

# Using Git and Github

## Contents

I-	Introduction .....	2
II-	Terminology and principles .....	2
III-	Getting started .....	2
IV-	Branching and Merging .....	5
V-	Using Github.....	7
VI-	Merge Conflicts .....	9
VII-	Pull Requests .....	10
VIII-	Further resources .....	12

## I- Introduction

Git is a version control system used by programmers to track the changes they've done to their work over time and retrieve past versions if necessary. Git discards the need for saving different folders with different versions of a project and allows simple navigation of previous versions.

Git is a software, installed locally on your computer on which it stores the different versions, or "snapshots". It is thus not capable of sharing code online, or at least not alone. Only when Git is paired with online code-sharing platforms like Github can it help share code between members of a team.

Other version control systems exist, but Git is the most popular because it is fast, open-source and supports very useful features like branching and merging, which we will talk about later.

## II- Terminology and principles

In git terminology, a "repository" refers to a folder where you store files, other folders and their old versions. A repository can be local or remote (see github repos described in part V). A "commit" or snapshot refers to a specific version of your project, as saved in git's history. It is like a picture of your work as it was in a certain moment in time.

You can see git's history as a tree of different snapshots connected to each other with parent-child relationships. Understanding how different versions of your project are connected in a tree can seriously impact your understanding of the commands you use.

You can explore that facet of git on <https://learngitbranching.js.org> after completing this tutorial, especially on the first 3 lessons of *Introduction Sequence*. However, be careful because these lessons simplify some commands and do not constitute a good guide for using the terminal.

## III- Getting started

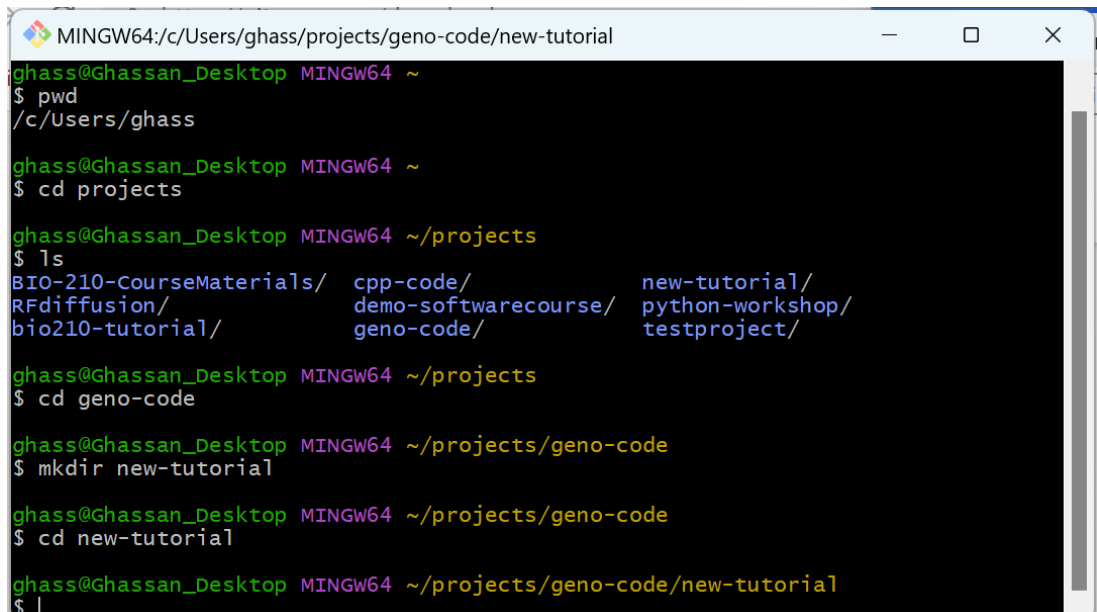
You can download git on this site: <https://git-scm.com/downloads>

When dialog boxes open asking you about preferences for the download, you can keep all the settings as they are. One dialog box to look for asks you what text editor to use, notably when resolving merge conflicts. The default one is Vim, which is arguably the worst, most uselessly complicated text editor I have come across. I would recommend changing it to VS Code.

