**GIT TUTORIAL:**

TERMINOLOGY:

**Local repository** – a local repository is a copy of the entire project’s codebase and history that resides on an individual’s machine. It is useful for tracking versions, being able to go back and forth from versions, and if a new added feature or code provokes a failure, you can always go back to the last version you committed.

**Remote repository** – A remote repository is a place where you can save your project and track the different versions committed to it, but the saving happens on the Internet or network somewhere. A remote repository is used to share a project with your colleagues. versions of your project that are hosted on the Internet or network somewhere. You can have several of them, each of which generally is either read-only or read/write for you.

**CLI** – Command Line Interface: A command line interface (CLI) is a text-based interface where you can input commands that interact with a computer's operating system.

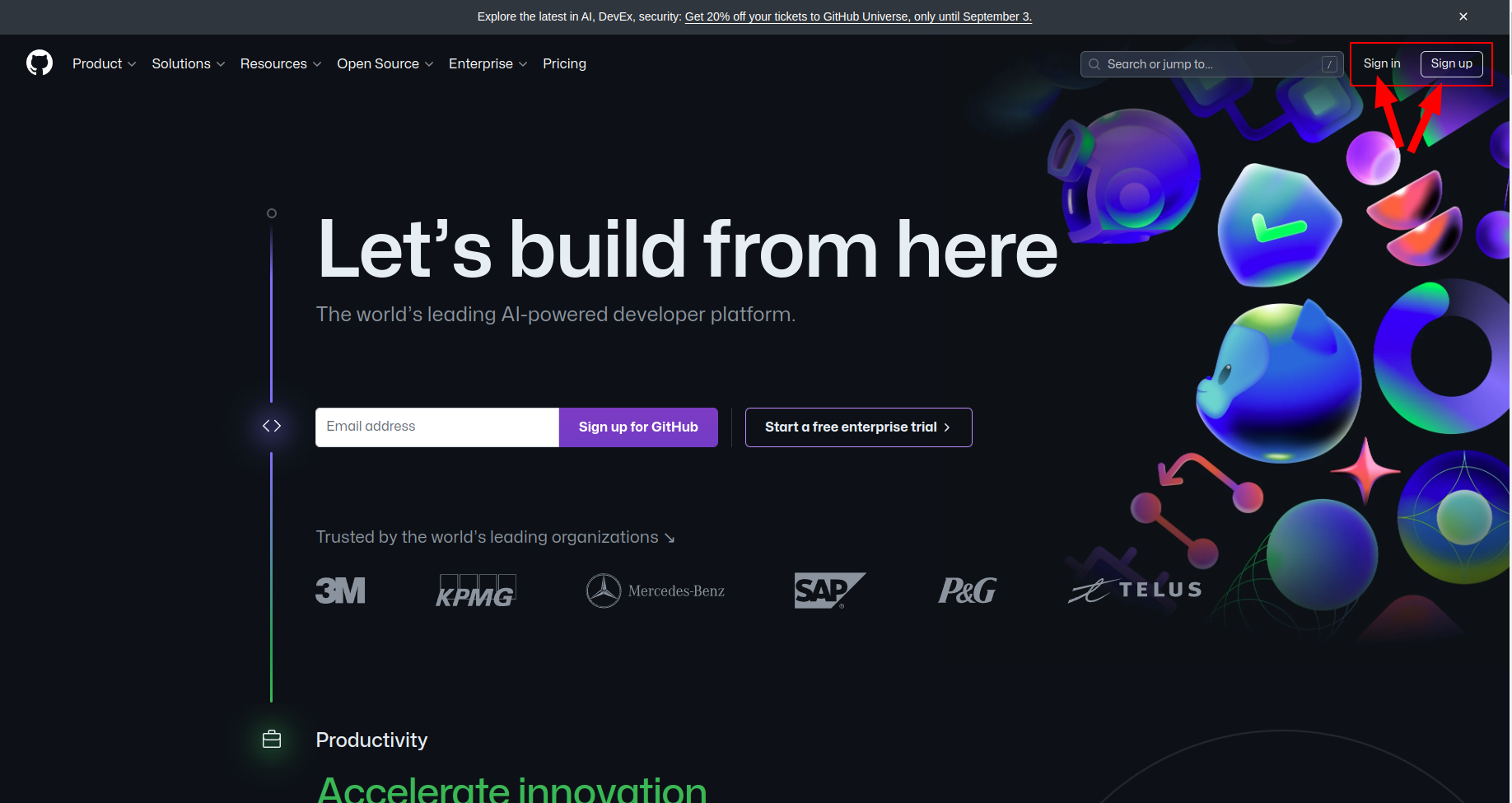
**Commits** – Like saving a file that’s been edited, a commit records changes to one or more files in your branch.

**Branches** - Branches isolate development work without affecting other branches in the repository. It’s like copying folder A, naming the copy folder B and saving your work on folder B without affecting folder A.

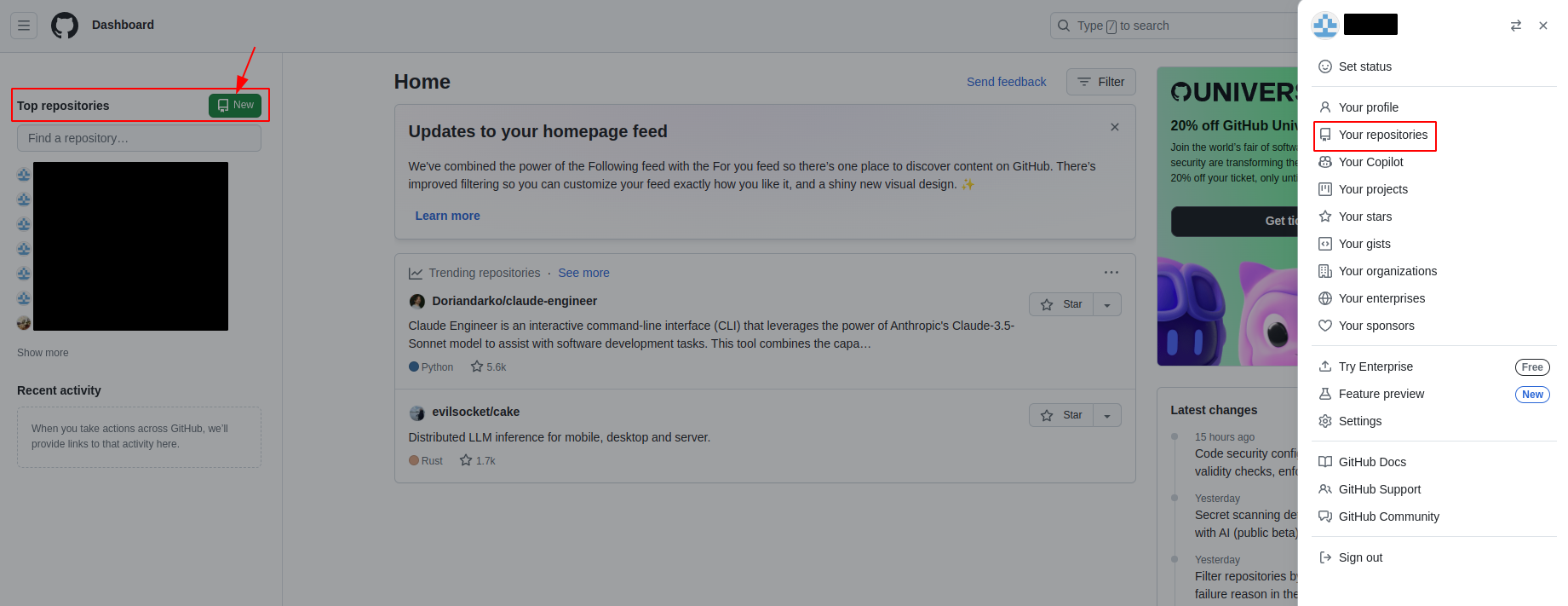
**PAT** – a Personal Access Token lets you push, pull or clone remote repositories to/from your local repository.

HOW TO LINK GIT REPOSITORY TO MATLAB

1. Navigate to github.com and sign in or create an account:

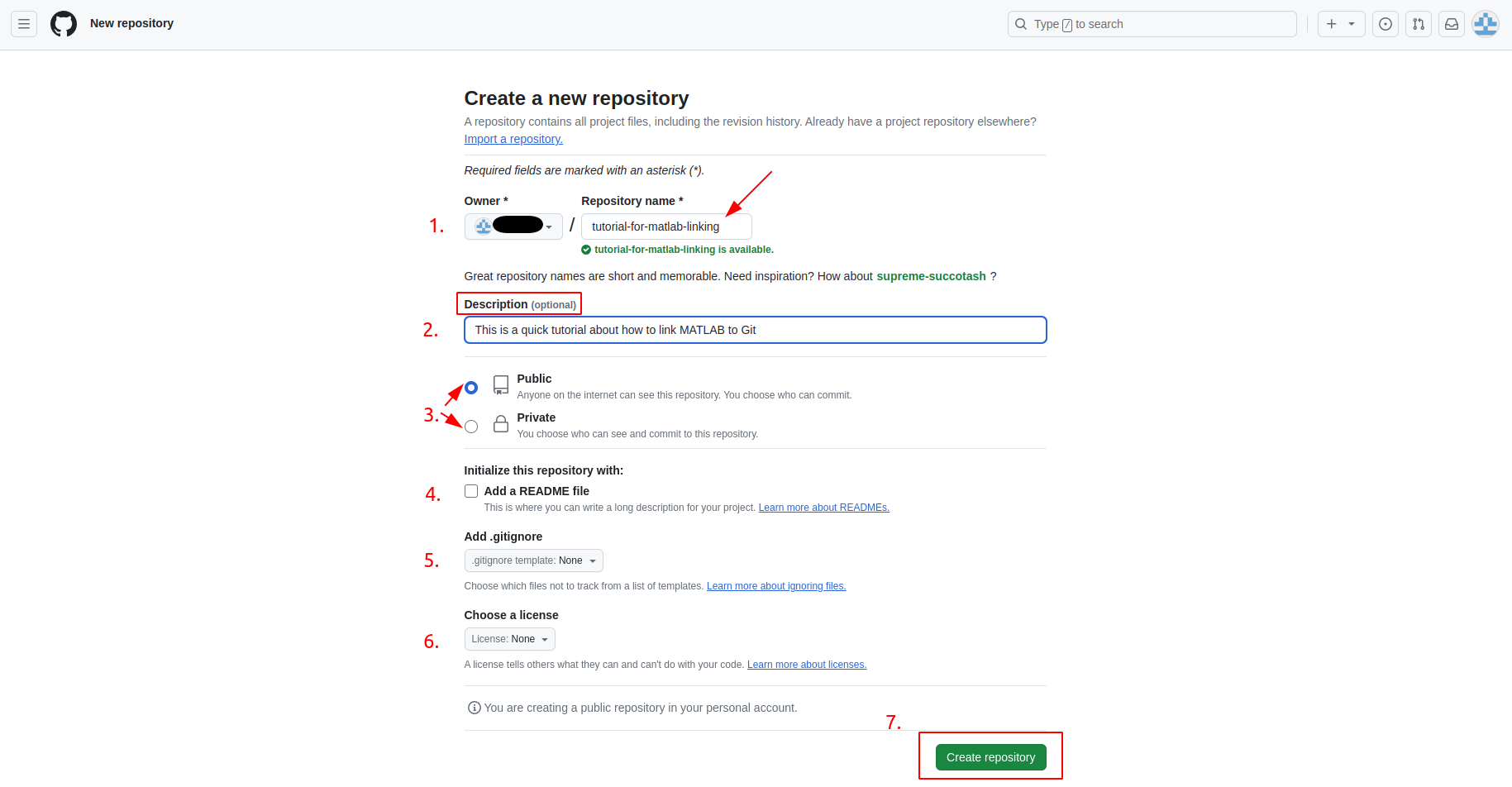


2. Once you are in, click on the left-upper green button “New” next to “Top Repositories” to create a new repository. Alternatively, click on the right-upper user icon, go to “Your repositories” where you will see all your repositories (if you have any). Then select the green button “New” on the right.

