Getting Started with GitHub + IDEs

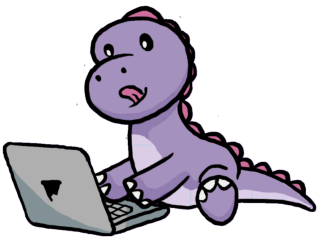
Kaalkidan (Kaal) Sahele

Picture this:
The time is 23:59 - you have a minute to submit your coursework. You've spent months on this coursework, you pushed through the days and battled through the night, but it's done. All that's left is to submit.

So we're just going to submit "Coursework_(final)"....
or was it "Coursework_(final)(final)" ….
or was it "Coursework_(FINAL VERSION SUBMIT THIS ONE)"...
maybe "Coursework_(NO DON'T SUBMIT THAT ONE, SUBMIT THIS ONE)"

But it's too late. It's midnight - time is up.

Well, fear not. Those days are over. There's a bright new dawn on the horizon, and it's name? Version control. Better yet: version control and software development with GitHub.



1 - Installation

1.1 - Create an account

If you don't already have a GitHub account, go to GitHub's website (https://github.com/) and sign up for a free account.

There may be an option to set up a student account but this will ask for verification so the best option for now is to just make a free one and then upgrade it later once you're familiar with the workings of Git.

1.2 - Install Git

For a Windows

GitHub uses Git for version control. This is what does all the tracking of changes in code and then GitHub is a web-based hosting service to host these Git works. You can't use GitHub without Git (hence the name).

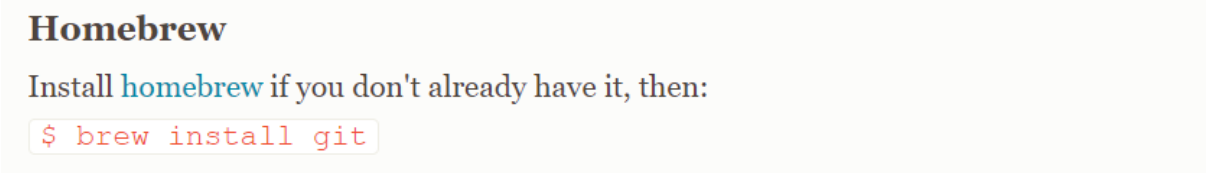
Download Git from their website: https://git-scm.com/downloads

Then once it's downloaded, go into your downloads, open the .exe file and click yes to allowing it to make changes.
Then click "Next" on all the pop up pages until it starts installing.

For a Mac

Unfortunately, if you're on a Mac, you have some extra steps. The easiest way to do this is via homebrew:

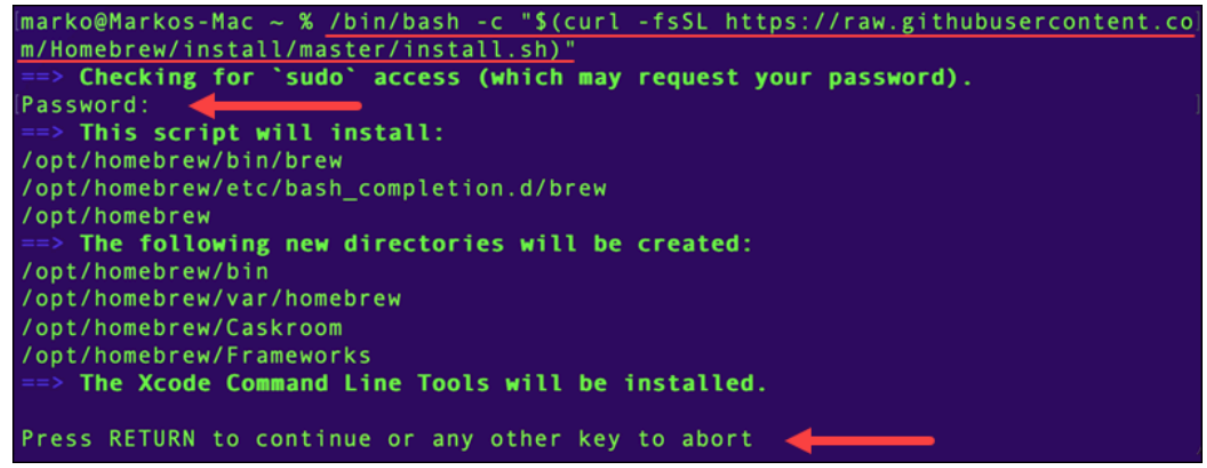
Click the *beautiful bloo* link for homebrew:



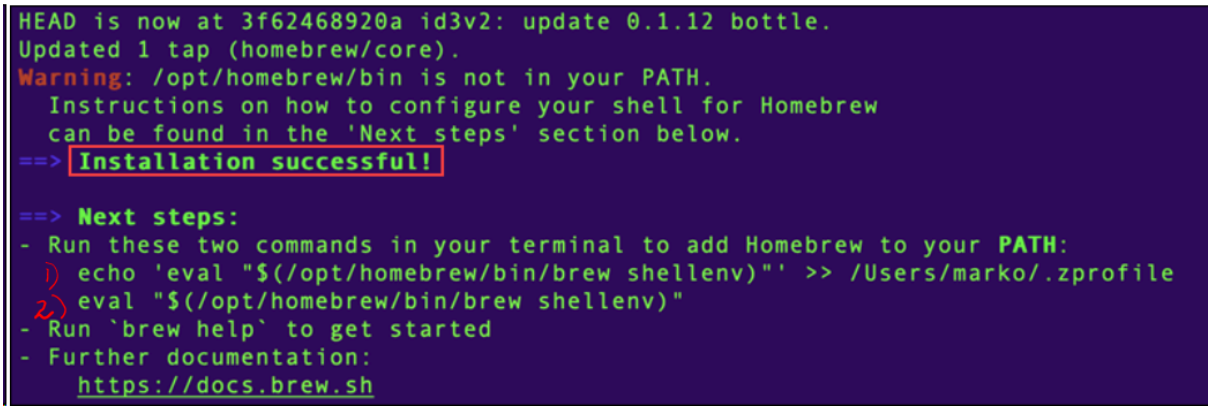
That will take you to this page:

Copy that line of code.

Then head to your launch pad and search for "terminal". It will open up a scary looking application, but I promise it's not scary. Paste that line of code into your terminal and press enter.



When you type in your password, **it won't show the characters being typed but this is for security reasons, it is in fact typing what you put in, so as soon as you type in your password, just press enter.**



Run the two lines it tells you to run - first, run line 1, then run line 2. Then you will have successfully installed brew. Hooray - you just installed the thing that lets you install git!

Now that you have brew, you can run these three beautiful words:

```
$ brew install git
```

And now you sit back and watch it work its beautiful magic and install git!

1.3 - Set Up Git

After installation, configure your Git identity by opening a terminal and entering the following commands::

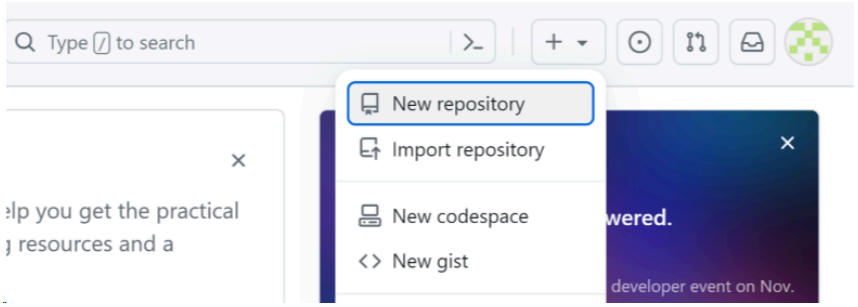
```
git config --global user.name "john smith"
git config --global user.email "youremail@example.com"
```

Make sure you use the username and email that you created your github account with!

