Week 3 - Git and GitHub

1 Introduction

As you learned in the lecture, git is a widely used software for distributed version control. Its main purpose is to maintain data integrity and to enable the collaboration of many contributors to the same project. It is defined as a ”distributed version control system”. It is a version control system (VCS) because it allows the developer to keep track of all the history which brought the project from its initial stage to the current one, allowing to have different versions of the same program in parallel, and to recover a past version at will. It is distributed, as its architecture prescribes that every user (the developer) has a copy of the project stored in their local computer. In order to work in teams on the same project and to share the source code through the internet, it is common to use an internet host for Git. One of the most popular options is GitHub. As we will see throughout this exercise session, GitHub offers more functionality than just hosting the project files and making them available to the developers.

Today’s exercise session has three main goals:

- complete the introduction exercises at https://learngitbranching.js.org to practice git virtually! We recommend that you complete at least:
  - the first 3 lessons of *Introduction Sequence* (Main tab): *Introduction to Git Commits, Branching in Git, Merging in Git*
  - the first 4 lessons of *Push & Pull – Git Remotes!* (Remote tab): *Clone Intro, Remote Branches, Git Fetchin’, Git Pullin’*

  Tip: if you make a mistake and you wish to reset the exercise, click on the ”?” at the bottom right.
- creating and editing repositories on GitHub (alone)
- collaborate with another BIO-210 student to edit their/your repository. The other student is referred to as your teammate, below!

2 Create a repository on GitHub

After practicing with https://learngitbranching.js.org, we will create our first Github repository. Go to the GitHub webpace and sign in. If you go on any page of Github, you will find in the upper-right corner a drop-down menu where you can select **New repository**.

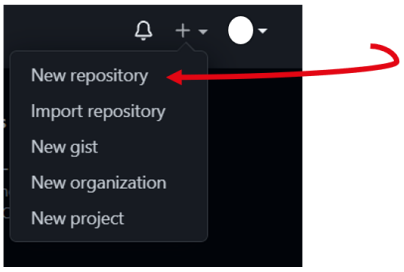

Figure 1: Create a new repository

A new page will open where you can select the characteristics of the repository. Let's give a short name to the generated repository. Pick your favorite name, or for consistency call it "bio210-[randomtag]". Optionally, you can always add a brief description describing the repository. For instance, we can add "Test repository to learn how to use Github - BIO 210"

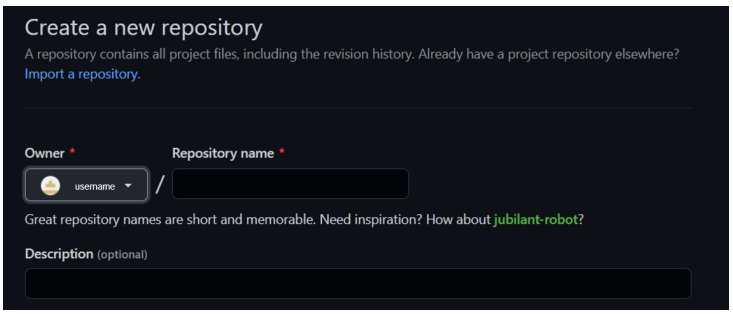

Figure 2: Provide a name and a description for your repository

Then, you can set the visibility of the repository. Depending on your project, you might want it to be *private* (invisible to others) such as for internal project development or you might want to share your repository with everyone and so to make it *public*. For our purposes, please make the repository **private** (this is just to use personal access tokens and access; but feel free to make it public and skip those steps).

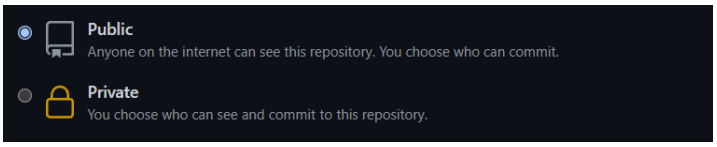

Figure 3: Select the visibility of the repository

To conclude, you can initialize the project with a *readme* file. The readme is displayed on the landing page and it's good practice to provide useful information about the project, its organization, dependencies (installation) and a manual to use it. For inspiration, check out the readme from a small project, such as https://github.com/DeepLabCut/Primer-MotionCapture and large one, such as https://github.com/numpy/numpy!

Another important file is the *.gitignore* file, Github provides different templates which you can select based on the IDE, editor or programming language you are using for developing your project. Lastly, it is important to license your repository in order to have an open-source project therefore allowing anyone to use, change and distribute the software (if you want that!). If you are curious about the difference between the licenses, have a look at this website `https://choosealicense.com/`!

