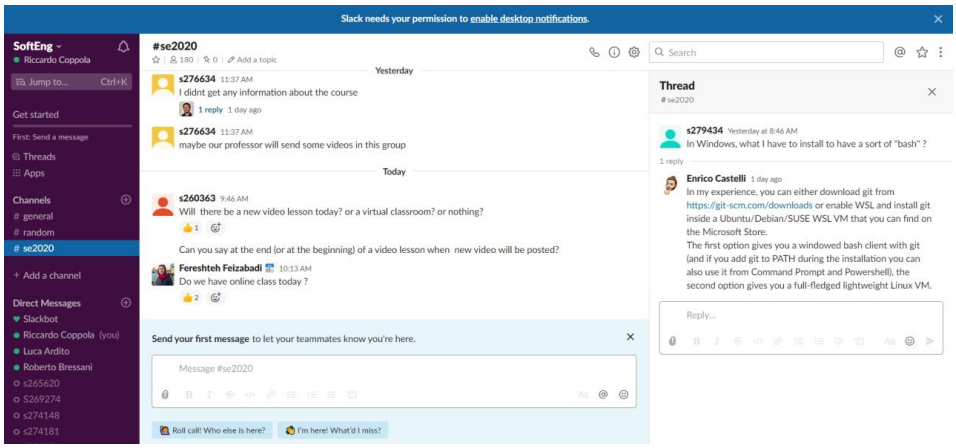
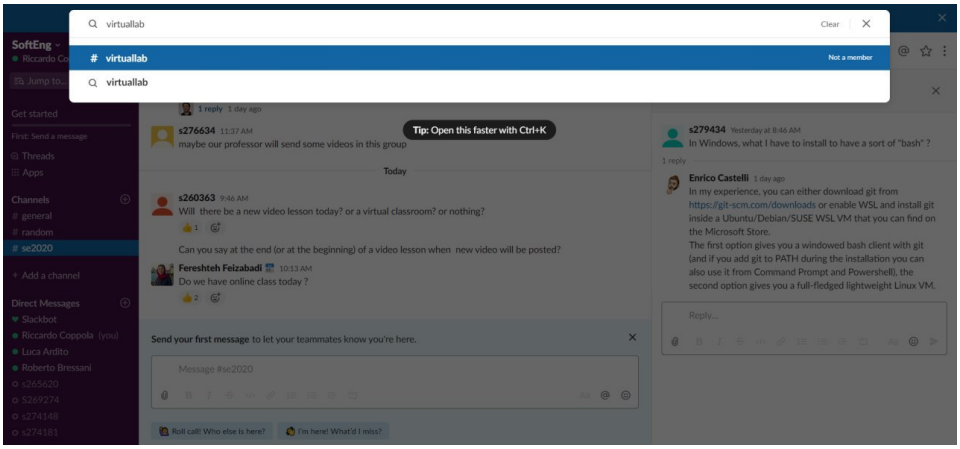
Lab 1: Git

How to join the #virtuallab channel for receiving assistance on Slack

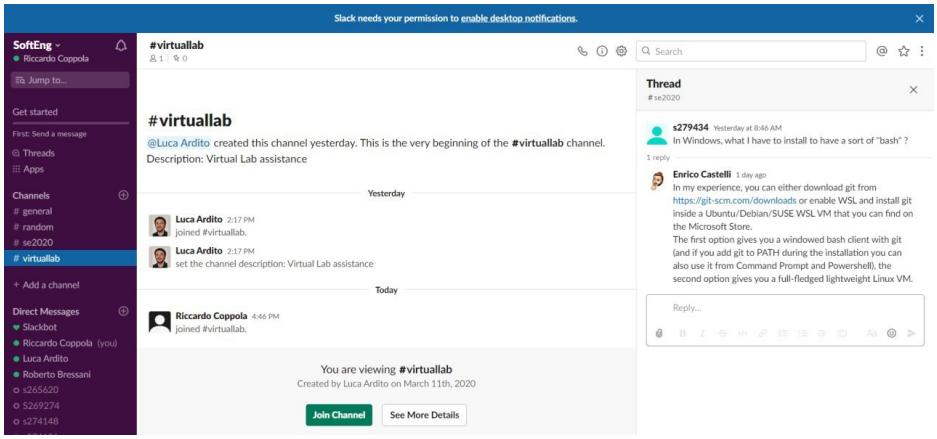
Select Jump to (top left of the screen) or press control + k