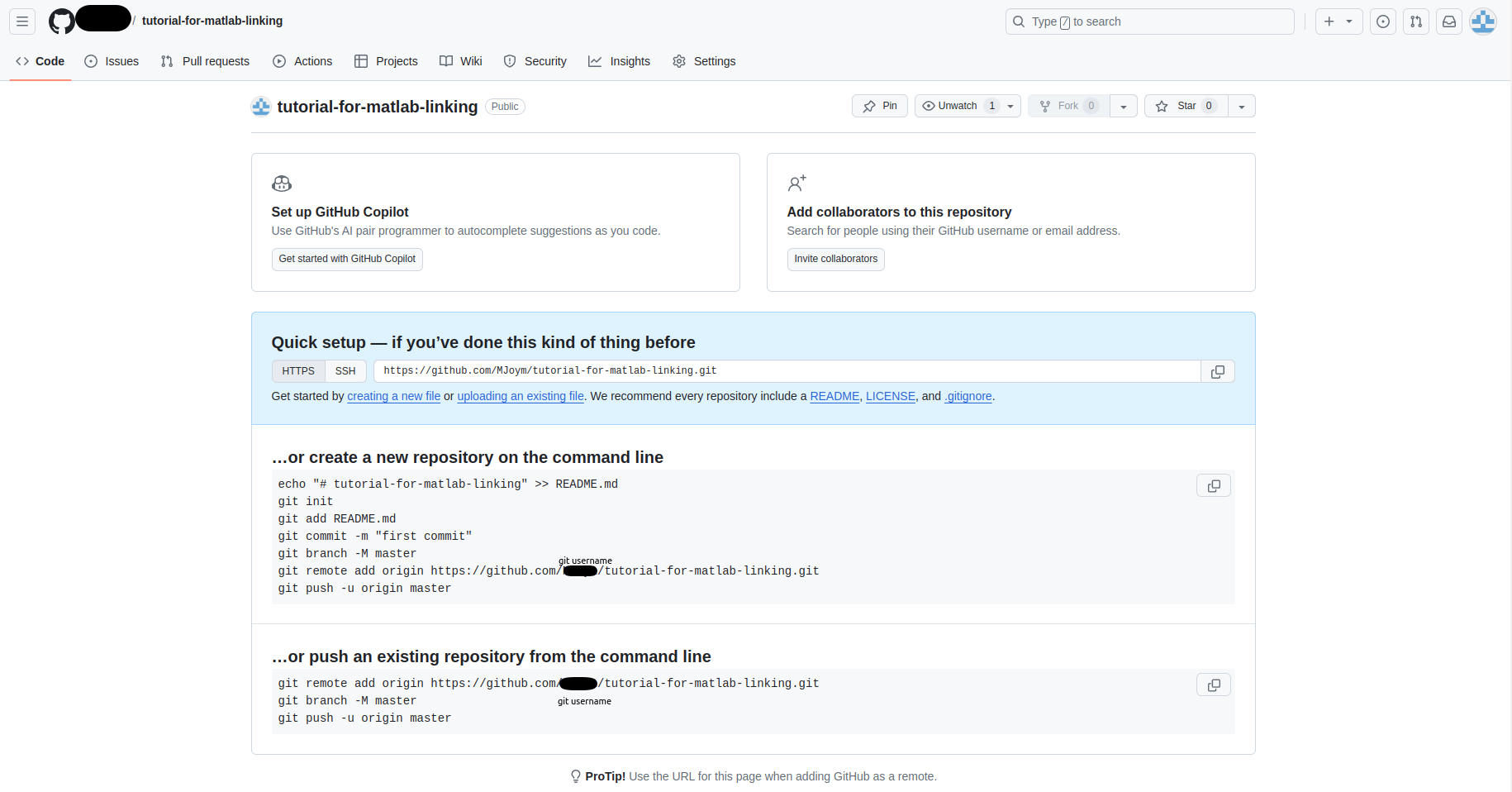


3. Choose a name for your new repository. You can add a description, as you can see in the image below (optional). The next important step is to choose who can see and commit to this repository. We will make this repository Public, as there is nothing to hide in it. You can add a README file, which is a file where you can explain in more detail about your repository, what is it about, how does it work and how to use it (optional).

Next, you can choose to add a gitignore file. This file can help you determine which files to consider uploading to your remote repository and which files to ignore. We will see an example in the next steps. You can also choose a license, here we will leave all these fields empty. If we will want to create a gitignore file or README later, we can.



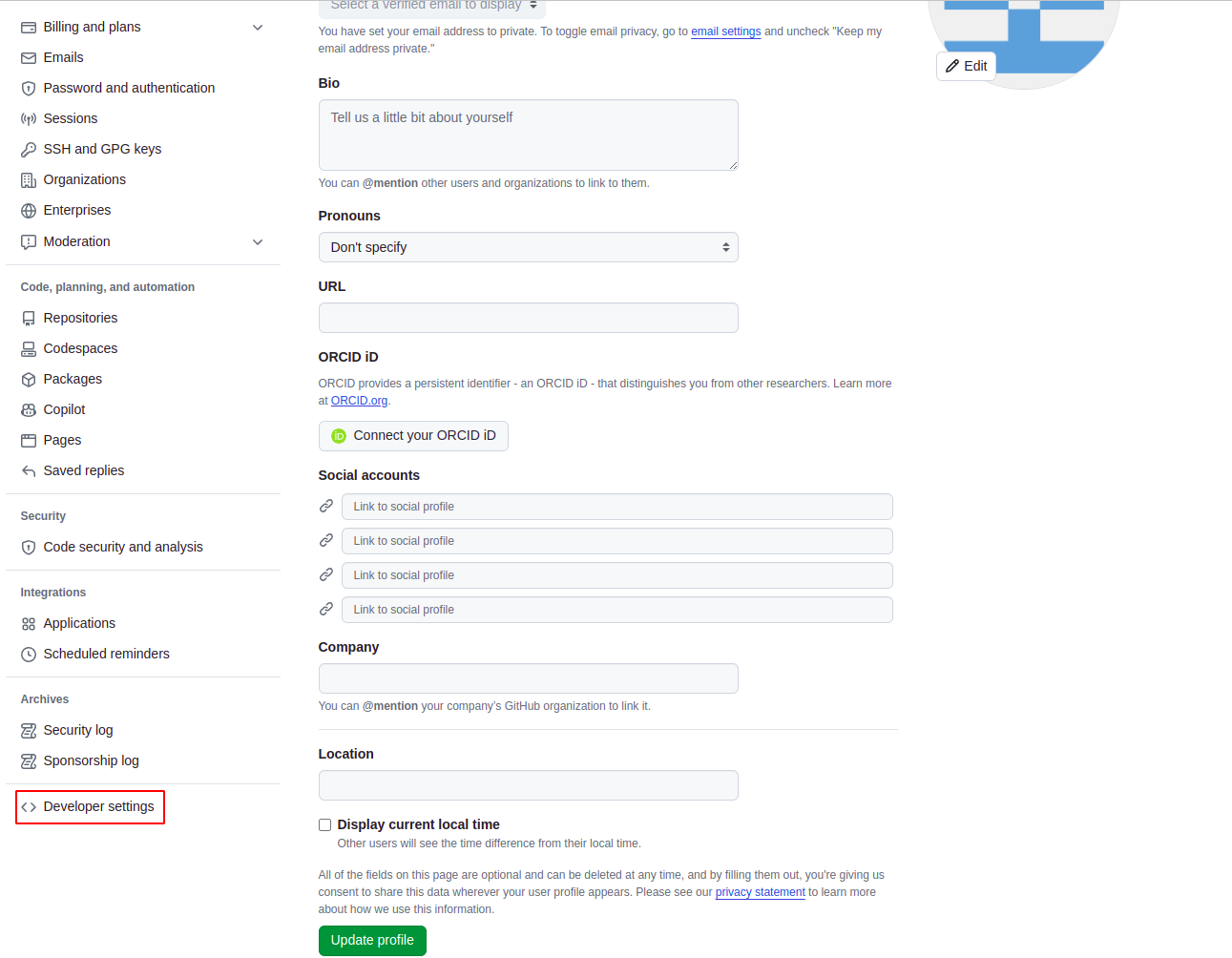
Click on “Create repository”.