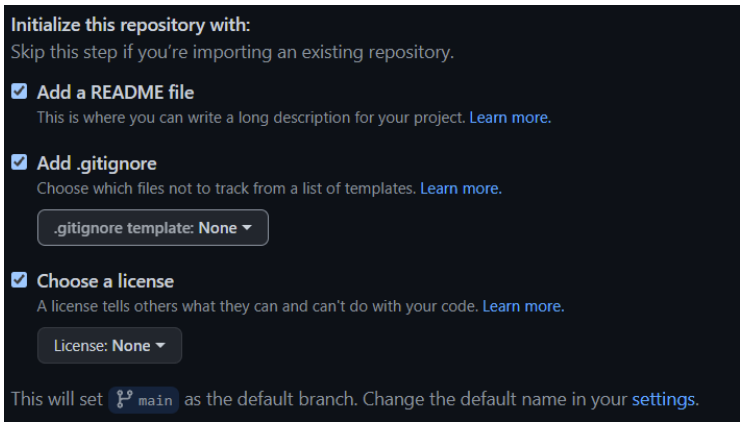

Figure 4: Select the readme, gitignore and license of your repository. If you want the licence and the gitignore files to be automatically generated, you have to select a template other than None.

Select **Create repository** and congratulation! (Perhaps) you have created your first repository.

3 Collaborate on a project

In this section you will work in teams of two. Please work with the colleague sitting next to you.

You are going to simulate a common situation in software development: two developers modify the same file and this generates a merge conflict. You will have to fix the conflicts and make sure that the code in the *main* branch is correct.

3.1 Generate a personal access token

First, we have to create a personal access token. In the upper-right corner of any page, click on your profile photo and go to ”Settings”. Now, on the left tab, go to ”Developer settings”

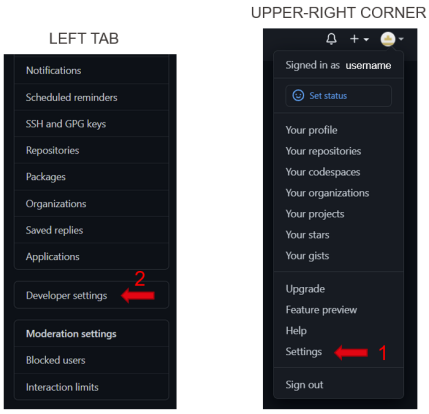

Figure 5: Go to developer settings

On the left tab, you can select personal access token and then select **generate new token** in the upper-right corner.

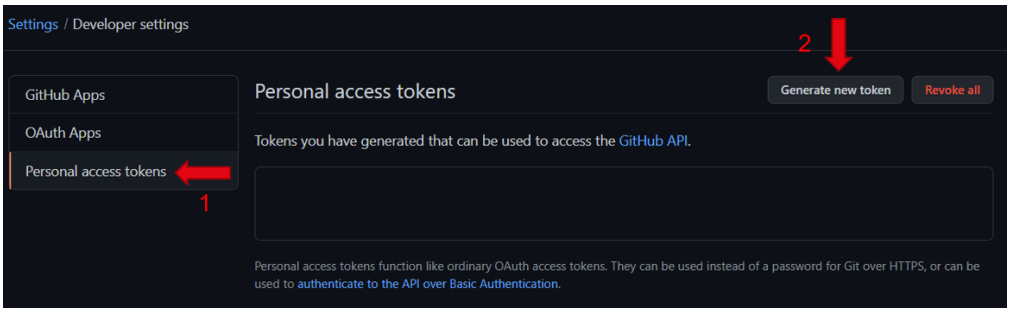

Figure 6: Generate a Personal access token

Insert now a note for the personal access token and the expiration date. In the ”select scope” section click:

- repo
- admin:repo_hook
- delete_repo

You can now generate the token. **BE CAREFUL**, make sure to copy the token and store it in a safe place. Treat your tokens like passwords and keep them secret (of course you can just reset it, when you lose it).

3.2 Clone a repository

In the previous section you created a repository on GitHub. The repository is the place where all the source code of your project is stored, together with other files, such as a *README.md*, a *.gitignore*, the documentation etc. The first thing to do is to clone the repository, which means to create a local copy of the project on your computer. To do so:

1. Go to the homepage of the repository ”bio210-[randomtag]” and copy the https address, either from the address bar of your browser or after clicking the green ”code” button
2. Open a terminal (on Windows: open Git Bash), move to the folder where you want to copy your project folder (e.g., *cd Documents*) and type

```
$ git clone <project_address>
```

3. Insert your username and insert your personal access token in the password.
4. Go to the root folder of the project:

```
$ cd <project_name>
```

3.3 Write code in a repository

If you cloned the repository from your project homepage, you are by default in the *main* branch. You can check it using the command *git branch*. The *main* branch is usually the one with the most recent working version of your code. It is a highly recommended practice to always have working code in the main branch. For experiments and to add new features ideally work on a separate branch. Give the branches meaningful names.