Input virtuallab and hit enter



Press the Join channel button



Exercise 1 – Stage and Commit
The exercise must be done using the terminal
Edit your git configuration:

```
$ git config --global user.name "NAME SURNAME sXXXXXX"

$ git config --global user.email sXXXXXX@studenti.polito.it
```

1. Create a folder called `lab1`.
2. `cd` into the `lab1` folder.
3. Create a file called `my_file.txt`.
4. Initialize an empty git repository.
5. Add `my_file.txt` to the staging area.
6. Commit with the message "adding my_file.txt".
7. Check out your commit with git log.
8. Create another file called `other_file.txt`.
9. Add `other_file.txt` to the staging area.
10. Commit with the message "adding other_file.txt"
11. Remove the `my_file.txt` file
12. Add this change to the staging area
13. Commit with the message "removing my_file.txt"
14. Check out your commits using git log
15. Make a change to `other_file.txt`. Use the `git diff` command to view the details of the change
16. Next, add the changed file, and notice how it moves to the staging area in the `git status` output. Also observe that the diff command you did before using add now gives no output. Why not? What do you have to do to see a `git diff` of the things in the staging area?
17. Now – without committing – make another change to the same file you changed in step 15. Look at the `git status` output, and the `git diff` output. Notice how you can have both staged and unstaged changes, even when you're talking about a single file. Observe the difference when you use the add command to stage the latest round of changes. Finally, commit them. You should now have started to get a feel for the staging area.
18. Create three files `one.txt, two.txt, three.txt, four.txt`
19. Add files `one.txt, two.txt, three.txt, four.txt` to the staging area
20. Commit with the message "adding files: one.txt two.txt three.txt and four.txt"
21. Use the `git rm` command to remove the file `one.txt`. Look at the status afterwards. Now commit the deletion.
22. Delete `two.txt`, but this time do not use Git to do it; e.g., if you are on Linux, just use the normal (non - Git) rm command; on Windows use del.
23. Look at the `git status`. Compare it to the status output you had after using the Git built-in rm command. Is anything different? After this, commit the deletion.
24. Use the `git mv` command to rename `three.txt` to `new_three.txt`
25. Look at the `git status`. Commit the change.

26. Now rename `four.txt` to `new_four.txt`, but this time using the operating system's command to do so. How does the status look? Will you get the right outcome if you were to commit at this point? Work out how to get the status to show that it will not lose the file, and then commit. Did Git at any point work out that you had done a rename?

