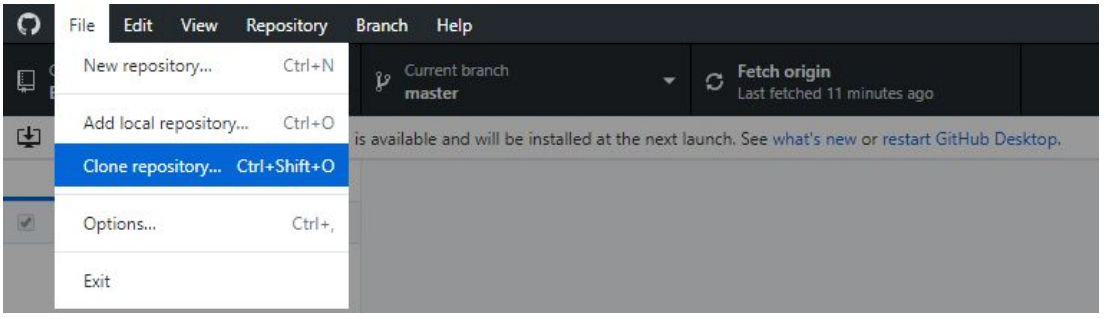
Introduction

Git is a source control system used to facilitate synchronizing work within a team. It essentially works by taking snapshots of the project and storing a history of them. This allows us to navigate through different stages of our repository and to branch out versions. in order to encapsulate our work in progress in a way that it doesn't affect the other team members. When the new feature is completed, the branch can be later merged into the stable version (usually located in the *master* branch).

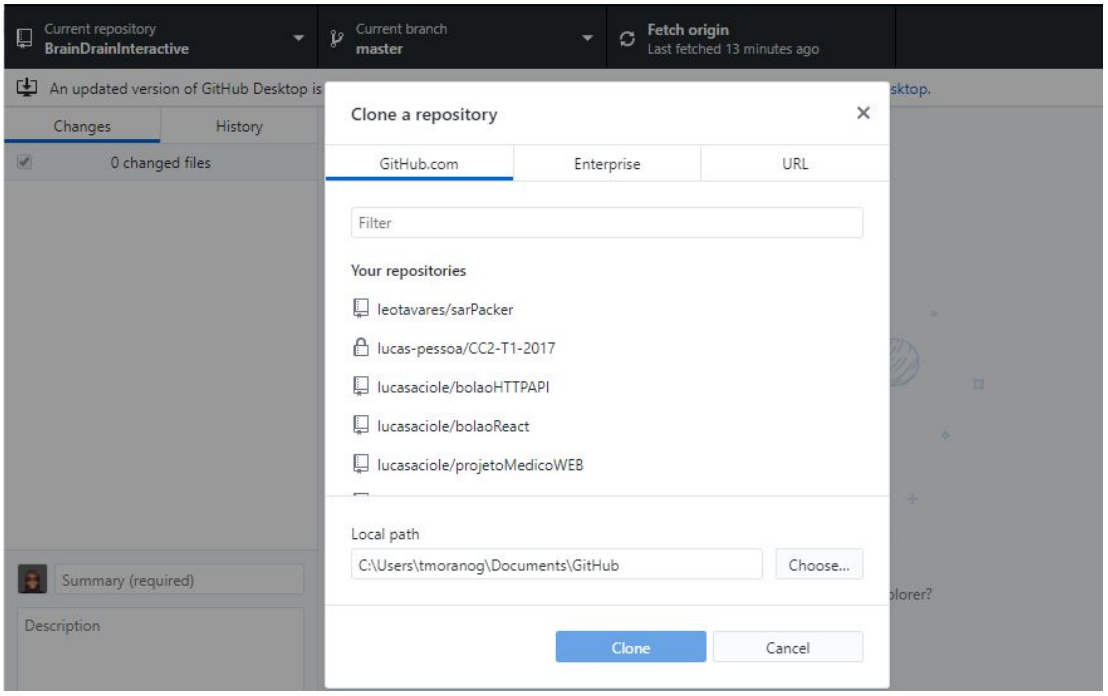
Setting up

After GitHub Desktop has been installed, we need to set up the repository locally. For that, we clone the repository from the origin.