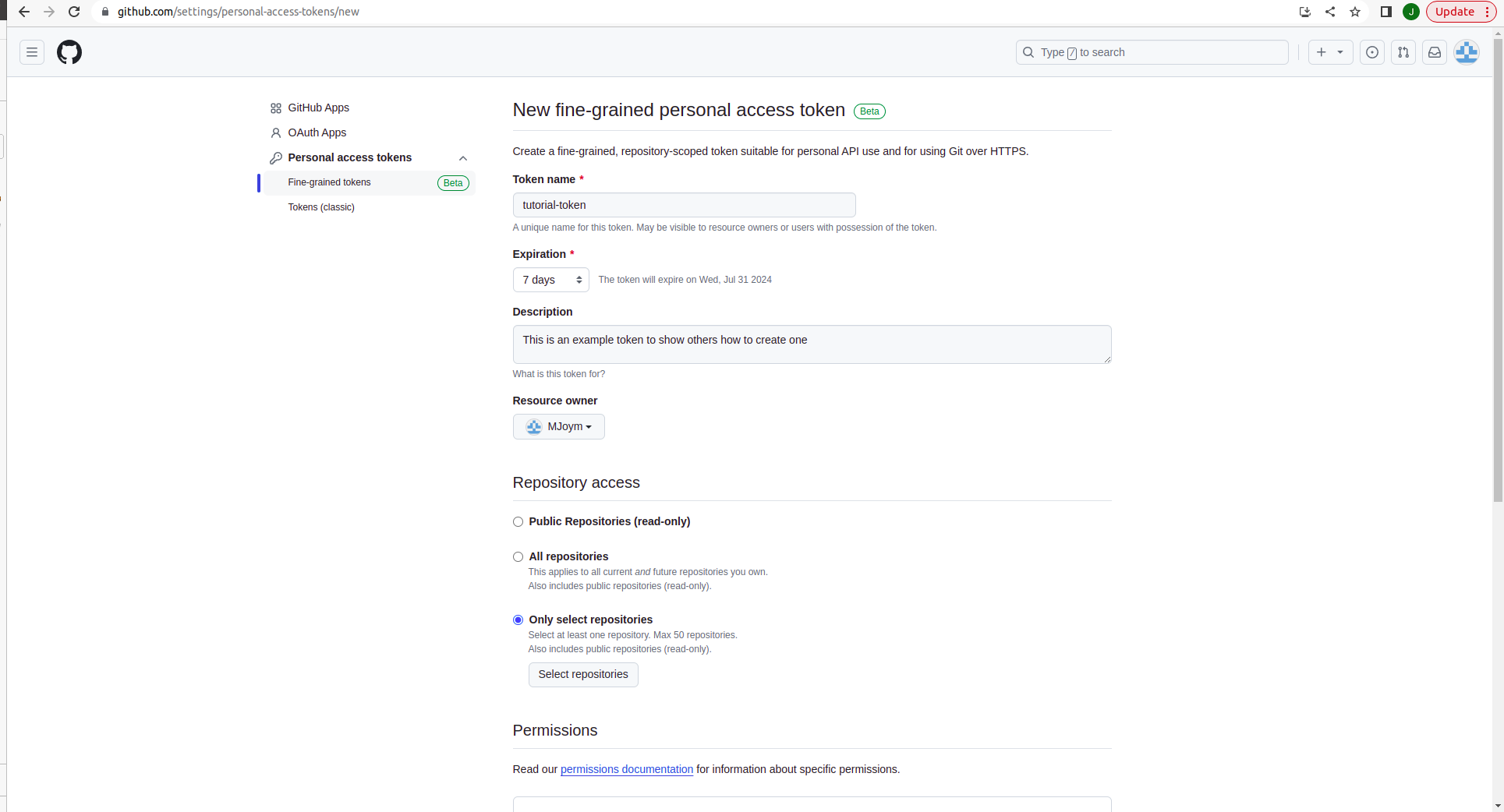


Now that you’ve created a new git repository you might see this page, where git offers you different examples for future needs. Like how to commit, push, add a README file, add a gitignore file, etc. using a CLI.

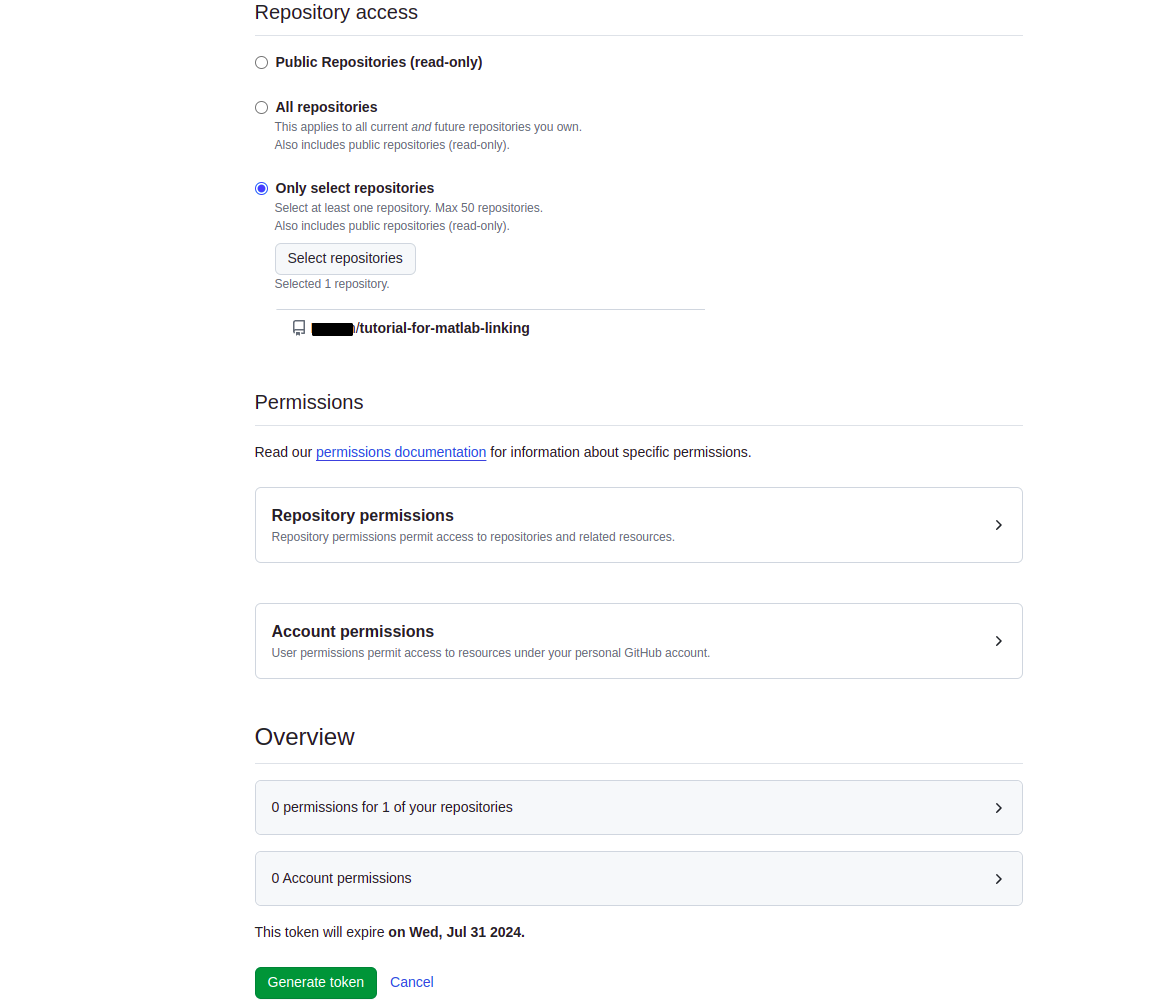
4. We will leave this page for now, and create a fine-grained personal access token (PAT). This token will serve us to pull/push files from our local repository to our remote repository.

For this, under your user picture (on the top right hand of your screen on Github web page), go to ‘settings’ and then select ‘Developer settings’ at the bottom of the left-side menu.



Now that you are in the Developer Settings, choose “Fine-grained tokens” under “Personal access tokens”. Give it a name, expiration date (for security reasons it is preferable) and a short description.

Choose if this token is for public-repositories (you can only clone them)/All your repositories/Select specific repositories. I think it is best practice to select specific repositories, for security measures, but you can work how you choose to.

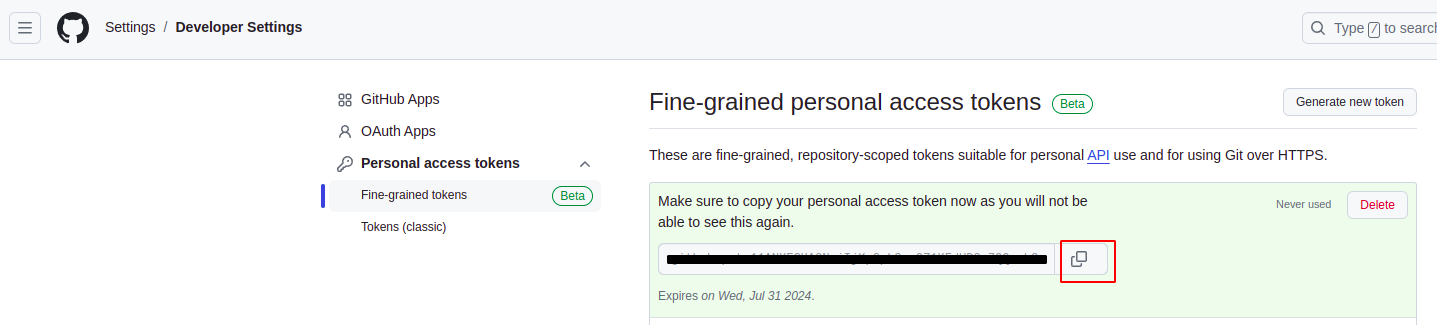


Next, give the necessary permissions to your PAT (Personal Access Token). It is not recommended to open all permissions to a git repository for security concerns. We added the read-write permissions to the ‘Administration’, ‘Commit statuses’, ‘Contents’, ‘Deployments’, ‘Environments’ and ‘Pull requests’ under “Repository permissions”. Make sure to read all of the permissions to see if you need any extra.

Once you click on ‘Generate token’ you will be given your new PAT.

\*\* Make sure you save it somewhere safe on your PC. Click on the ‘copy’ button next to your PAT and paste it somewhere (notes, word, etc.). We will use this shortly.

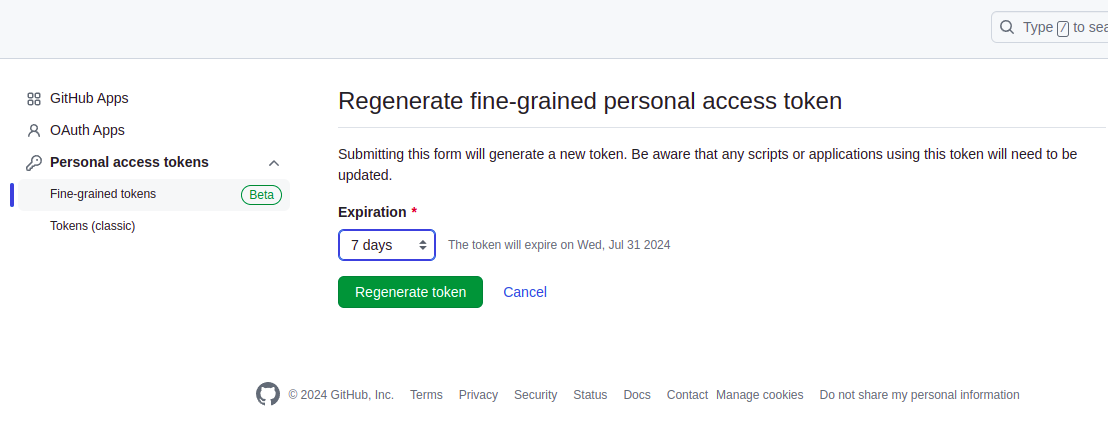
\*\* NOTICE THAT ONCE YOU LEAVE THIS WINDOW YOU WILL NOT BE ABLE TO COPY THIS PAT AGAIN! SO SAVE IT NOW.