2 - Creating Your First Repository

2.1 - Create a New Repository on GitHub

Log in into your GitHub account and find the the “+” in the upper right corner and select “New Repository”:

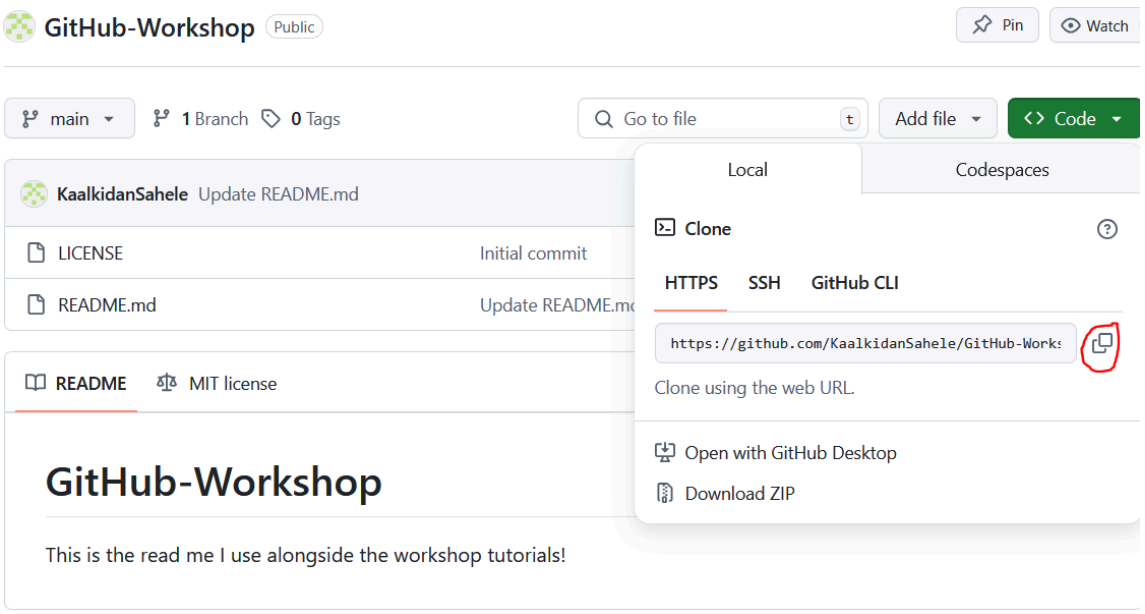


You’ll be directed to a page that asks you to fill out some information, such as the name of the repo, whether you want it to be private or public etc.

For future reference, if you ever use GitHub for your university projects, **always ensure they are private.** If they are left public and a student in the future plagiarises off of you, you will be in equal trouble (!!)

2.2 - Clone the Repository

The following is an example of a repo I made. Use your own repo that you just made in the previous step for all of the following.



Then you can clone this repo by heading back to your terminal and copying in the following lines of shell script:

```
git clone https://github.com/yourusername/your-repository-name.git
```

```
Command Prompt
Microsoft Windows [Version 10.0.19045.5854]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kaalk>git version
git version 2.47.1.windows.1

C:\Users\kaalk>git clone https://github.com/KaalkidanSahele/GitHub-Workshop.git
Cloning into 'GitHub-Workshop'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (7/7), done.
```

2.3 - Navigate to the Repository

Then use the cd command to navigate to your repository's directory in your files:

```
cd your-repository-name
```

3 - Using your repo from your IDE

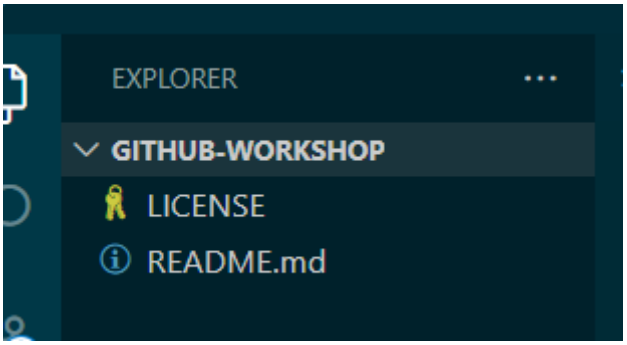
3.1 Using Your IDE

IDEs are integrated development environments and, while aren't crucial, make programming and software development projects so much easier. It's practically impossible to code without them.