To use the commands associated with git, you can use your newly installed Git Bash Terminal (on Windows) or download a graphical user interface like Github Desktop. This tutorial will mostly use the terminal.

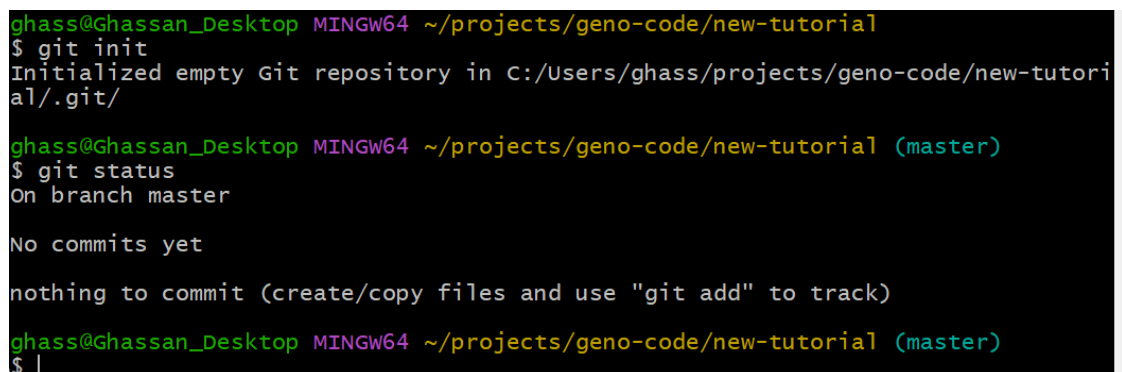
a) We will start by initializing a git repository, open the terminal and place yourself where you want to initialize a git repository. These commands help you navigate your files:

- pwd: print working directory: returns the directory in which you are now. By directory, we mean a certain location (folder) on your computer
- ls: lists what is present in the current directory: files and folders
- cd [path name]: lets you navigate to a certain location
- mkdir [name]: create a new folder

A screenshot of a terminal window titled 'MINGW64: c:/Users/ghass/projects/geno-code/new-tutorial'. The terminal shows the following commands and output:

```
ghass@Ghassan_Desktop MINGW64 ~  
$ pwd  
/c/Users/ghass  
  
ghass@Ghassan_Desktop MINGW64 ~  
$ cd projects  
  
ghass@Ghassan_Desktop MINGW64 ~/projects  
$ ls  
BIO-210-CourseMaterials/  cpp-code/  new-tutorial/  
RFdiffusion/             demo-softwarecourse/  python-workshop/  
bio210-tutorial/         geno-code/  testproject/  
  
ghass@Ghassan_Desktop MINGW64 ~/projects  
$ cd geno-code  
  
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code  
$ mkdir new-tutorial  
  
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code  
$ cd new-tutorial  
  
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial  
$
```

b) We can initialize a git repository in our newly created folder, aka tell git that we would want to track the changes inside our new-tutorial folder. The command “git status” allows you to ask git what is happening in the repository right now: changes made but untracked, files ready to commit, etc.

A screenshot of a terminal window showing the following commands and output:

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial  
$ git init  
Initialized empty Git repository in C:/Users/ghass/projects/geno-code/new-tutorial/.git/  
  
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)  
$ git status  
On branch master  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)  
  
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)  
$
```

c) We can now create our first file, you can do that manually, or use the “echo” command as below, “git status” now tells us that we made some changes but they have not been added to git’s history, which means this version will be lost if we make changes to the file!

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ echo "print('GenoRobotics is cool')" > new-file.py

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    new-file.py

nothing added to commit but untracked files present (use "git add" to track)
```

d) Let us now add this new file to our git history. The following commands will create a “snapshot” of our project with this version. We will thus be able to go back to this version if necessary:

- `git add [file-name/s]`: this is the first step of submitting our changes to git, this adds our files to the “staging area”, an in-between step that allows you to choose which changes you want to add to git. Use “`git add -A`” to add all the changes
- `git commit -m “[your message]”`: this creates the snapshot, choose a message that is descriptive of the changes you make, as it will appear in your version history.
- `Git log`: Now that it is all done, git log allows us to visualize all the versions we have until now.