Exercise 2 – Working with remote repository

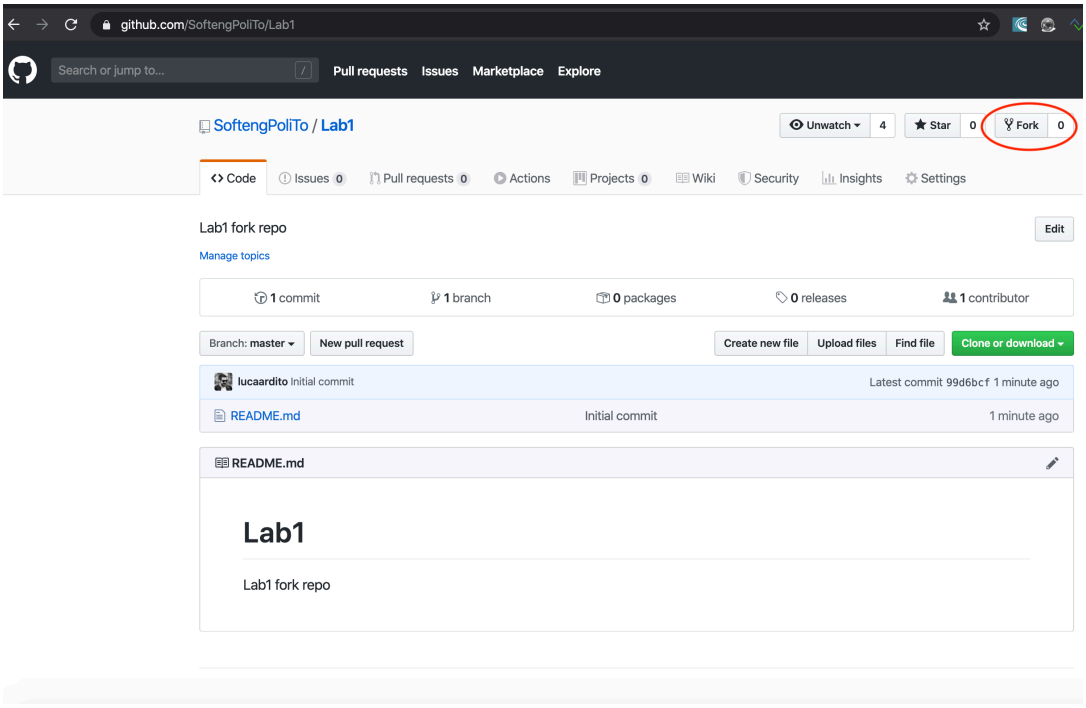
- Create a GitHub account: https://github.com/join
- Setup GitHub with a SSH key: https://help.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh
- Create a new repository: https://github.com/new
- Copy git@github.com:**USERNAME/project**.git
  WHERE: USERNAME is **your** GitHub username and project is **your** project name. change these two values accordingly.

```
mkdir exercise2
echo "exercise 2" >> test.txt
git init
git add test.txt
git commit –m "first commit"
git remote add origin git@github.com:USERNAME/project.git
git remote –v
git push –u origin master
```

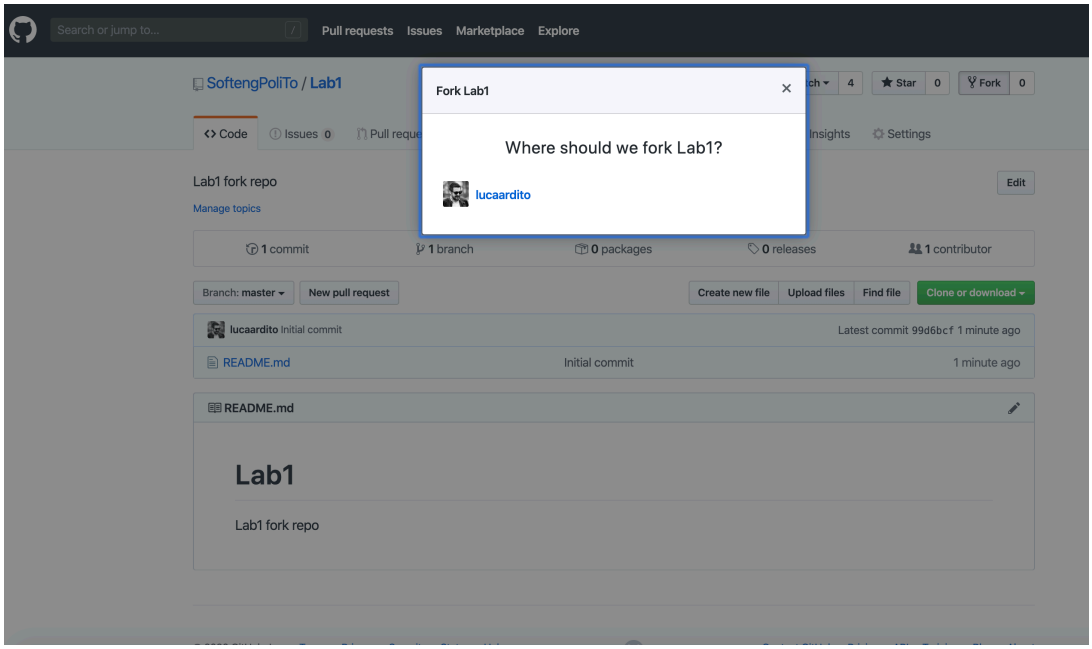
- `mkdir exercise 2` - creates a new folder
- `echo “exercise 2” >> test.txt` - creates a new file called test.txt containing the text exercise 2
- `git init` - make sure we initialize the repository
- `git add test.txt` - let's add the test.txt file (we can also do git add . or git add -A here)
- `git commit –m "first commit"` - add a commit with the message "first commit"
- `git remote add origin git@github.com:USERNAME/project.git` - This command tells our local repository about a remote repository located somewhere. The location of our remote repository is the project called **project** owned by the user **USERNAME** on Github (git@github.com:USERNAME/project.git) .
- `git remote –v` - To see your remotes locally you can type git remote -v. If you need to remove a remote you can use git remote rm NAME_OF_REMOTE
- `git push –u origin master` - We can send our code from a local repository to our remote repository (which we aliased to origin in the previous command). The -u flag allows us in the future to only have to type git push instead of git push origin master.