1. Go into the File > Clone Repository (Ctrl+Shift+O for hotkey)



2. This will open the following window. You just have to select the Repository (ThiagoMorano/BrainDrainInteractive, in our case) and the directory you want to download the project to.

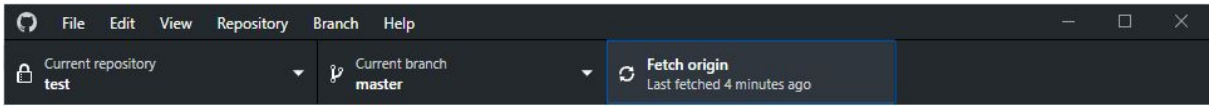


It's also possible to clone using the link. You can use either ThiagoMorano/BrainDrainInteractive, or the link https://github.com/ThiagoMorano/BrainDrainInteractive.git

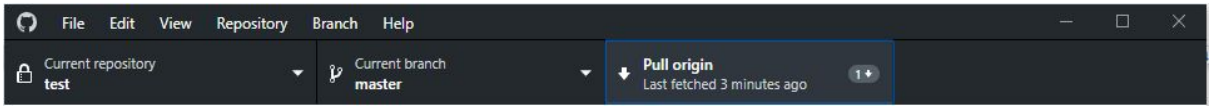
Workflow for updating your local version

Git doesn't update our local version of the repository automatically. In order to do that, we need to fetch and pull the changes.

1. Fetching is done by clicking in the button "Fetch origin". That checks if there any changes in the origin that you could be missing locally.



2. If there are changes, after fetching the button transforms into the Pull origin button. By clicking it, it downloads the changes into your local repository and updates it to the most recent version of the branch.

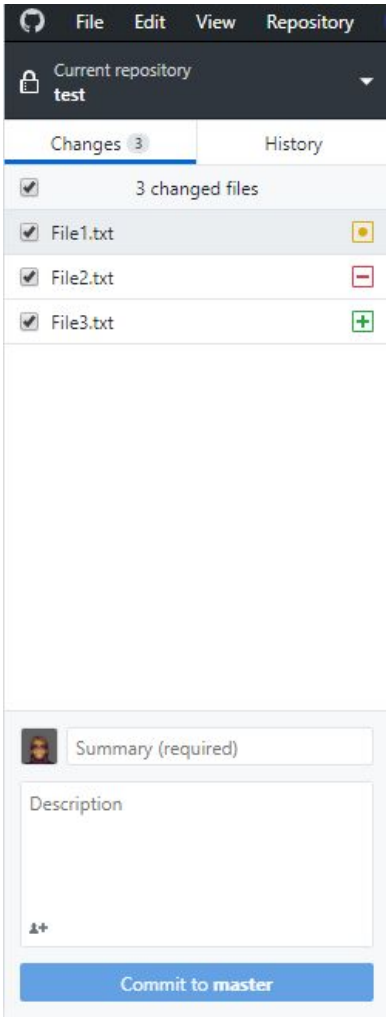


After this, your local repository has the most recent version of the project at origin.

Workflow for committing/pushing

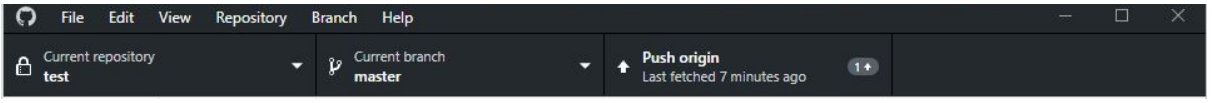
After we've finished working or something, we need to create a commit to store our changes and push to the origin. Pushing to origin means that we're uploading our snapshot of the project to the Git server, and now it will be the most recent version of that branch. Therefore our teammates are able to update their local repositories and receive our changes.