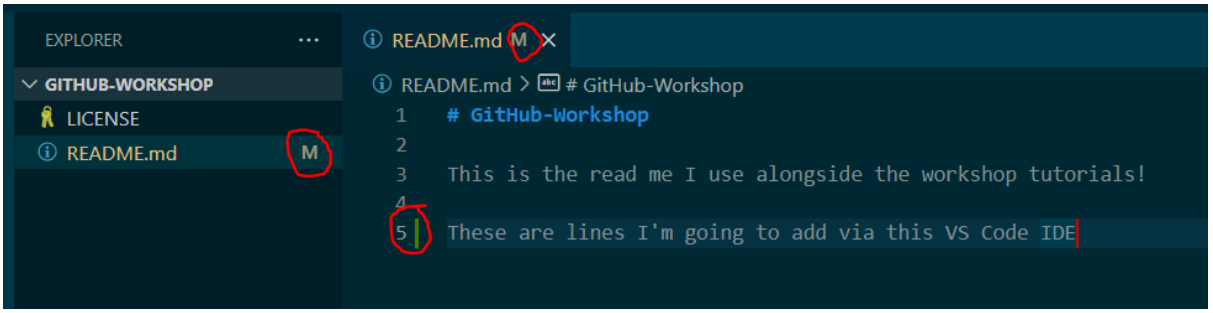
The most commonly used one is VS, which can integrate your GitHub repo to keep track of your changes.

First open VS code and locate your cloned repository and open it by going to:

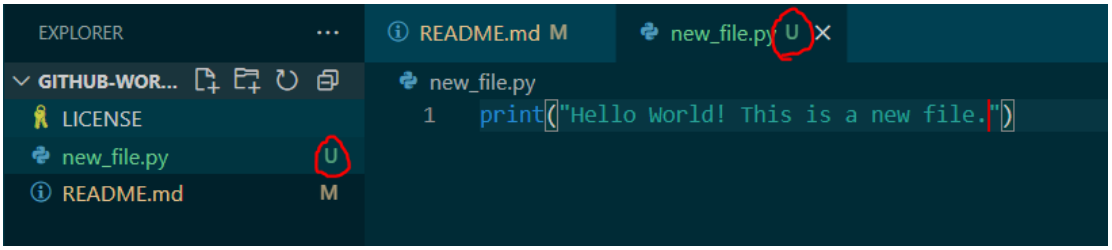
File >> Open Folder >> (find your folder) >> Select Folder



Then whenever you make edits to files in this cloned repo via the IDE, it will show up via green and red indicators (indicating added lines and deleted lines):



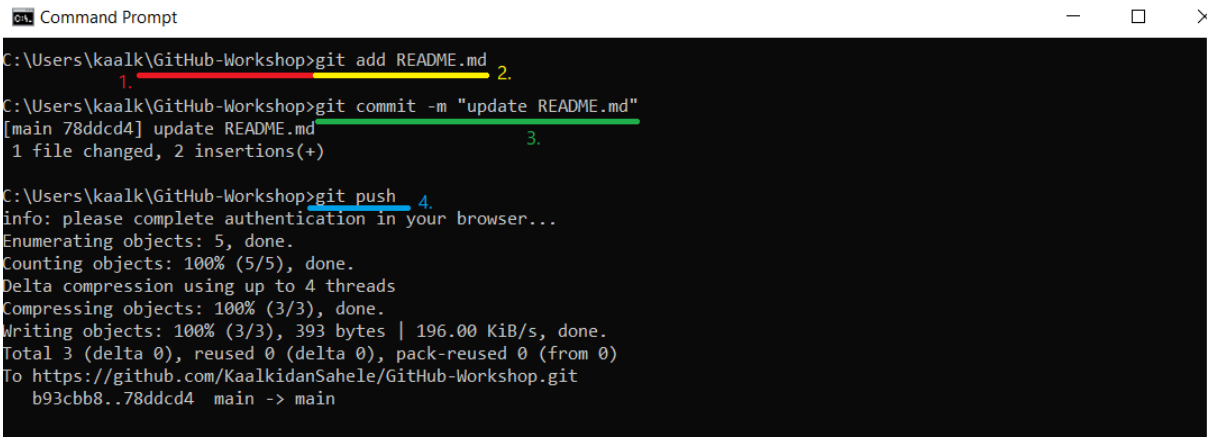
The **M** here indicates modified. If you add a new file, you would see **U** for untracked, which means it's a new file that hasn't been added to your git space yet:



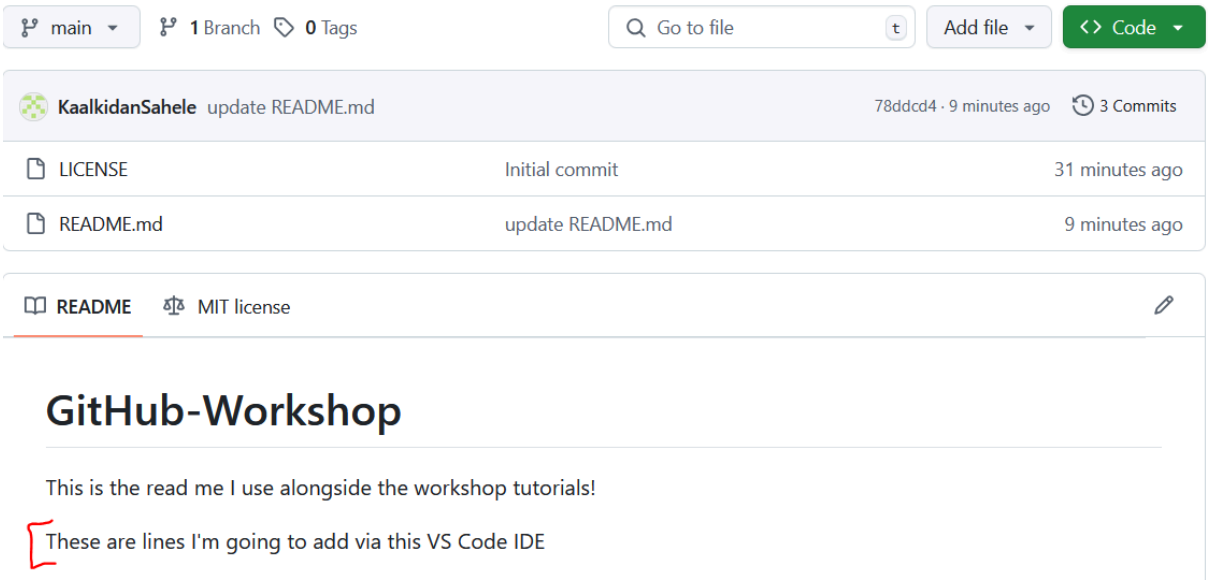
3.2 - Git Commit

To commit your new code you'll need to follow the same steps every time:

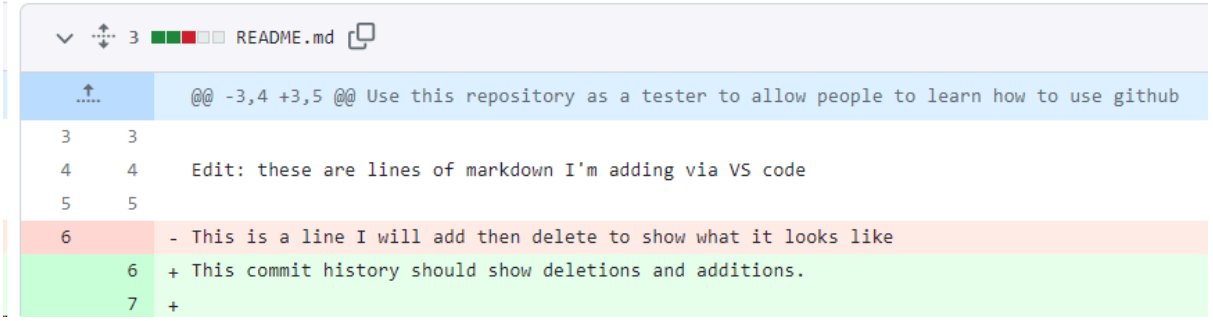
1. Make sure you are in the directory of your cloned repo
2. Git add [file name]
3. Git commit -m [message] e.g.
4. Git push



These changes will register in your repo as such:



You can actually see exactly what changed in a previous commit if you click on it. This is pretty useful as it allows for inspection of what's changed:

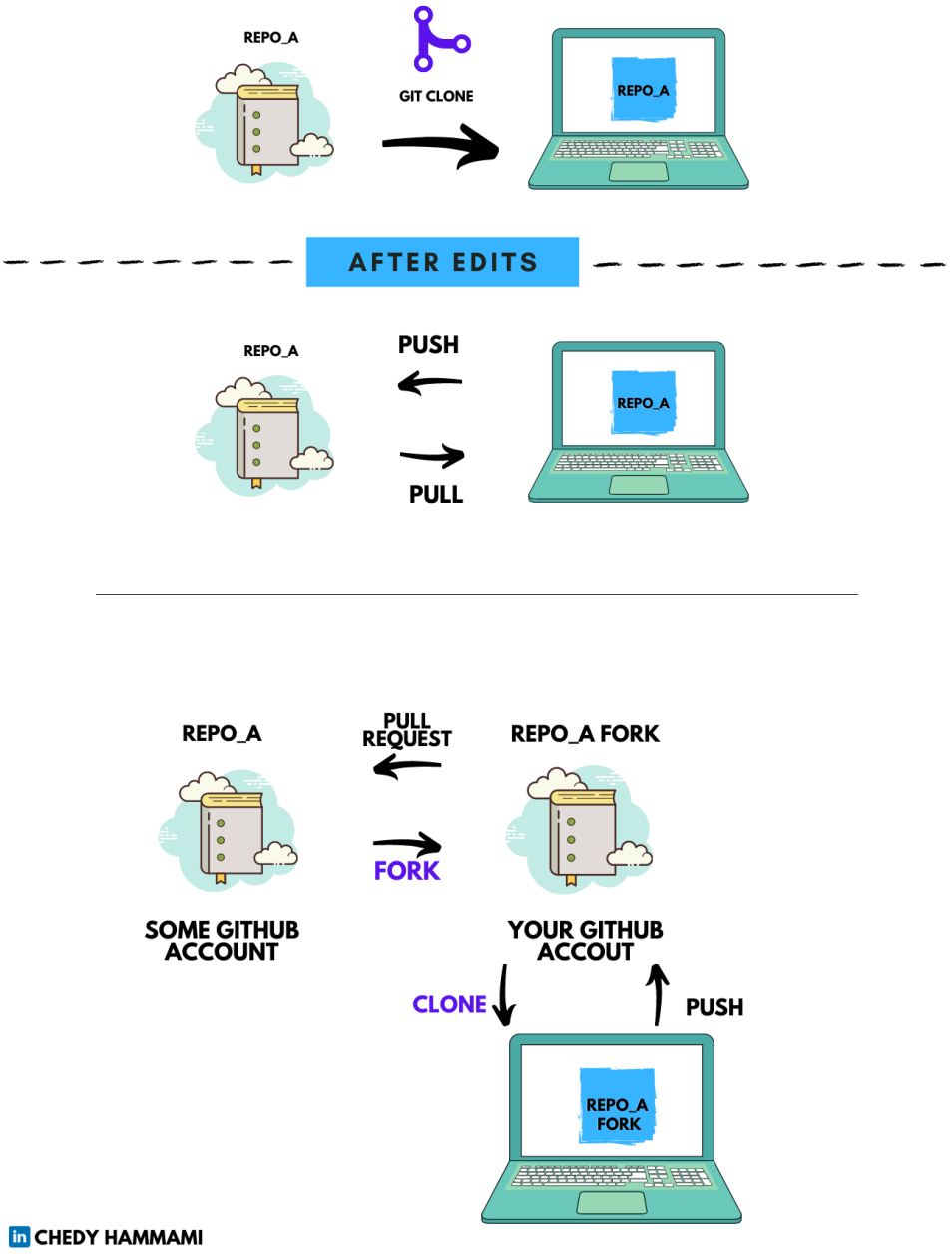


3.3 - Cloning vs. Forking

If you're working in a team from the same repository, then each team member can individually clone it, make changes, and push back to the main repository (provided there are no conflicts). To do this, each member needs access to this repository.

If you're working from an existing repository that nobody in your team created and has control of, then an appropriate move would be forking the repository, and then cloning this forked repository. This means you have an exact copy of an existing repo, but now you have control over this forked version.

Git committing would allow you to make changes to this forked version, and once you've made all the changes you'd like, you can ask for a pull request on the original repository. A pull request is a developer asking a repository owner to *pull* the changes they made up onto the original branch.