Exercise 3 – Fork, Clone and Pull Request

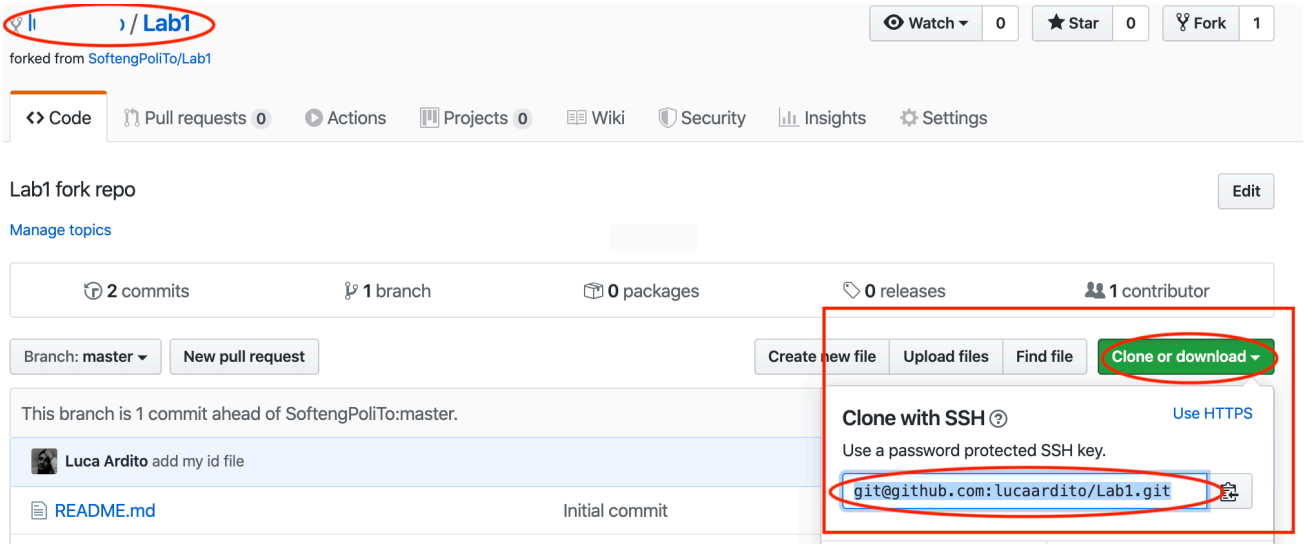
1. Open https://github.com/softengPoliTo/Lab1
2. Click Fork