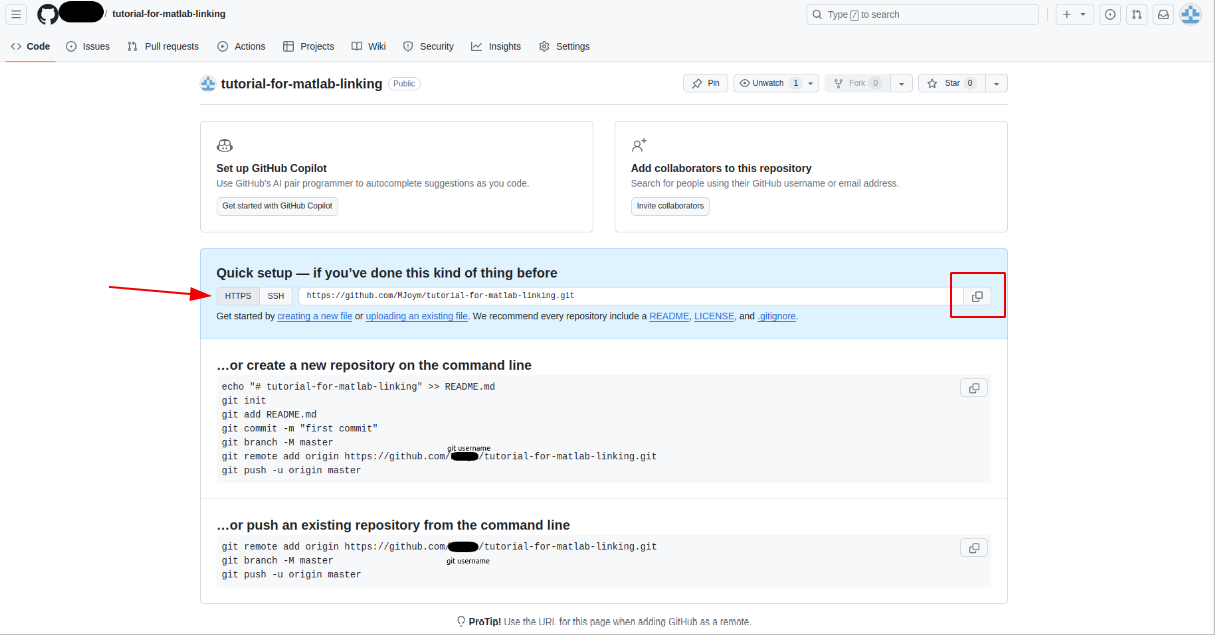
REGENERATE A PAT:

If, although the warnings, you lost your PAT, do the following:

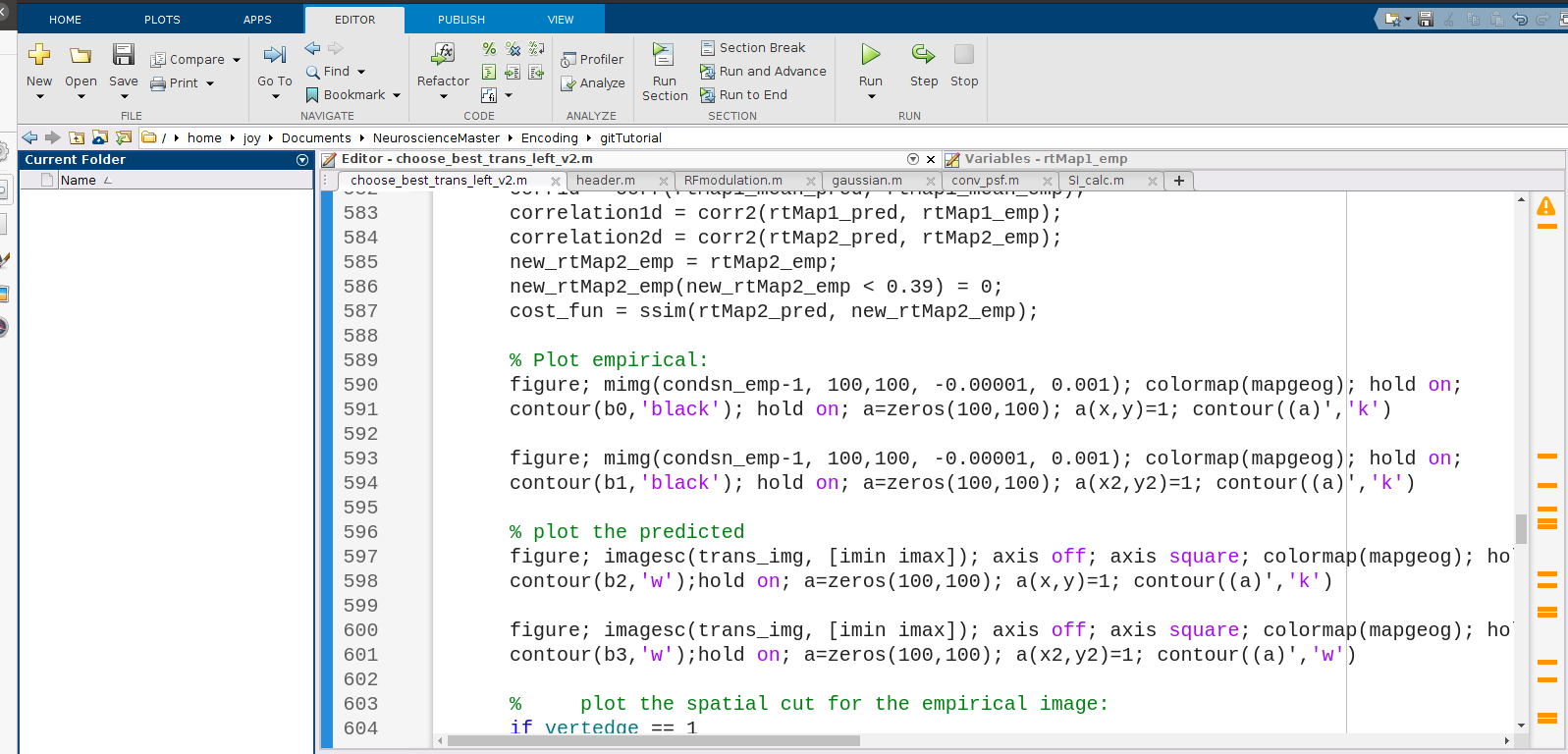
Go to Developer Settings → Fine-grained tokens → choose the relevant token from your list of tokens → and click on “Regenerate token”. Make sure to copy and save this new token.



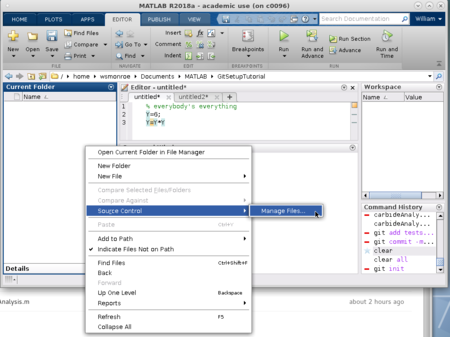
5. Now lets go back to our repository on Github. The one thing we need from here is the HTTPS path of our repository so copy the path using the ‘copy button’ on the right.



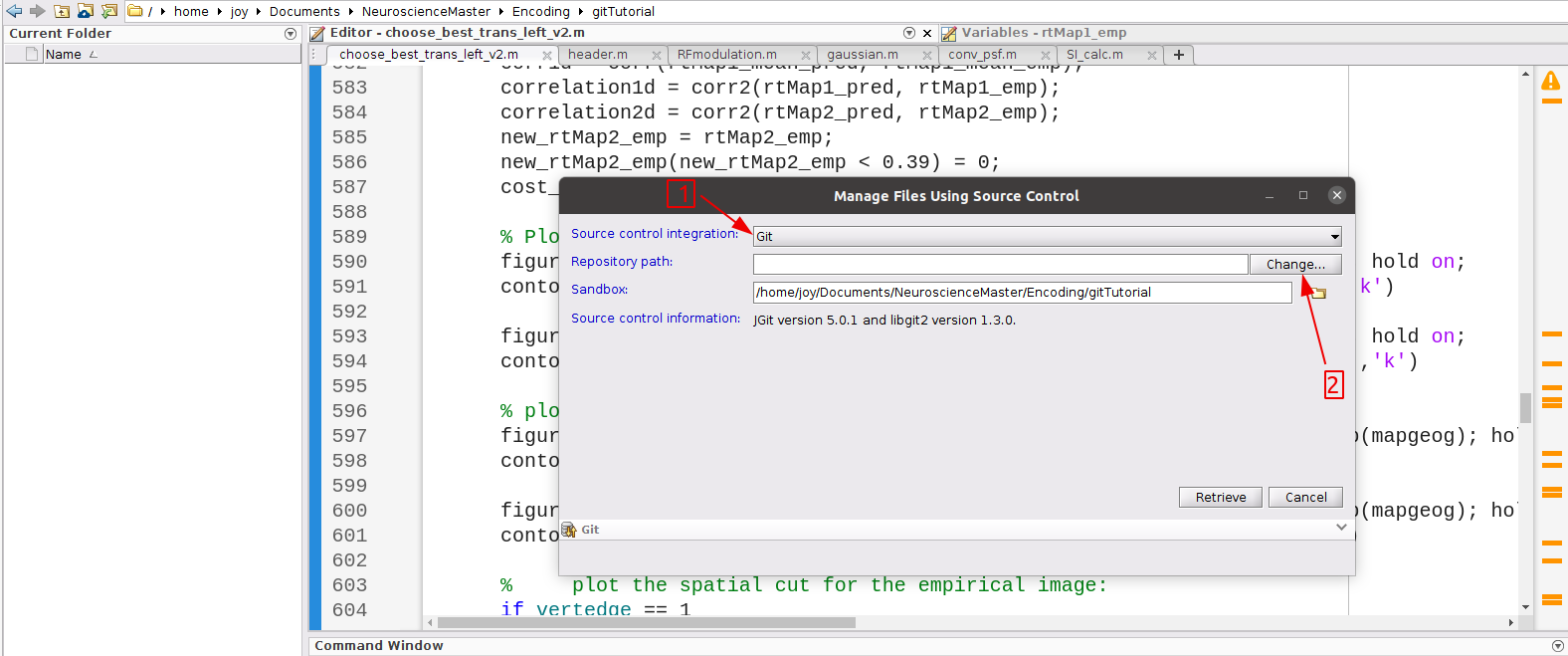
6. Go to your MATLAB. Navigate to the path you want to create the git linkage. Usually it will be in your project folder you want to upload/save/work with. To start, MATLAB's Git requires an empty folder (a regular MATLAB folder).