1. We can choose what changes we want to commit. This is done by checking the boxes related to the files we want to include. Only the files that have been checked will be included in the commit, so it is possible to keep some changes only locally. The icons next to the files are related to the type of changes: Yellow circle is a change in the file's content, Green + is a new file, and Red - is a file that has been deleted.



2. After selecting the items, we need to give a summary of the commit (usually a short descriptive name, like "Add Jumping feature"). If wanted, we can give a more in-depth description of what the commit is about, but this is not required.

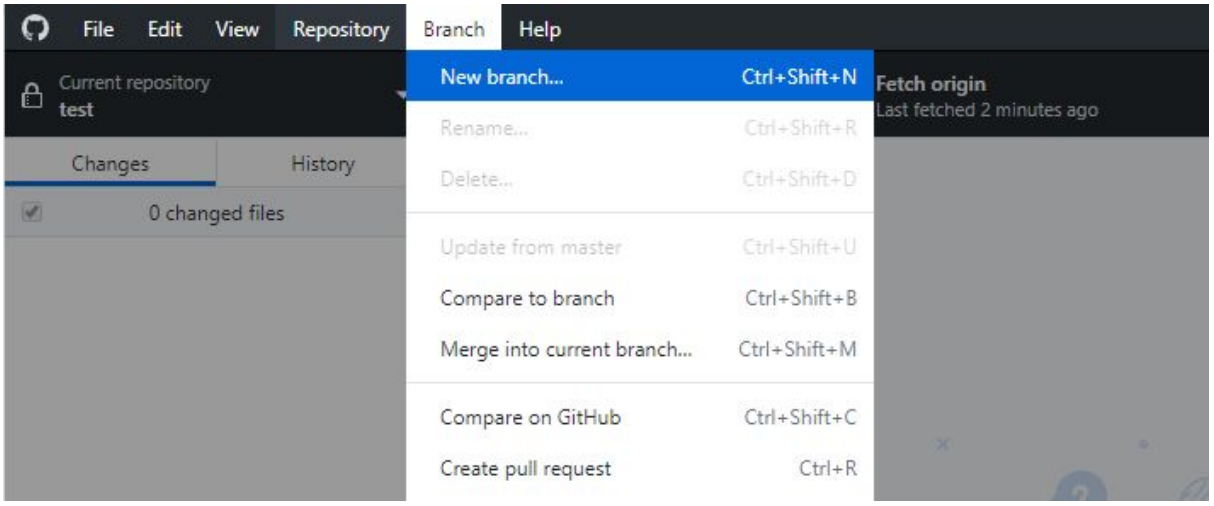
3. When we finish the commit, our snapshot is only local. In order for our teammates to have access to the changes, we need to push them. After commiting a change, the Fetch/Pull button turns into the "Push origin" button. The small number shows how many commits our local repository is ahead of the origin. By pressing the button, our changes are uploaded and now available in the remote server.