3. Fork the Lab1 repository in your workspace



4. Copy the ssh link of your project (the one in your account **not the one in SoftengPolito account**)



5. Clone it in your pc with this command

   `git clone git@github.com:`**`USERNAME`**`/Lab1.git`

   where USERNAME is your username

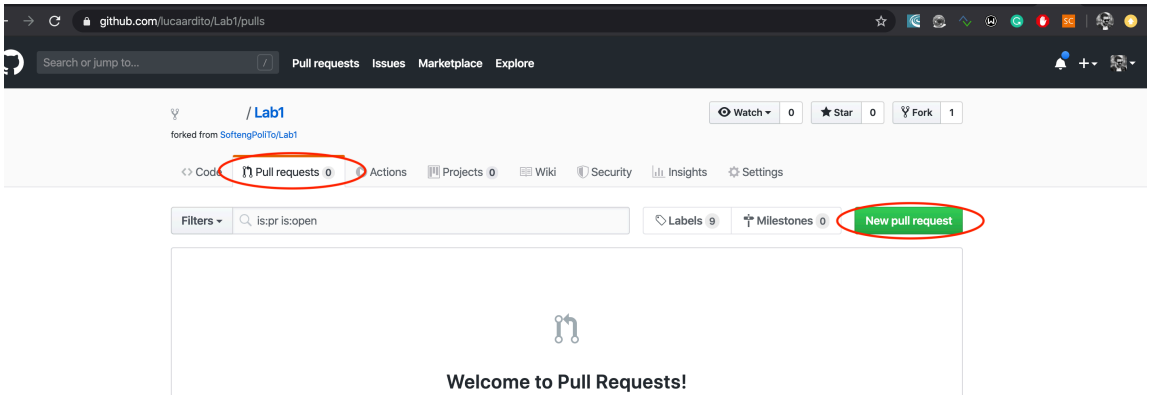
6. Create a new file called sXXXXXX.txt (where XXXXXX is your polito id)

7. Add it to the staging area

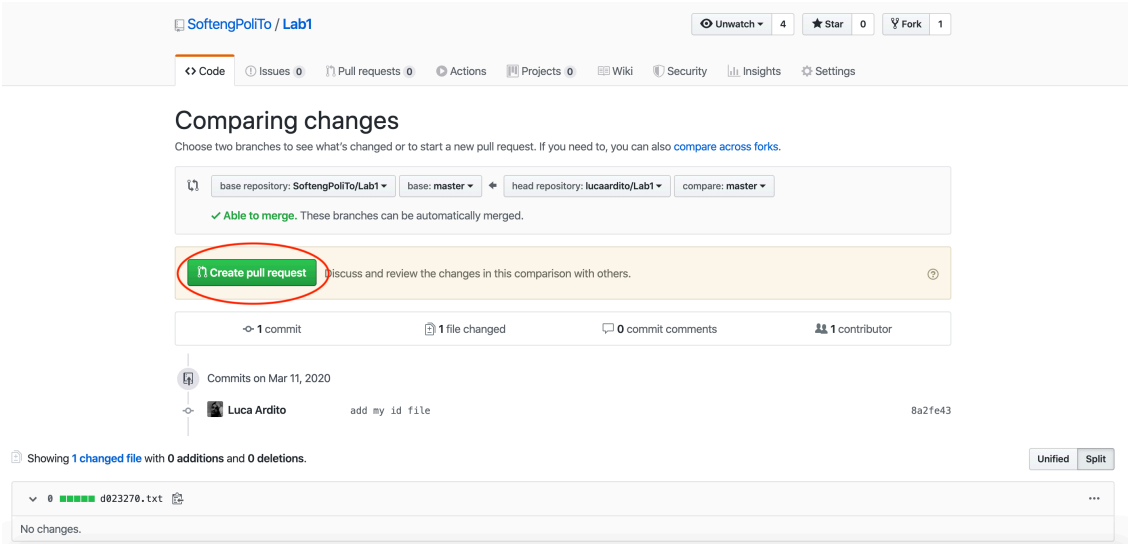
8. Commit with message “adding sXXXXXX.txt file”

9. Push it in origin master

10. Ask for a pull request



11. Create the new pull request



12. Write a comment