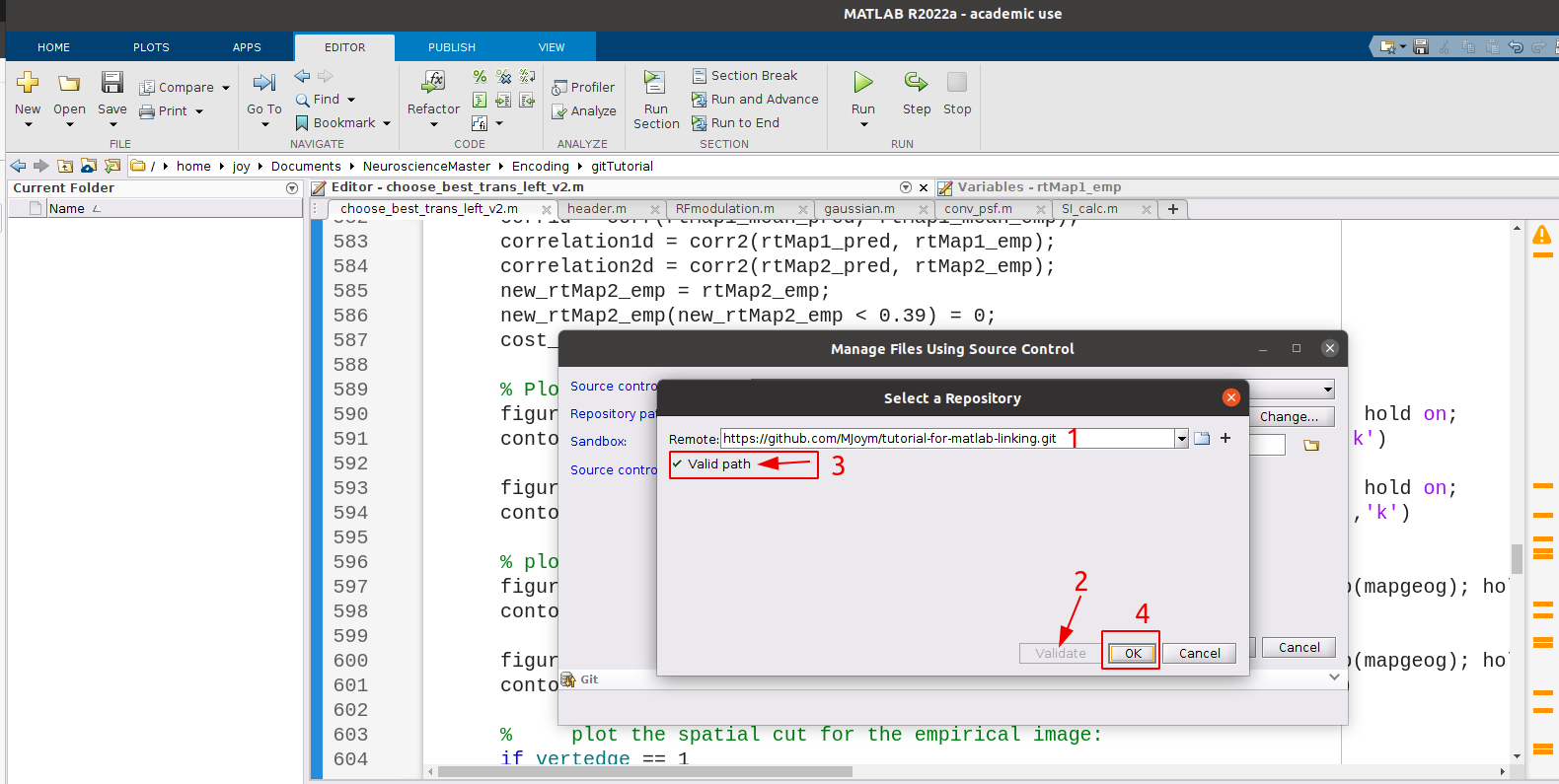
Now, right click on the “Current Folder” section (which is on the left of the above image) and select “source control → manage files.



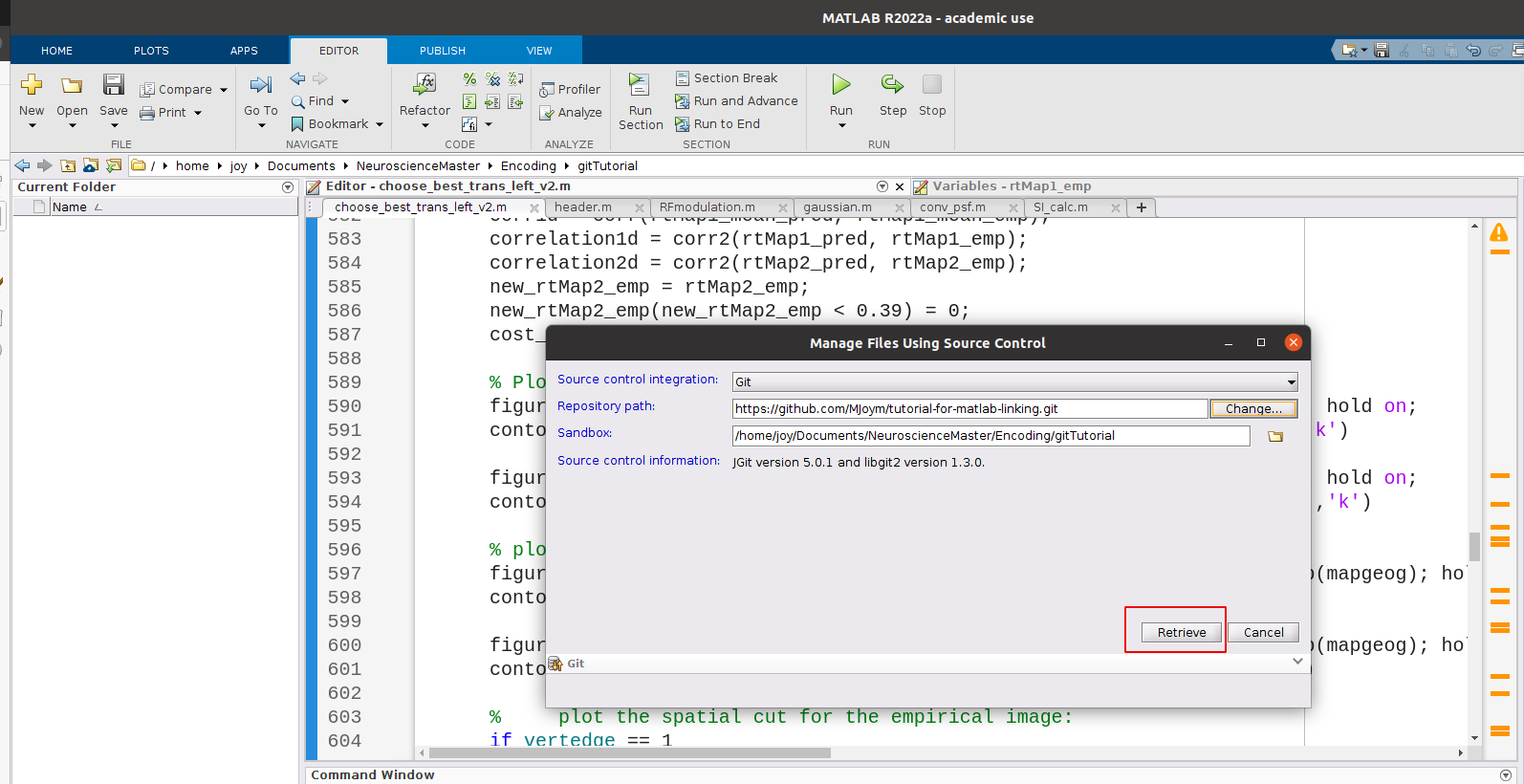
A pop-up window will open called “Manage Files Using Source Control”. Select ‘git’ in “Source control integration”, paste the HTTPS repository path we copied in step 5. For this, click on the “Change…” button under “Repository path”:



Paste the link and click on “validate”. MATLAB will take a few seconds to validate the path and once it is validated, click on the “OK” button:



Finally, press the “Retrieve” button:



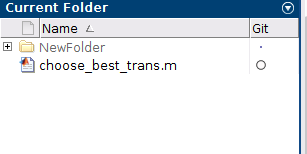
ADDING, COMMITTING AND PULLING FILES TO/FROM REPOSITORY:

Adding files:

When you add a file to Git, you are essentially telling Git that you want to start tracking the changes made to these files, or in other words saving “versions” of these files. In Git terminology, this would be “staging” the files, which refers to the process of preparing changes to be included in a commit (see in next step what is a commit). You can decide which files not to add in your project if you do not want to make them accessible in your remote repository. But these files wont be saved in your local repository history either.

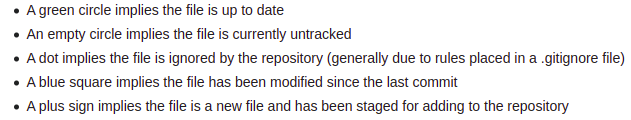
7. To add files to your repository, copy and paste the relevant files or folders within your MATLAB console or in your File Explorer to the Git project recently created.

You can create new folders within your Git folder project to organize your files in the repository.



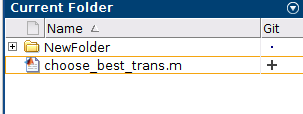
Here we added only two files as an example (‘choose\_best\_trans.m’ and another file inside ‘NewFolder’).

You will notice that there is a new column now next to the files’ name, called ‘Git’. This column shows the status of the files.



8. Right-click on the file you want to add to your repository, select ‘Source control’ → ‘add to git’.

Now you will see the plus sign, indicating that the file has been added to the repository.



But this does not mean that you will see it in your remote repository yet. For that you need to commit the file and push it.

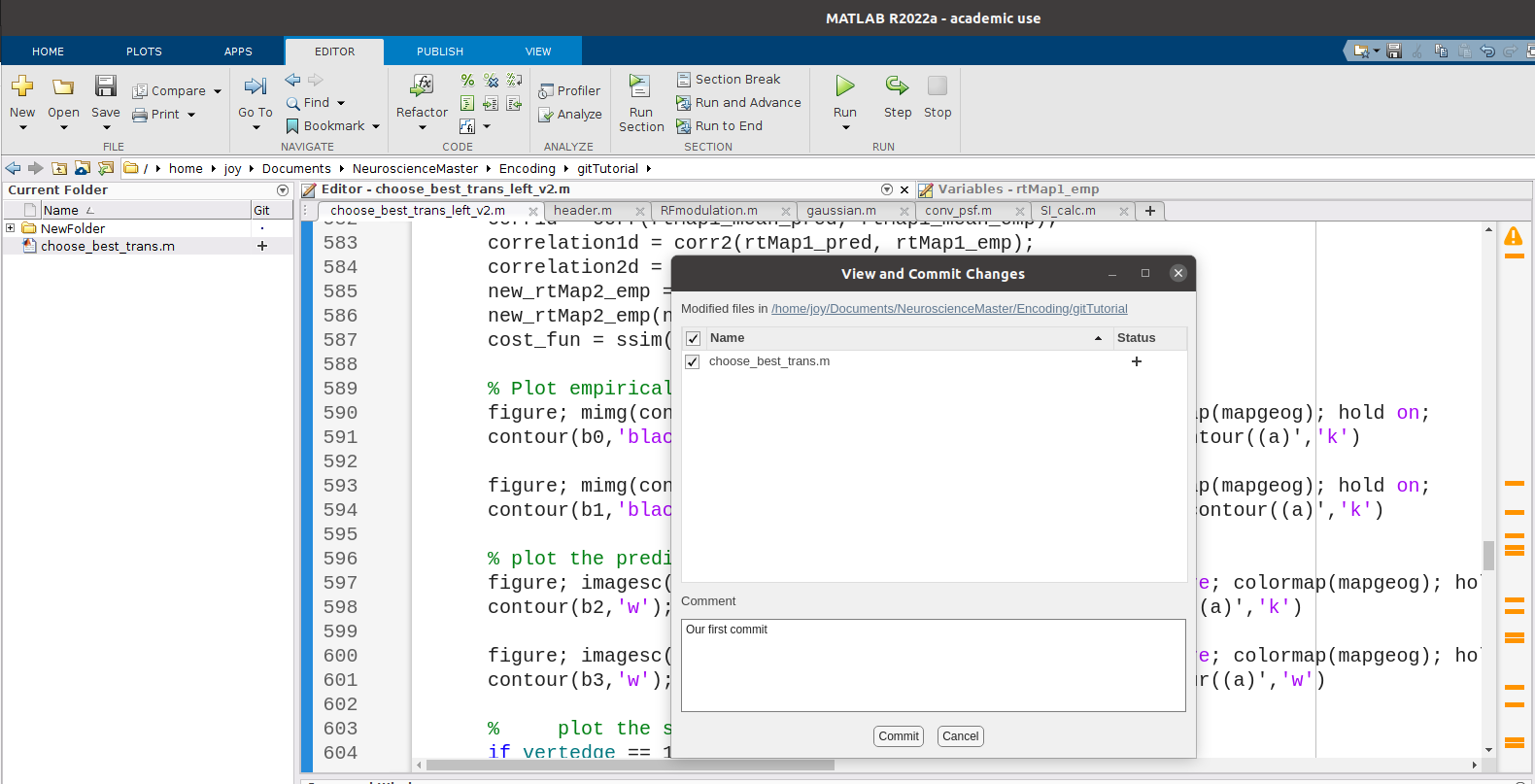
Committing a file to Git:

A “commit” is like taking a snapshot of all the changes you performed on you file/s at a specific moment in time. It’s a way of saving your work and keep track of what you’ve done. Think of it as a checkpoint that you can refer back to it or revert to if needed. This commit is first saved to your local Git repository on your computer. To make your files accessible in your remote repository you need to push these commits.