1. Create a new branch called *feature/fibonacci*
2. Create a new file called *main.py*, where you write a python program, which prints the first 10 fibonacci numbers
3. Create a commit in which you add the python script that you have just written
4. Move back to the *main* branch
5. Merge the branch *feature/fibonacci* into the main branch
6. Push the main branch to your GitHub repository, so that your local copy and the remote repository are synchronized.

```
$ git push origin main
```

Note: By default, this final command (*git push*) pushes the current local branch to a matching remote branch of the same name.

3.4 Contribute to someone else's project

At this point both you and your teammate have created your own local repositories with the code to print the first 10 fibonacci numbers. You will now both modify that file to experience a merge conflict.

To this end, it is important to learn how to give access to your repository to other people. To invite collaborators, go in the settings of your repository (this is not required for public repositories). Under the tab "Manage access", you can visualize who has access to your repository and add new people. You can select a specific person based on the username account on GitHub, the full name or the email. Now, you can add your teammate to your repository. Similarly, your teammate should add you to their repository.

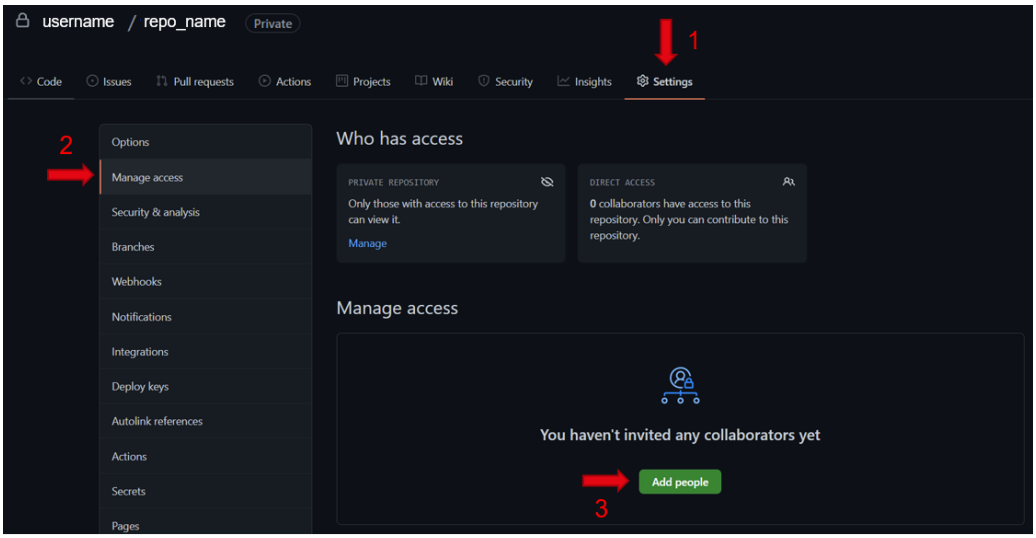

Figure 7: Invite people to collaborate on your repository

Start modifying your teammate's project:

1. Accept the invitation sent to you by email.

2. As you previously did with your own project, clone your teammate's project on your local computer. Make sure to navigate into the project root folder

```
$ cd <project_path>
```

3. Make sure you are in the *main* branch. Create a new branch called *feature/fibonacci_15*
4. Open the file *main.py*, where you should see the code to print the first 10 *fibonacci* numbers. Modify it so that it now prints the first 15.
5. Commit your changes and push them to a remote branch with the same name as your local branch (*feature/fibonacci_15*)

```
$ git push origin feature/fibonacci_15
```

Now move back to your own project. You will also modify the function fibonacci, as your teammate did, so that, at the moment of the merge, Git will not know which of the two changes to keep and which one to discard. This event is called a *conflict*. You will then solve it with the help of GitHub and successfully merge your branch into the *main*

1. Make sure you are in the main branch
2. Open the file *main.py*, where you should see the code to print the first 10 *fibonacci* numbers. Modify it so that it now prints the first 100.
3. Commit your changes and push them to the remote *main* branch.

```
$ git push origin main
```