**Tip:** keep your commits small and write good messages, you’ll be thankful later!

**Tip:** `git log --all --graph --decorate` lets you see the history as a tree, particularly useful to track branches. To exit git log press “q”

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git add new-file.py
warning: in the working copy of 'new-file.py', LF will be replaced by CRLF the n
ext time Git touches it

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git commit -m "new printing file"
[master (root-commit) 8d621f8] new printing file
1 file changed, 1 insertion(+)
create mode 100644 new-file.py

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git log
commit 8d621f86ea04f047ad68f871e2f7cb2ea5a47bce (HEAD -> master)
Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
Date: Sun Oct 22 20:03:24 2023 +0200

    new printing file
```

### Digression – git ignore:

You might decide to have some files or folders in your repository that you do not want to track with git. These are typically files to do with running or building the code or test files you change frequently.

To do that you can create a .gitignore file in which you write the name of all the files or folders to keep untracked. To learn more about how to write git ignore files go to <https://git-scm.com/docs/gitignore#:~:text=The%20purpose%20of%20gitignore%20files,being%20reintroduced%20in%20later%20commits.>

## IV- Branching and Merging

You might have noticed the little “(master)” next to our command line since we initialized our repository. It refers to the branch we are currently on!

In git, branches can be used to work on different changes to your project simultaneously, you can try things out and do different commits on a branch while still being able to easily go back to the main branch (master). It is good practice to always have functioning code on the main branch and to create new branches for each feature we want to work on.

When creating a new branch, it will have all of the code that is already on the master branch, from which you can start changing.

a) Let us now create a new branch:

- git branch [name]: creates a new branch
- git checkout [name]: makes you switch to an existing branch
- git checkout -b [name]: shortcut to create a new branch and place yourself in it.
- Git branch: lists all the branches available

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git branch
* master

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git checkout -b feature1
Switched to a new branch 'feature1'

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (feature1)
$ git branch
* feature1
  master
```

**Note:** The little asterix in git branch tells you where you are placed now

**Tip:** You can even initialize a branch with the code of an old commit using `git checkout -b [name_of_new_branch] [id_of_commit]` where the id is the long number you get in your git log history

- b) Now try creating a new file in your branch, adding and committing it, then doing the same in your master (or another branch). Use `git log --all --graph --decorate` to visualize how our history has diverged into two paths. You should get something like this:

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git log --all --graph --decorate
* commit b123c87afa91ac81f7684889933159f19d856dbf (HEAD -> master)
  Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
  Date: Sun Oct 22 20:40:46 2023 +0200

    random add

* commit eb6d79ce3c49f6ecee066394d120d47930b913cc (feature1)
  Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
  Date: Sun Oct 22 20:39:05 2023 +0200

    new text file

* commit 248d97d68bead241aa05ce3de445da962735dedb
  Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
  Date: Sun Oct 22 20:18:26 2023 +0200

    new main

* commit 8d621f86ea04f047ad68f871e2f7cb2ea5a47bce
  Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
  Date: Sun Oct 22 20:03:24 2023 +0200

