where GitHub displayed information about one change I made to a file in this the repository for this class using the web browser. This is an example of a `git commit` as displayed by GitHub. Because commits are so important, GitHub displays information about the latest commit (`ID`, `User` that committed and the `timestamp`) at the top. You can also see how many commits have been made to the project, and clicking on the number of commits or the little unwinding clock will show you a list of all the commits.



Our goal is to write code, to not mess up our code (even though we will!), and to be able to go back to working code when we do mess up. We use `git` and GitHub as version control systems to help us track down any potential mistakes and correct them. A commit is the simply the operation of saving the latest code changes to the repository. The commit makes the code part of the most up-to-date version of the repository. More specifically, we use commits to both save the recent code inside the repository of code and to add a record in the log of the history of the repository to remind us (just in case) that we made a change at a specific time. Commits are kept in the repository history indefinitely.

Once a recent change to the code is committed to the repository the new changes become officially part of the code repository. Uncommitted code versions are visible only to you, locally on your machine or in your web broswer. Committed code is visible to everyone that has access to the repository.

In summary, we use commits to keep track of any changes we make to our stuff. At every commit, `Git` will create a timestamp in the log of the code repository. The commit code will also be saved in the repository and protected so we easily roll back to an earlier version if we need to. See https://en.wikipedia.org/wiki/Commit_(version_control) for further reading.

The idea of a "commit" actually predates `git` and GitHub (as in and older version control system called `SVN`, for example). Imagine a lonely developer in an ivory tower. It takes many hours of solitude for the developer to write useful blocks of code. The developer writes for several hours going back and forth on the same lines of code. After multiple tests, the developer decides that the code she wrote is OK. It serves the purpose. That is when the developer is ready to `commit`. The words literally means that the code block that was written is meaningful for the project (or at least the developer things so at the time), so meaningful that it would be important to save a copy of the code into the code base in the repository.

Now, so far we have not really written code. Only created repositores, files and edited them. Yet, if you go back to the previous tutorials you can appreciate that every time we made a change change to a file, we made a change to the repository. Before the change became live and visible online we had to click a `green button` indicating that we were committing. So we have been committing to the repository all this time without knowing. Or at least, without discussing it. Great.

Here is some additional reading material about `commits`.

How does a commit work in Git (in broad strokes)?

You might think that when you make a commit, only the file or files you change would be updated. But, actually, each commit creates a snapshot of the *entire project*. You only have to worry about the specific changes you made but, automatically, Git creates a copy of the entire project that is frozen in time forever. This way, Git makes sure that we always have a record of *all* our code and files each time a change was made. This means that, in terms of our project, we can be *time travelers*, moving to the past and back to the future as we wish!

Make commits useful.

You should make new commits often, based around logical units of change. Over time, commits should tell a story of the history of your repository and how it came to be the way that it currently is.

Commits include lots of metadata in addition to the contents and message, like the author, timestamp, and more! One very important thing that goes with a commit is a short summary of the changes you made (this is actually required by Git). In addition to that there is an optional field where you can add additional comments, with more details and structure. Even though the description is optional, you should always provide the best description you can. You are helping out future you when you do this!