Finally, once your teammate has also finished pushing his modifications to the remote *main* branch in his repository, move to the homepage of his project on GitHub. You will now create a **pull request** based on your edits in the branch *feature/fibonacci_15*,

1. Click on the tab "Pull requests" and click on "New pull request"
2. Select the branch *fibonacci_15* as the *compare* branch. You should see a disclaimer "Can't automatically merge", but you can ignore it for the moment.
3. Set your teammate as a reviewer and assign the pull request to yourself. Select the label "enhancement"
4. As you both modified the same file in the same lines, the pull request should create a merge conflict. GitHub allows you to solve them directly in the browser. At the bottom of the pull request page, click on "resolve conflicts"

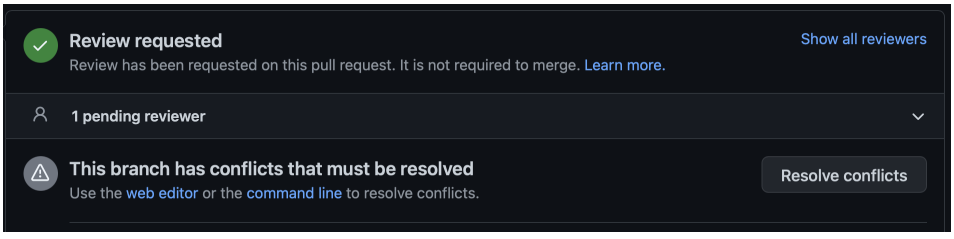

Figure 8: Example of how pull request looks like in the presence of conflicts

5. You should now see the two versions of the code, one in which the code prints the first 100 fibonacci numbers, and one in which it prints the first 15. You are confident that your version is the better one: remove the line with 100 and keep the one with 15. Remember to also remove the lines that git added as a delimiter for the two versions.
6. Once the code looks nice, click on ”Mark as resolved” and then ”Commit merge”. Leave the pull request for your teammate to review and approve it.

Now, you have to go back on your repository in the pull request tab. Here, you can review the pull request that your teammate has done and approve the change.

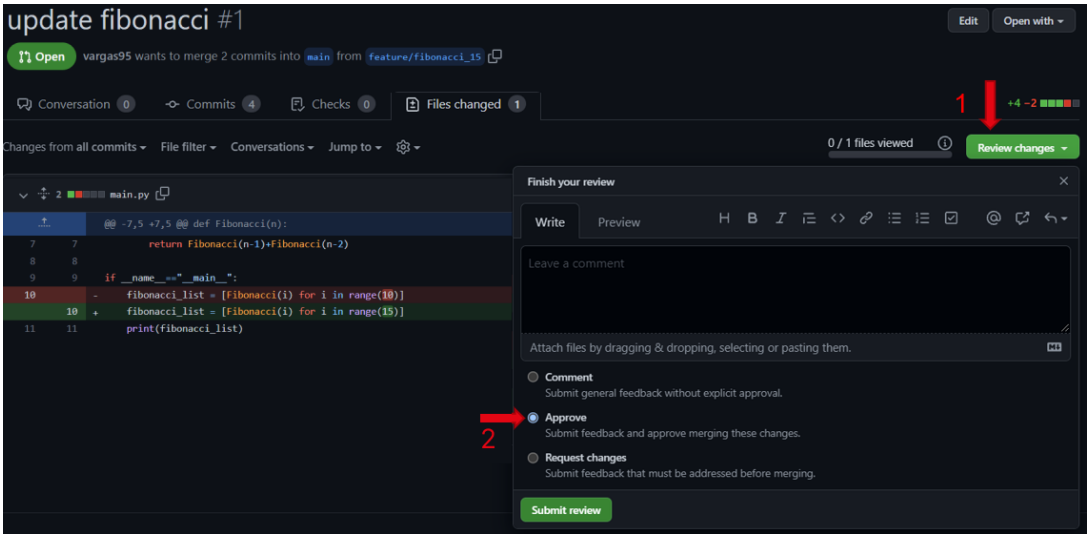

Figure 9: Review the modified files

Once you have reviewed the pull request, you will be redirected back to the ”conversation” tab. At the end of this page, you can finally merge the pull request.