3.4 - Git Branch

When team work becomes the dream work

Say you're project is a web app, and you have someone working on front end and someone working on back end and you don't want to mess up the working prototype you have in your main branch. You can start two separate branches for front and backend. This essentially makes a copy of everything in the main branch for you to toy around with without affecting the main branch!

If you want to make changes on your local machine, you can use branches to work on separate features or fixes.

You can create a new branch by:

```
git checkout -b new-branch-name
```

Alternatively, you can re-clone the repository to your local PC (this only works for very small project - as soon as they become larger, this gets very impractical).

So now your project has the main branch, and the new branch. Once you've finished toying around with your version, you can **merge back** to the main branch.

If you're working on a branch, any commits made to the main branch will not appear in your branch. It's very important that when you work on a branch, you don't work on it for too long so it doesn't end up to far behind the main branch that it causes conflicts when you try and merge your changes back.

Find more information on branching here..

3.4 - Merging

Once you've made changes to your branch/ local copy, you'll need to merge it back with the main branch. You can do so with:

```
git checkout main
git merge new-branch-name
```

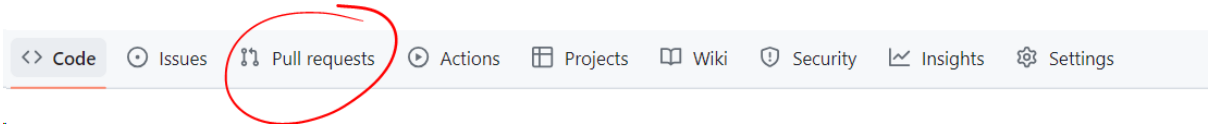
If you're only working from the one main branch, you don't even need to specify the branch name, just use "git merge".

Then to update your local repository with changes made on GitHub, use

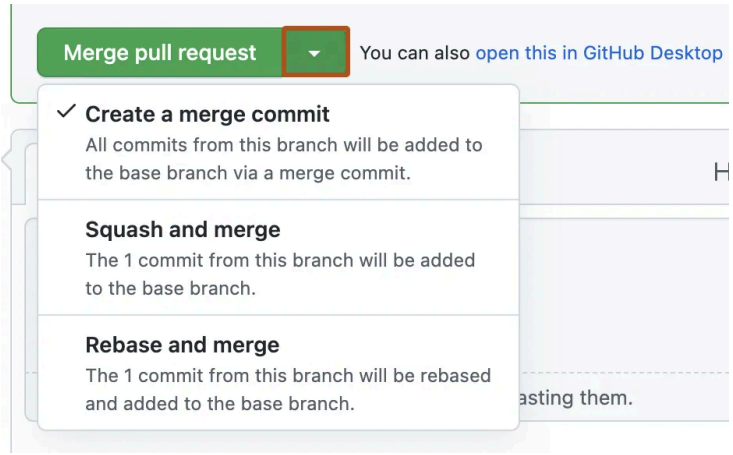
Just ensure you're in the directory of your repo and type the following command:

```
git pull
```

You'll be able to see all the pull requests for a repo by selecting that tab:



Then it's up to the owner to review and merge these pull requests into the main branch. This can be easily done by clicking the big green button:



If there are bugs in the code that makes it incompatible, it will through a merge conflict error so be weary of this.

Differences between Merge Commits

Those of you with particular eagle eyes will have noticed the different types of merges.A tl:dr way to remember which type of merge to use is the following:

Merge Commit
Your typical commit. Combines all changes from your branch `branch-name` into `main`, resulting in a new commit on `main` that represents this merge.

Squash Merge
Suppose you have a feature branch (`feature-y`) with 15 commits. When you squash merge into `main`, those 15 commits will be combined into a single commit and added to `main`. Use when you want to simplify the commit history and don't care about traceability.

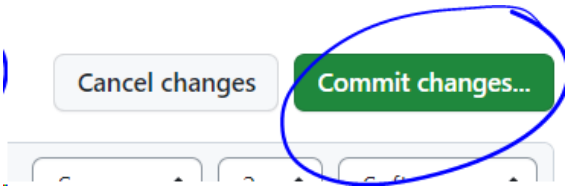
Rebase and Merge
Useful when you have a branch (`feature-z`) that is behind `main` by several commits. Rebasing updates the branch `feature-z` so that its base matches the latest state of `main`. Once rebased, the branch can be merged linearly without a merge commit. Useful when you want to replay your changes on top of the latest `main` to ensure a clean integration.