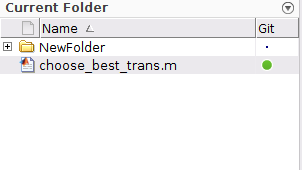
\*\* Don’t forget to commit your changes after finishing your work!

9. Right-click on the file with the plus sign, choose ‘Source Control’ → ‘View and commit changes’.

Here we have one file, called ‘choose\_best\_trans.m’ selected for the commit action. Notice that we must add a comment to our commit in order to successfully commit our file. This helps us be organized and understand what each commit added/changed/deleted from our files. Try to be expressive and precise in your commit comments. Trust me, it will help you in the future if you find an error and want to go back to an older version of your code.



You’ll now notice that the Git column shows a green dot, denoting that this file is being tracked. Now we need to push this file to our remote repository in order to see it there.

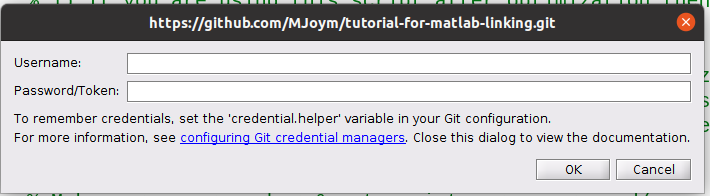


\*\* Commits happen when you changed your code, it works and you want to save it.

Pushing a file:

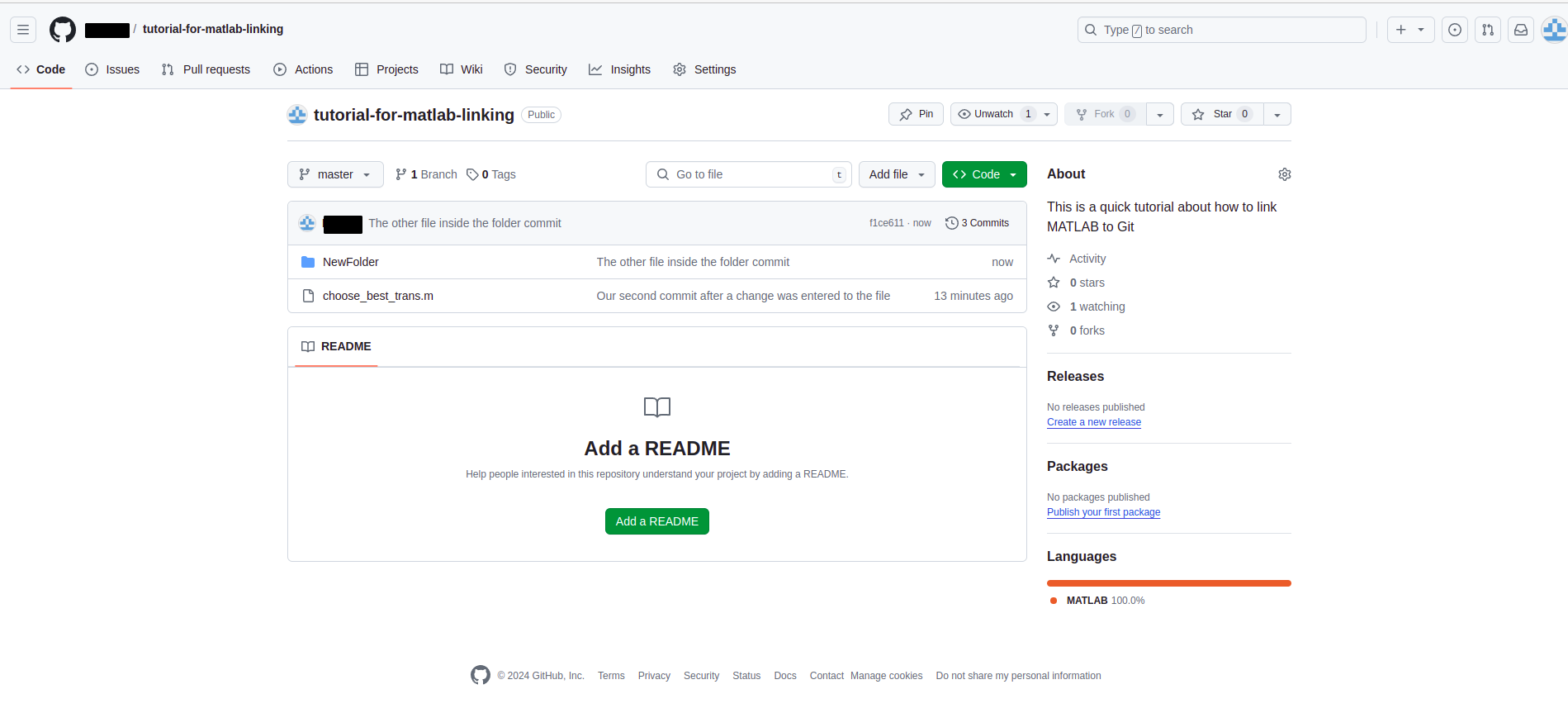
\*\* PUSHING A FILE TO YOUR REMOTE REPOSITORY HAPPENS ONLY WHEN YOU KNOW YOUR CODE WORKS AND HAS NO ERRORS. ESPECIALLY WHEN YOU WORK WITH COLLEAGUES ON THE SAME PROJECT!

10. Right-click on the committed file/s and select ‘Source control’ → ‘Push’. A pop-up window like the one demonstrated below will show up.



11. Enter your e-mail address linked to your Github and your fine-grained PAT that you created and saved in step #4. Click ‘OK” and wait until MATLAB pushes your file to your remote repository.

Now you can go back to your remote repository in the web and see that indeed this file is accessible. Proceed with all the other files that you need to add to your repository. Another way to do this easily is to right-click on the blank space on the ‘Current folder’ window (not on specific files) and add, commit and push everything at once.



EDITING A FILE THAT IS ON YOUR REMOTE REPOSITORY:

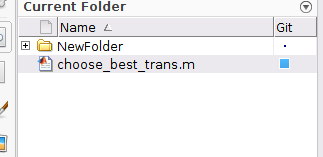
Now that you added all your files, we will learn how to maintain your Git repository up to date after changing a file. Suppose you arrive at the lab in the morning, after working the day before on your code, when you added, committed and pushed your files to your new remote repository. Today, Hamutal gave you some more ideas and analyses to perform and so you keep adding changes to your code. The code in your remote repository is not longer up to date with the code you changed on your local computer. Even a more real situation is when you work with other colleagues on the same project, and one of your colleagues made changes the night before. Now the code you have on your local computer this morning, without even starting to change what Hamutal told you to, you are not up to date with the remote repository and your colleagues. So a good practice is to every day make a pull request from your remote repository. If no changes have been made, then everything will remain the same in your local code, but if there were changes pushed then you will make sure you are up to date.

*“Git pull* updates your current local working branch and all of the remote tracking branches (we will see what are branches in the next steps, right now our branch is the default branch created by Git, called *main*). Without git pull, your local branch would not have the updates that were made on the remote.” - for more information see References github.com/git-guides/git-pull

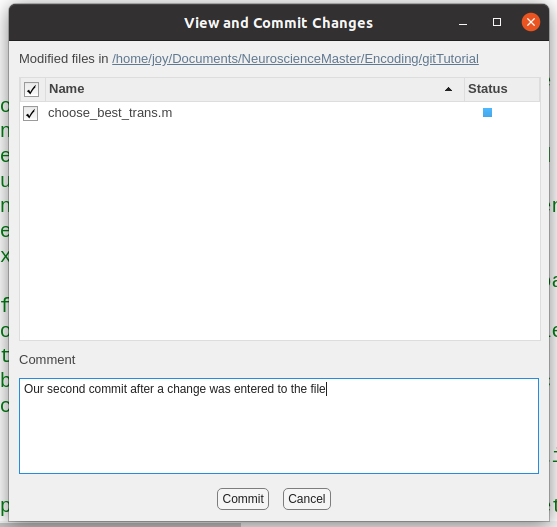
Pull requests:

Right-click on the ‘Current folder’ window (not on a specific file! But on the background) and select ‘Source control’ → ‘Pull’. Wait until MATLAB pulls all the changes from your remote repository into your local repository. As simple as that!

You will notice that after your new changes, the MATLAB Git column with show the file with a blue square:



This means that the file has been modified since its last commit. What you need to do now is repeat the commit action as you learned, and push the file to the remote repository.



After pushing, the green dot will return.

\*\* If you are the only one managing the remote repository, a pull request is not really necessary, since you know no one performed changes.

\*\* Changes that were not committed can be overwritten by a *git pull*. So make sure to commit your changes!

\*\* Pull requests for the main branch is a good practice if you are working by yourself. Otherwise, mandatory to keep the repository up to date and free from conflicts.

\*\* *Git pull* updates only the current branch, not all of the existing ones!

If many people are managing one remote repository, it is very recommended to work with branches.

Even when you are the only one on your repository, I would recommend working with branches. It provides an additional method for maintaining organization.

WHAT ARE BRANCHES?

Branches allow you to develop features, fix bugs or safely experiment with new ideas in a contained area of your repository. In other words, branches isolate development work without affecting other branches in the repository. For example, when two different colleagues are working on the same repository, the commits and pushes that they perform can be on separate branches. This allows each colleague to push his/her changes to their branches, while the main branch remains unaffected. Only when the colleagues are sure their code works they can merge their changes to the main branch.

\*\* If you are more than one person on a repository, we want to remind you again that pulling before a commit and push is crucial and checking that your code works after this pull request is also important before committing and pushing to the main branch!

The main branch, which can be called master or main, is the default branch which Git creates automatically when you create a new repository. If you repository is public, the main branch is also the branch that Git displays to whomever visits your repository and the branch from which cloning occurs.