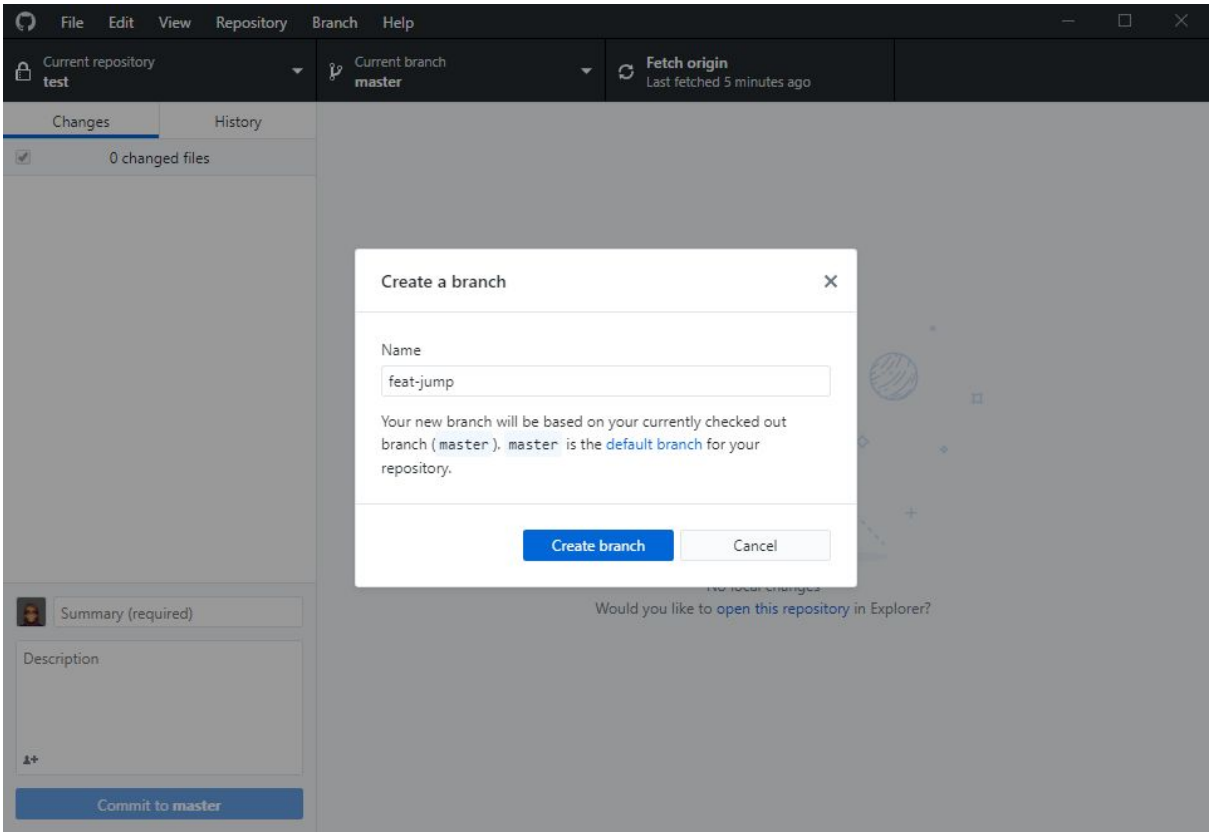
Workflow for branching

Branching allow us to create encapsulated versions of the project, giving us the opportunity to submit our work to Git (thus it's available to our teammates) without impacting the *master* branch. As a rule of thumb, the master branch should have only stable versions of our project.