🎉 **And that's all the basics - you should be able to hopefully manage a project on it. Congrats! You're a software developer now :)** 🎉

Cool Fun Facts, Tips, Tricks and Advice with using GitHub

GitHub Desktop

If the command line scares you, you're not alone. You can do everything we covered today on a much more friendly interface called GitHub desktop where rather than writing lines of code to commit, you just click the friendly green button:



Collaboration

GitHub allows for, and actually encourages, collaboration. To work with others on a repository, add them as a collaborator through the repo settings such that everyone can contribute to the development of a project.

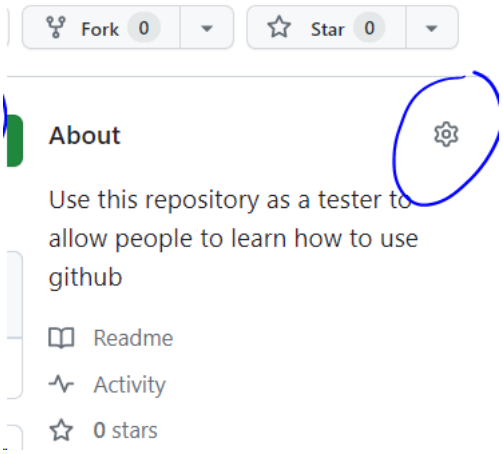
Stars & Achievements

For each repository you make, you can collect stars and achievements (which potential employers love to see). These are given by other programmers on GitHub and can be for a range of reasons, even just having a pretty README.md

Profile READMEs

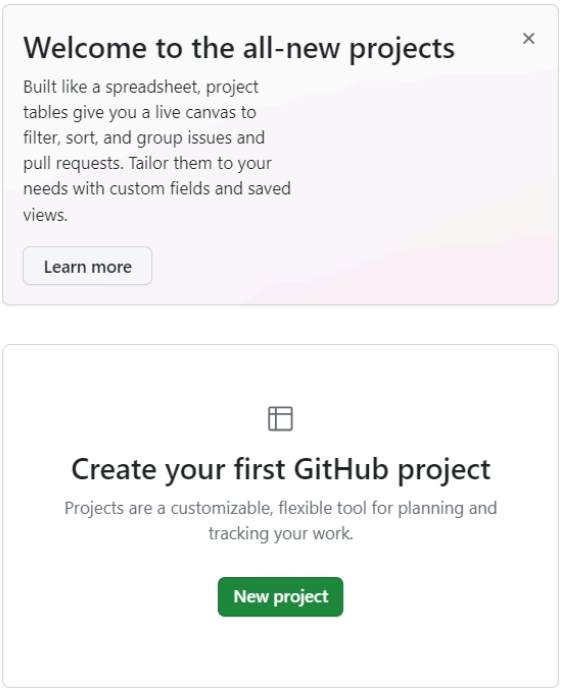
The same way your project can have READMEs, so can your profile. If you want to present yourself via a mini CV, this would be the perfect place to do so.

Adding Tags



Lets you add tags so people can find your project more easily

GitHub Profile Projects



They're like Trello where you can split your tasks between done, currently doing, and done. ***Very useful for a hackathon.***

And many other cool features :)))

Workshop Take Aways & DurHack

There are lots of other fun and cool things you can do with GitHub so if there's one thing I'd love for you to take away from this workshop is that this all looks and sounds scary because there's so much technical jargon being thrown your way - "terminal", "repository", "clone", "commit".

But I promise you, it's not scary - the terminology sticks the more you use it. You can start with "I'm going to whack this back into the online safe space so I don't lose it" and soon enough, you'll be the one teaching others how to use it.

So have a play around with it and if you get stuck, GitHub is so deeply rooted in industry that there is most likely no issue that someone hasn't already made a post about on stack overflow or the GitHub help page, so have a little google and if not - seek your nearest computer scientist. Just make sure you bring them cookies (!)

DurHack will have mentors at the event ready to assist you so don't worry about being a pro at any of this, it comes with practice. There's no better place to get started than in a hackathon where the nation's greatest programmers at 2 feet away from you, so give it a go and don't be afraid :)