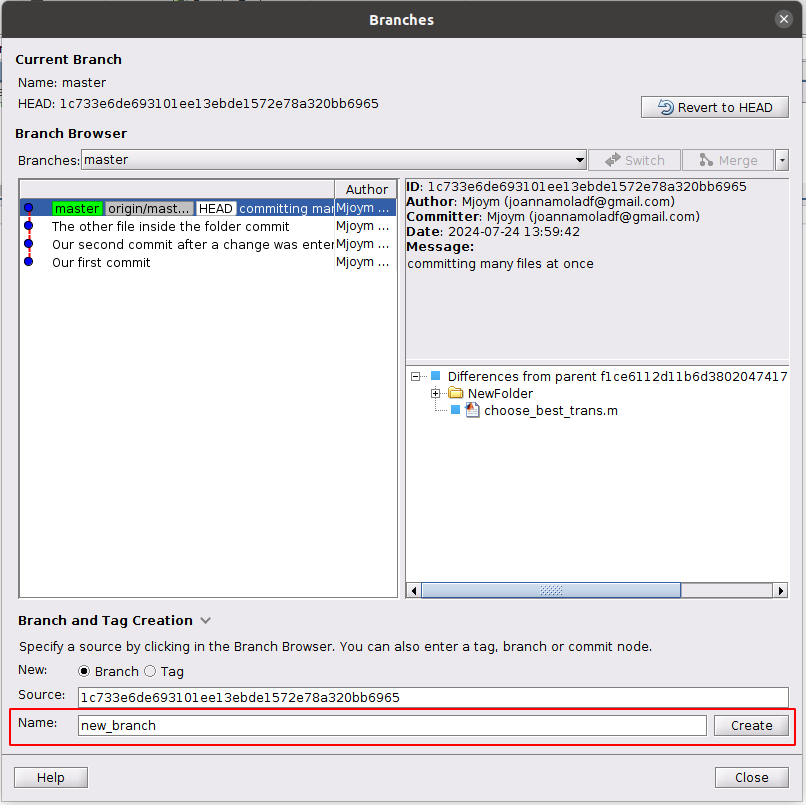
\*\* Working with branches can be a little misleading and complicated at first, so it is important to work carefully.

How to create a branch and work with it:

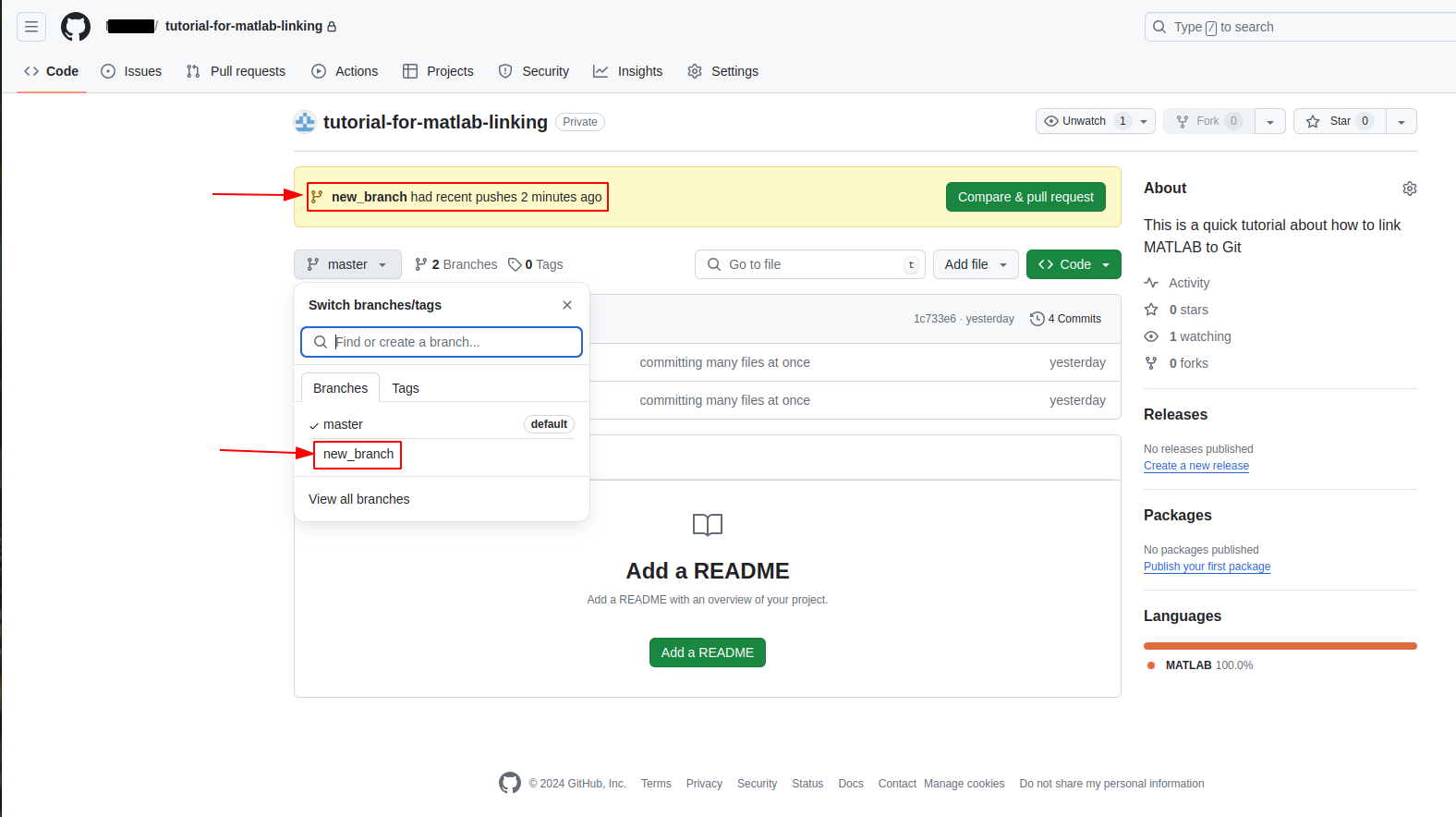
Before you start working/once you are satisfied with your work, right-click on the ‘Current folder’ window on the background (not a specific file) and select ‘Source control’ → ‘Branches’.

A window like the image below will open. At its bottom, select “Branch and Tag Creation”, select “Branch” under “New” and give it a name. Click on “Create”.

At the top of this window, under “Branch Browser”, select from the drop-down menu your new branch and select “Switch”. Close the window.



When you are ready, commit and push your files as we learned before. You will now see in your GitHub remote repository a notification on the top that a new branch was created and the amount of pushes that where performed. Your changes were pushed in your new branch, but, as we expected, your changes were not pushed to the main branch.



Merge a branch into your main branch:

We merge two branches when we want to unify two codes into one. When you merge your branch to the *main branch* you want to be sure that your branch has been updated with the latest changes from the main branch. This is because suppose you arrived this morning and continued working on *your branch*. Meanwhile your colleague finished working on his/her feature/s and committed, pushed and merged his/her changes to the *main branch*. Now you are not updated with the *main branch* anymore. You colleague’s new changes might produce conflicts with your changes. This is why before merging your changes to the *main branch*, it is important to merge the *main branch* into *your branch*. Lets learn how to merge and then learn how to deal with conflicts, avoid them, and so on.

Once your are satisfied with your code and you want to push your changes to your main branch into the remote repository, first merge the *main branch* into *your branch.* For this, go back to the branches window as we did before and select the *main branch*. This time you will press on “**Merge**” (which is on the right-side to “Switch”). If someone pushed his/her changes to the main branch before you did and you are outdated, you will see a window with conflicts. You will need to resolve these conflicts. Go one by one and resolve them carefully. After the merge (or the resolved conflicts), you will see your files updated, with the changes you performed on *your branch* up to date relative to the *main branch.* Now you can merge *your branch* to the *main branch*.

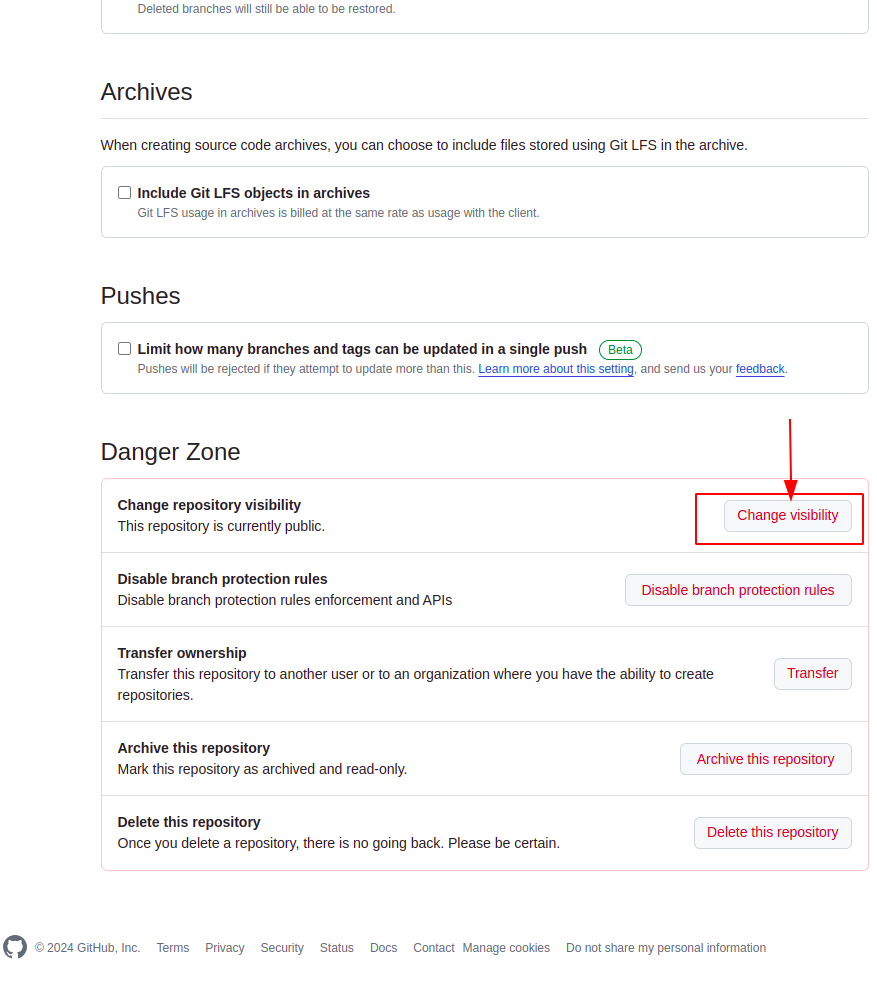
To merge *your* *branch* into the *main branch* go back to the branches window as we did before, **switch** to your *main branch*. Close the window. You will now see the code as it is in the *main branch*, without the changes you performed on *your branch*. This is good. Reopen the branches window and select *your branch*. This time you will press on “**Merge**” (which is on the right-side to “Switch”). Close the window. After the merge, you will see your files updated, with the changes you performed on *your branch*, now in the *main branch.* Now you can push your local main branch to your remote repository to update it with your new changes.

\*\* If you are working on your branch by yourself and not one else is supposed to change it, then pull requests are irrelevant. Otherwise, pull requests should be done every morning before continuing work.