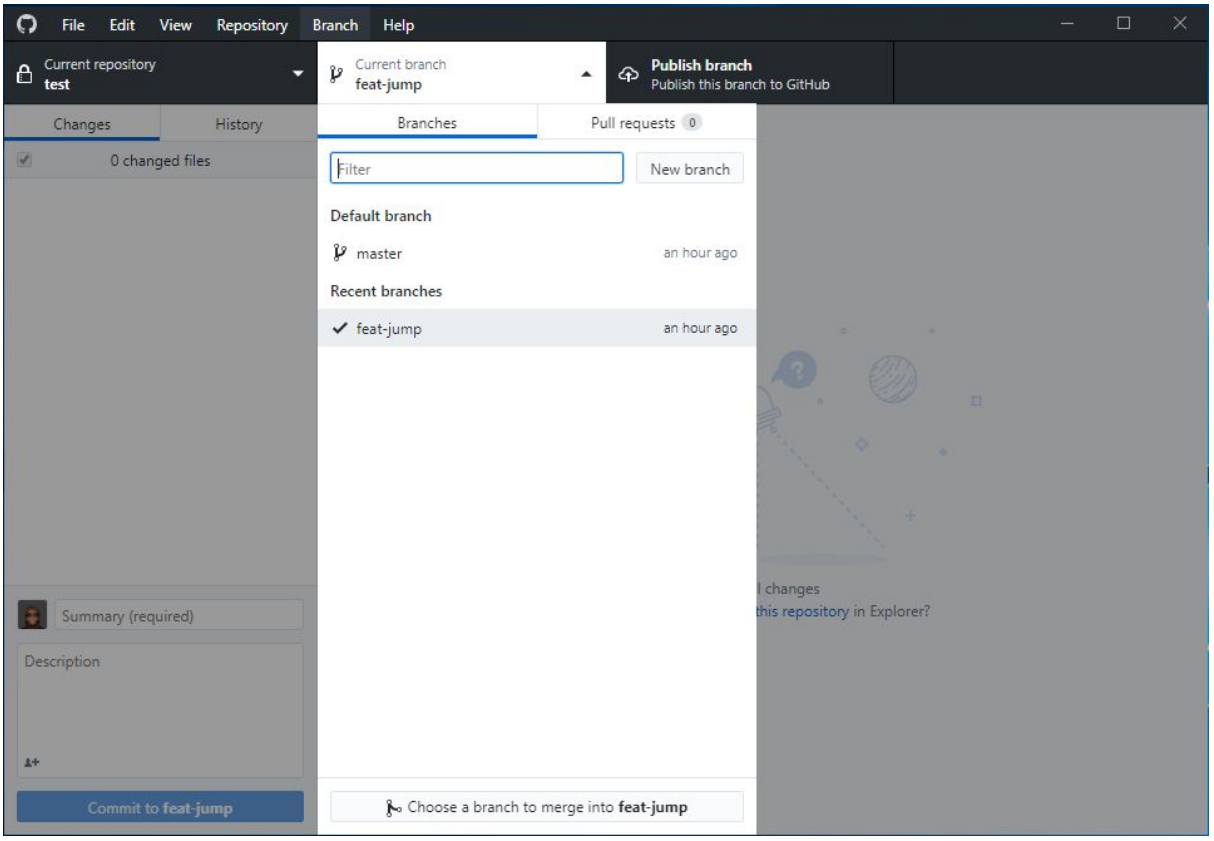
1. To create a branch, we need to go into Branch > New Branch



2. We need to name the new branch in order to complete its creation. Usually names are related to the purpose of that branch or to a specific name convention (either a feature that will be worked on or just "wip", or even the person that is working on it). There are no set rules, just conventions that can be loosely defined.



3. When the new branch is created, it will be indicated in the “Current branch” button. As the number of branches in a project grows, it’s important to make sure into which branch we’re committing our changes.
4. The “Publish branch” button has now appeared, and by clicking on it we push this new branch to the origin. We can either do this straight away, or do some changes and commit them together when we set up the new branch.



To **switch between branches**, we just need to select it in the Branches tab. When you do so, Git tries to update your local files to the last commit of the branch you switched to. If the same file has been changed in both branches, you won’t be able to change as that would override your local changes. You either have to commit the changes first or discard them.

To be updated -> Merging branches and solving conflicts
(I’ll add those as they happen in our project)

Glossary

- Origin: refers to the project located remotely, i.e. on GiHub's server.
- Local: refers to the project located locally, i.e. on the user's PC.
- Commit: the name given to the snapshot of the project

Everyday commands

Fetch is used to look at the origin to see if there are any changes. It automatically downloads commits and references files.

To update your local repository with the version in origin, you need to fetch, and if there are any changes, you can ***pull*** them. Pulling downloads the changes and updates your local branch to the most recent commit.
In case there are changes in the same file both local and remotely, pulling won't work. This prevents your work from being overridden unintentionally.

In order to commit our changes, we need to ***add*** them to the list of tracked files. With the visual UI, this can be done by checking the checkboxes

Commit is the command to take the snapshot of the files we added to the list. It's recommended to give a short descriptive name to what changes were made in your commit, but a more thorough description can also be added.

For your commit to be added to the origin, you need to ***push*** your changes. This adds the commit to the remote branch, allowing others to fetch for your changes.

A ***branch*** is used to encapsulate work, especially if it would mess with the stability of the project. If you want to commit your changes into a new branch, remember to create the branch before committing. Otherwise, the commit will be done in the current branch.

Merge is used to integrate the changes of a branch into another one. It creates a new commit in your current branch with the changes from the other one.

If anyone would be interested, here is a nice tutorial for command line use.