    new printing file

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
```

You have now finished working on your feature and you want to integrate it in the master branch, it is now part of your project. To do that, you can merge your branch with the master. This will create a new commit with both branches as parents.

- c) Checkout to the master branch and use the command “`git merge`”. You will be asked to write a message describing the merge. You can write it on Vim if you are sadistic and type `:wq` (write and quit) to finish or on VS Code like sane people.

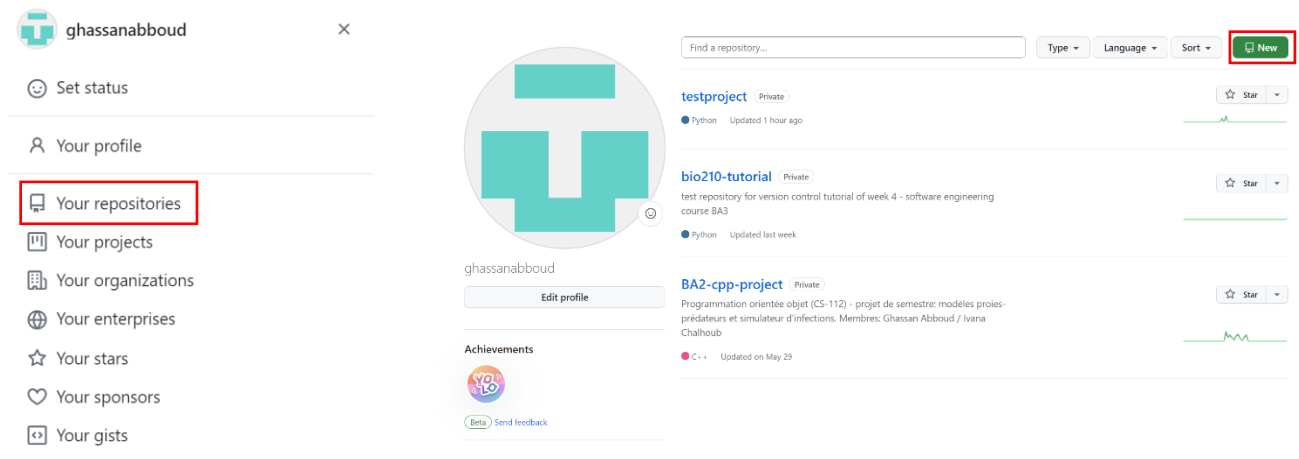
```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git merge feature1
Merge made by the 'ort' strategy.
 new_text.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 new_text.txt

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git log --all --graph --decorate
* commit c4f0662c92dc27c7e700d8b1617f9c9bc6604306 (HEAD -> master)
  \ Merge: b123c87 eb6d79c
  | Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
  | Date: Sun Oct 22 20:45:50 2023 +0200
  |
  | Merge branch 'feature1'
  |
  | * commit eb6d79ce3c49f6ecee066394d120d47930b913cc (feature1)
  | | Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
  | | Date: Sun Oct 22 20:39:05 2023 +0200
  | |
  | | new text file
  | |
  | * commit b123c87afa91ac81f7684889933159f19d856dbf
  | | Author: Ghassan Abboud <ghassan.abboud@epfl.ch>
  | | Date: Sun Oct 22 20:40:46 2023 +0200
  | |
  | | random add
  | |
```