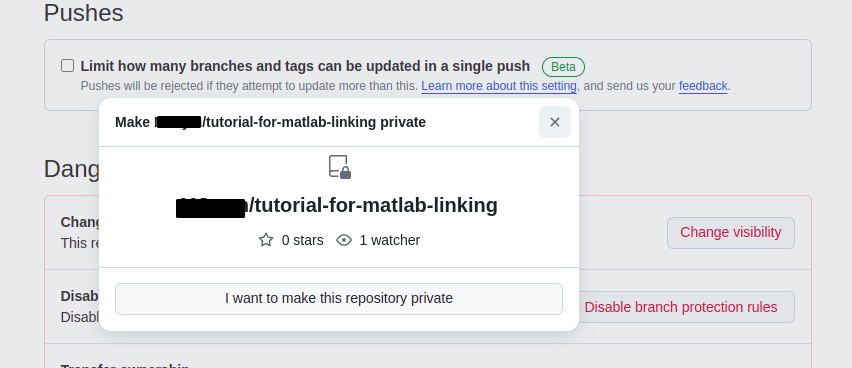
\*\* I do not recommend two people working on the same branch! Each person should have his/her own.

Changing your repository from private to public or vise versa:

Go to your remote repository in GitHub web page, go to “settings” in the top horizontal menu. Scroll down to the “danger zone” field and select “Change visibility”.



Git will warn you that this step is not recommended, you can read the warnings and decide wether you want to continue or not. If you decide to continue, select “I want to make this repository private/public”. You will now see the change in your repository when you go back to it.

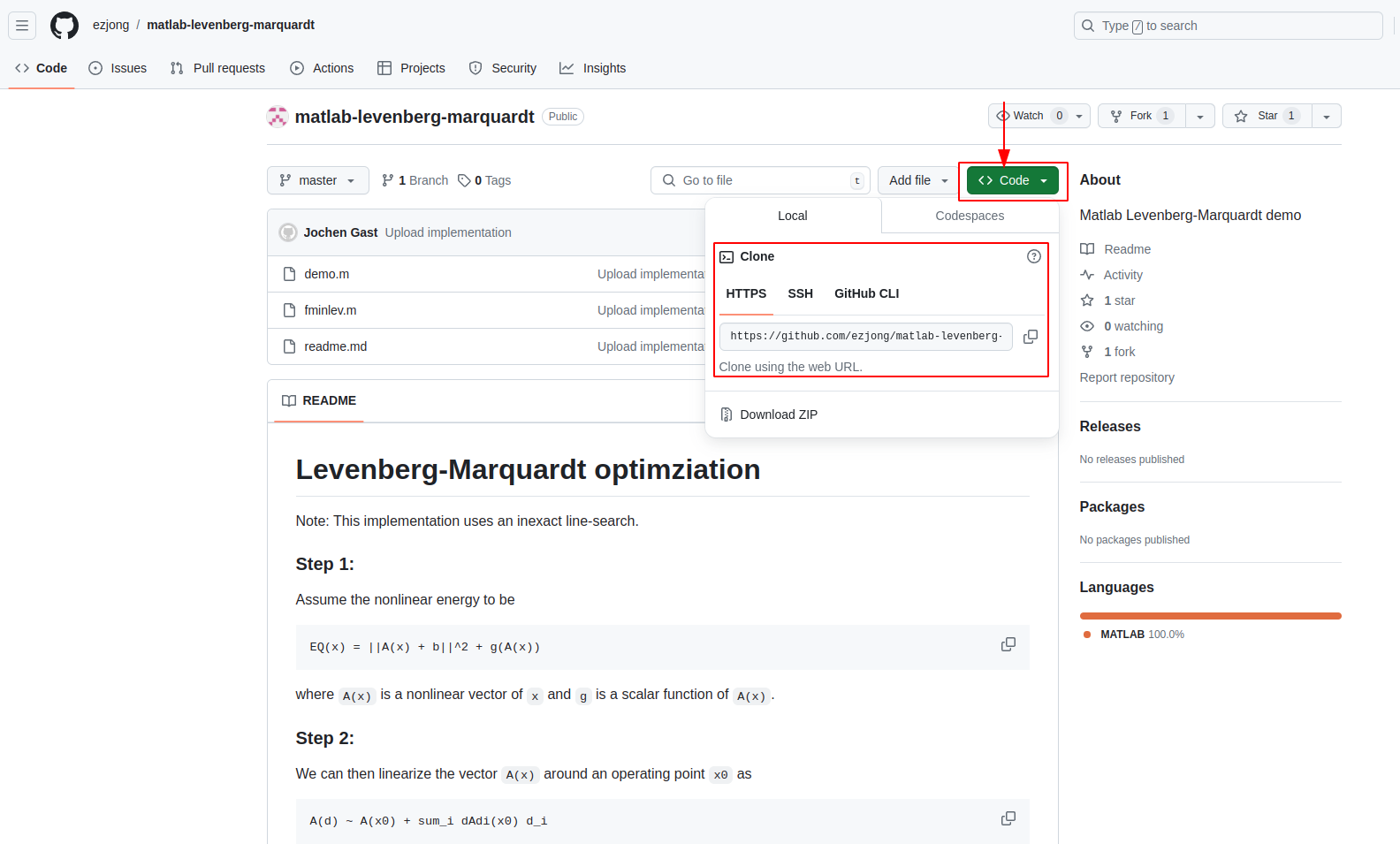


Git clone:

reference for more information: https://www.mathworks.com/help/simulink/ug/clone-git-repository.html

Let’s say I want to clone ezjong/matlab-levenberg-marquardt repository (chose arbitrarily someone’s remote repository for sake of tutorial).

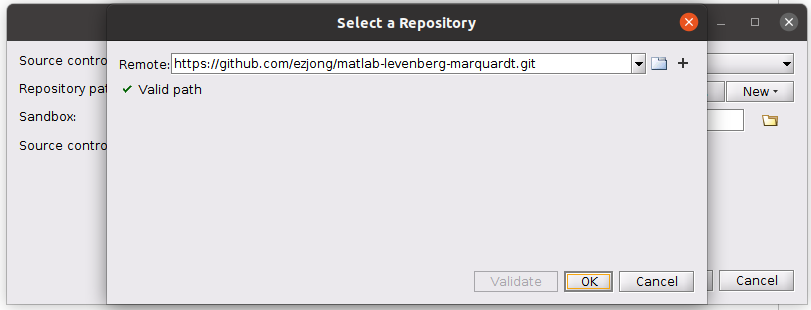
Go to his/her repository, click on “<> Code” and copy the HTTPS path.

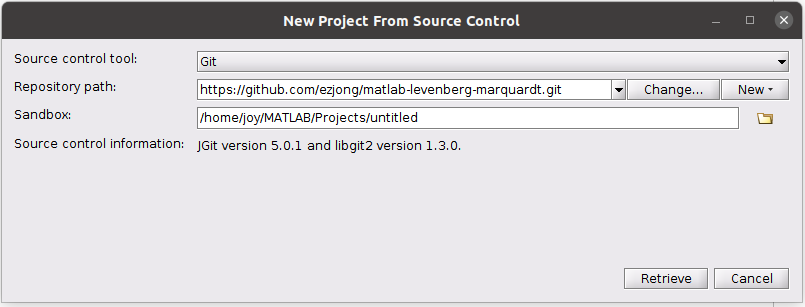


Go to your MATLAB and select from the home menu → “new project” → “from git”.

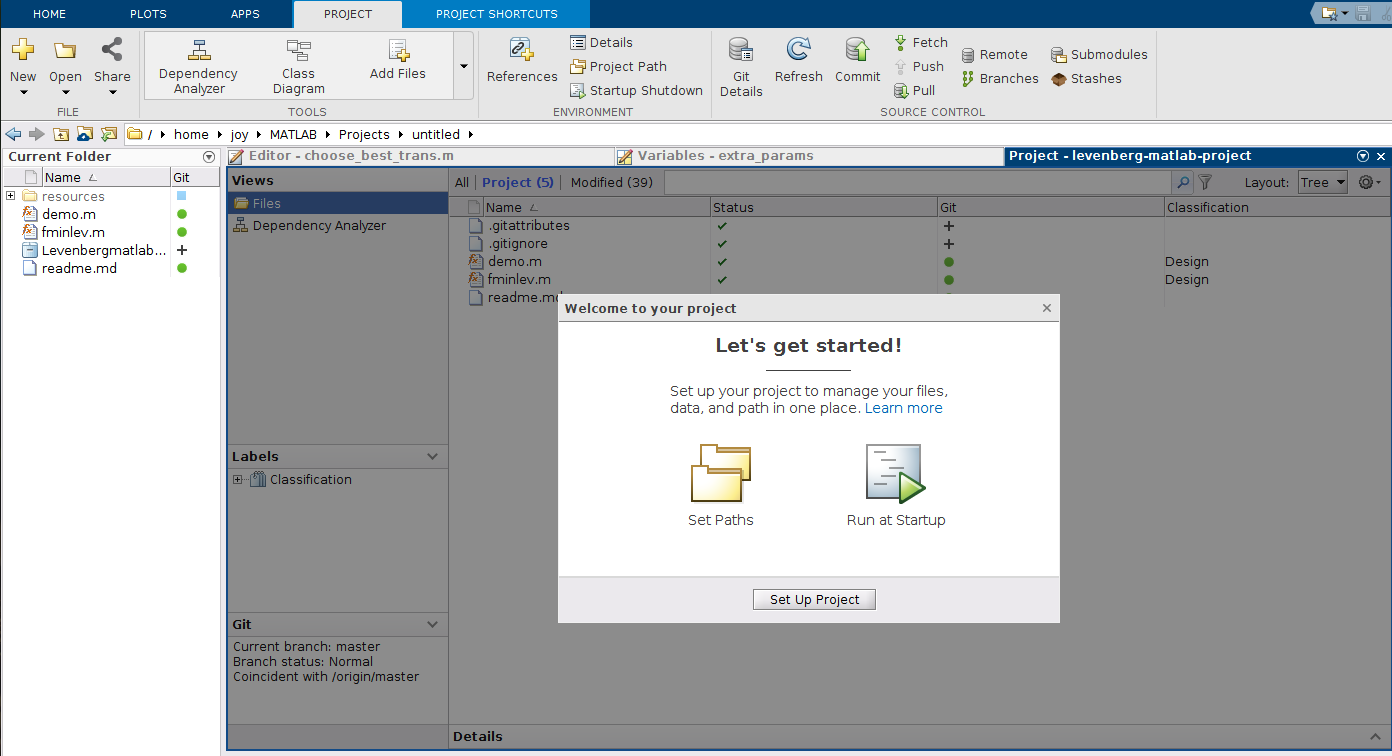
The “New Project From Source Control” window opens. We have already seen this procedure, which is similar to step # , where we linked our remote repository to our MATLAB local repository.

Here, you paste the HTTPS path you copied, validate it, retrieve it, give a name to the project if you do not have one already.





You will see the files of the remote repositoty in your local computer.



You can now add, delete and modify your project’s files.

\*\* Notice it is not possible to commit and push changes to someone’s else repository.

REFERENCES:

https://www.mathworks.com/help/matlab/matlab\_prog/use-git-in-matlab.html

<https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits>

https://docs.uabgrid.uab.edu/wiki/Git\_For\_MATLAB

<https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches>

https://www.mathworks.com/help/matlab/matlab\_prog/branch-and-merge-with-git.html

https://github.com/git-guides/git-pull

https://www.mathworks.com/help/simulink/ug/clone-git-repository.html