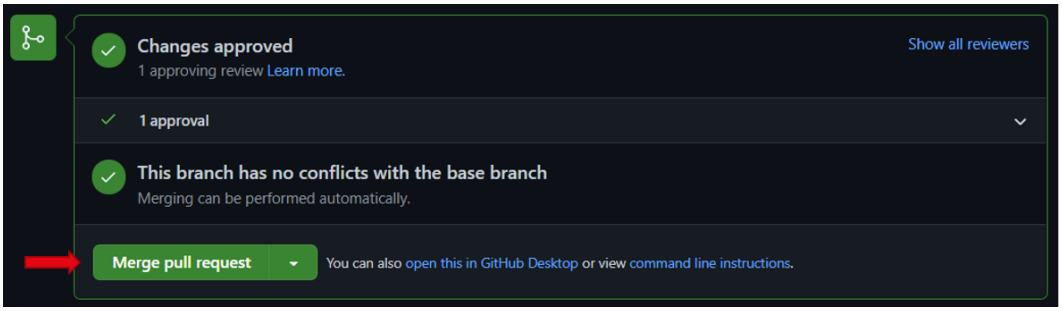

Figure 10: Merge the pull request

Today you learned a lot of new operations – rest assured that over the course of the semester, as you will use git and GitHub for your project, you will grow comfortable!

4 Common git issues and solutions

In this section, you will find some common git mistakes that you might meet during the project.

4.1 Permission Denied (publickey)

Situation: You are trying to push or clone a repository and Git returns a ”Permission Denied (publickey)” error message.

Error Message:

```
Permission denied (publickey).
fatal: Could not read from remote repository.
```

Solution 1: using Personal Access Token
You can follow the instruction in Section 3.1 to setup your personal access token.
Note: Remember to keep your personal access token secure and do not expose it in your scripts or repositories to maintain security.

Solution 2: using SSH keys
If you haven't configured SSH keys for GitHub, follow these steps to generate, add, and use an SSH key:

```
# Generate an SSH Key:
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
# Ensure ssh-agent is running
eval $(ssh-agent -s)
# Add your SSH key to the ssh-agent
ssh-add ~/.ssh/[your-ssh-key]
# Copy the SSH key to your clipboard and add it to your Git account
clip < ~/.ssh/id_rsa.pub
# If 'clip' does not work, you might use `pbcopy` (macOS) or
    manually open the file and copy the content.
pbcopy < ~/.ssh/[your-ssh-key].pub
```

Add the SSH Key to your GitHub Account:

- Go to GitHub, log in, and navigate to Settings.
- In the left sidebar, click on SSH and GPG keys.
- Click on New SSH key.
- Provide a descriptive title, paste your key into the ”Key” field and click on Add SSH key.

Now, you should be able to clone, push, and pull to/from repositories using SSH.

Note: Ensure to replace ‘your_email@example.com‘ with your actual email.

4.2 Resolve Merge Conflicts

Situation: When you attempt to merge two branches, Git may display an error indicating a merge conflict because the branches have competing changes.

Error Message:

```
Auto-merging [file_name]
CONFLICT (content): Merge conflict in [file_name]
Automatic merge failed; fix conflicts and then commit the result.
```

Solution: To resolve the merge conflict, follow these steps:

1. Open the file in a text editor.
2. Look for the conflict markers '<<<<<<<', '>>>>>>>', and '=======‘. The code between '<<<<<<< HEAD‘ and '=======‘ is your current branch's code, and the code between '=======‘ and '>>>>>>> [branch_name]‘ is the conflicting code from the branch you're trying to merge.
3. Manually edit the code to resolve the conflict, then remove the conflict markers.
4. Save and close the file.
5. Add the resolved file to staging using:

```
    git add [file_name]
```

6. Commit the resolved changes using:

```
    git commit
```

Note: Be careful while resolving merge conflicts and ensure that the resolved code maintains the desired functionality.

4.3 Failed to Push Some Refs

Situation: You attempt to push your commits to a remote repository and encounter an error because your local branch is behind the remote branch.

Error Message:

```
error: failed to push some refs to [repository_url]
hint: Updates were rejected because the remote contains work that
    you do
hint: not have locally. This is usually caused by another
    repository pushing
hint: to the same ref. You may want to first integrate the remote
    changes
hint: (e.g., git pull ...) before pushing again.
```

Solution: This issue can be resolved by following these steps:

1. Pull the changes from the remote repository and merge them with your local changes using:

```
    git pull origin [branch_name]
```

2. Resolve any merge conflicts if they occur and commit the changes.
3. Push your commits to the remote repository using:

```
    git push origin [branch_name]
```

Alternative Solution: If you want to overwrite the remote changes with your local changes (use cautiously as this will discard the remote changes):

```
git push -f origin [branch_name]
```

Note: Force pushing is a very dangerous operation which can cause remote commits to be deleted if your local repository is not up to date. Use it as a last resort! Always ensure to communicate with your team before forcing a push to prevent unintentional data loss.