## V- Using Github

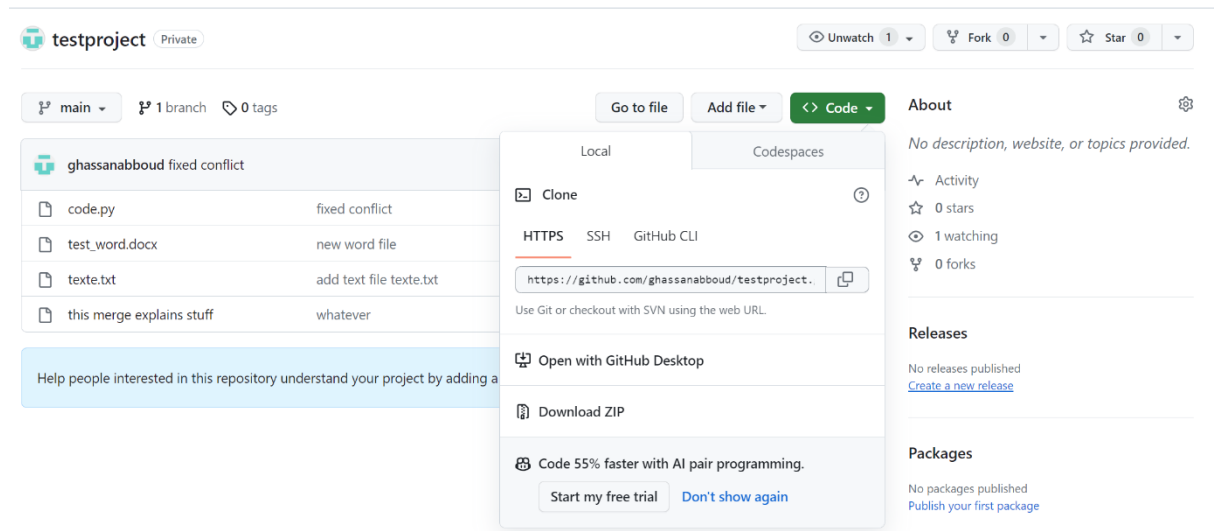
All the work we have done so far has been **local**, we learnt to save snapshots of our work, to improve our workflow with branches and merge them back in our project. Now to the next step, collaborating with other programmers on github. We will see how to join github repositories and contribute with our code. Set up an account on <https://github.com/>

For the following part you will need to create a github repository. You can do so on the website by clicking on your icon in the top right -> your repositories -> new .



When you create a new repo, you can add:

- A description
  - Choose whether it is public (everyone on github can see) or private (only you and the people you invite)
  - a README file: a file with a special format (markdown) to describe your project and how to use it
  - .gitignore file: you can pick templates for specific languages like Python. One might add that when collaborating on projects, it is useful to include your .gitignore file inside the .gitignore. That way, you do not have to push to your teammates updates to your .gitignore that are irrelevant to them.
- a) When working on a project, you will often be asked to join an existing github repo, to do so we use the command `git clone [github_link]`. To get the link, go to the site of the project and click on the copy icon. Git clone creates a link between your newly created local repo and the one online. Use `cd` to navigate to that repo.



```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial (master)
$ git clone https://github.com/ghassanabboud/testproject.git
Cloning into 'testproject'...
remote: Enumerating objects: 37, done.
remote: Counting objects: 100% (37/37), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 37 (delta 4), reused 33 (delta 3), pack-reused 0
Receiving objects: 100% (37/37), 4.22 KiB | 864.00 KiB/s, done.
Resolving deltas: 100% (4/4), done.
```

- b) Create a new file in your local repository, add it and commit it. Running git status now not only tells you the state of your local changes, but also how they compare with what's published online.

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial/testproject (main)
$ git add -A
warning: in the working copy of 'newtextfile.txt', LF will be replaced by CRLF the next time Git touches it

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial/testproject (main)
$ git commit -m "new text again"
[main 7cd42cb] new text again
1 file changed, 1 insertion(+)
create mode 100644 newtextfile.txt

ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial/testproject (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
```



- c) Indeed, we have just added one commit **locally**, but our teammates cannot see these changes. In order to publish them online, we use “git push” or “git push origin [branch\_name]”.

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial/testproject (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 281 bytes | 281.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ghassanabboud/testproject.git
d178381..7cd42cb main -> main
```

- d) Similarly, when our teammates have done changes and pushed them, we want to get their versions locally, hence we use git pull. Try doing so with another teammate and see how git pull allows you to get your teammate’s work.
- e) On Github, branches work similarly to git. In the context of Genorobotics (and in general), it is good practice to create a new branch for every new feature/change you work on. Finally, you can open a pull request (see later) to integrate your changes into main. While working on a branch, if you ever need code completed on other unmerged branches, you can specify where to pull from with “git pull origin [branch\_name]”

**Note:** Sometimes using “git status” will tell you you are up to date with the main online branch even though changes have been done. It is because git has not checked whether changes were done since the last time you connected to the online repo. To fix this run “git fetch” to get up to date information on the online repo. I recommend using git fetch every time you start working on the project

## VI- Merge Conflicts

Imagine the following scenario, you are collaborating with a friend on the same branch for a new feature. you have done some changes to the code locally, and forget to push your work. You come back the same evening to push your work and get the following message:

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial/testproject (main)
$ git push
To https://github.com/ghassanabboud/testproject.git
! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/ghassanabboud/testproject.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

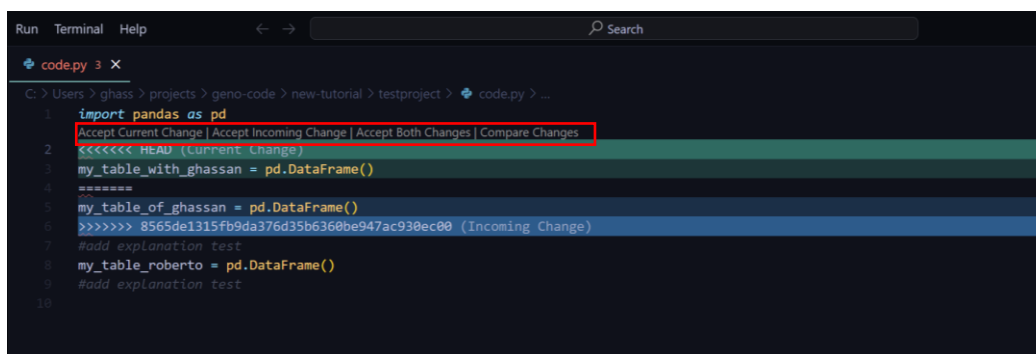
You then ask your friend and learn that they did some work while you were away. You need to pull their changes and integrate them before you can push yours. If you had worked on

different files or different lines of the same file, there would be no issue. Git is smart, it can add your friend's changes into your version with git pull and you can then publish yours.

What happens if you two had changed the same lines of code? What should git do? Overwrite your changes or lose those of your friend? We have just created a merge conflict. Pulling your friend's changes creates the following message:

```
ghass@Ghassan_Desktop MINGW64 ~/projects/geno-code/new-tutorial/testproject (main)
$ git pull
Auto-merging code.py
CONFLICT (content): Merge conflict in code.py
Automatic merge failed; fix conflicts and then commit the result.
```

The easiest way to fix this is to open the file creating a conflict in VS Code where you can clearly see the different versions and pick what to do with the changes (keep current, accept incoming, etc.). When the conflict is resolved, you can push your changes.



Create a remote repository with a teammate and simulate a merge conflict by editing the same file. Try to resolve the conflict with VS Code.

## VII- Pull Requests

You have finished working on your feature and now want to integrate it into the main branch. If you try to push to the main branch, you notice that it is **protected**, which means some restrictions have been imposed on pushing to the branch to prevent it from becoming a hot mess.

In this case, you must open a pull request to be validated by other teammates in order to get the changes to the main branch. *Even when the main branch is not protected*, pull requests (PR) are the preferred way to integrate work to the main branch as they allow you to discuss the changes with your teammates, open issues, etc.

- a) Using everything you have learnt so far, create a new github repository online (with a README), clone it then create a new branch "feature" to work on. Commit some changes and push them to the "feature" branch. When you create a new branch and push changes to it online for the first time, github does not know where to post them online, since your new **local** branch does not have a **remote** counterpart. This results in the fatal error you see below. It is easily resolved by copy-pasting their proposed command, it created a new remote branch with the same name as your local one and pushed changes there.

```

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics
$ git clone https://github.com/ghassanabboud/finaltest.git
Cloning into 'finaltest'...
warning: You appear to have cloned an empty repository.

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics
$ cd finaltest

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics/finaltest (main)
$ git checkout -b "feature"
Switched to a new branch 'feature'

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics/finaltest (feature)
$ echo "print('almost done')" > code.py

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics/finaltest (feature)
$ git add -A
warning: in the working copy of 'code.py', LF will be replaced by CRLF the next
time Git touches it

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics/finaltest (feature)
$ git commit -m "new code"
[feature (root-commit) 0ab7a98] new code
1 file changed, 1 insertion(+)
create mode 100644 code.py

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics/finaltest (feature)
$ git push
fatal: The current branch feature has no upstream branch.
To push the current branch and set the remote as upstream, use

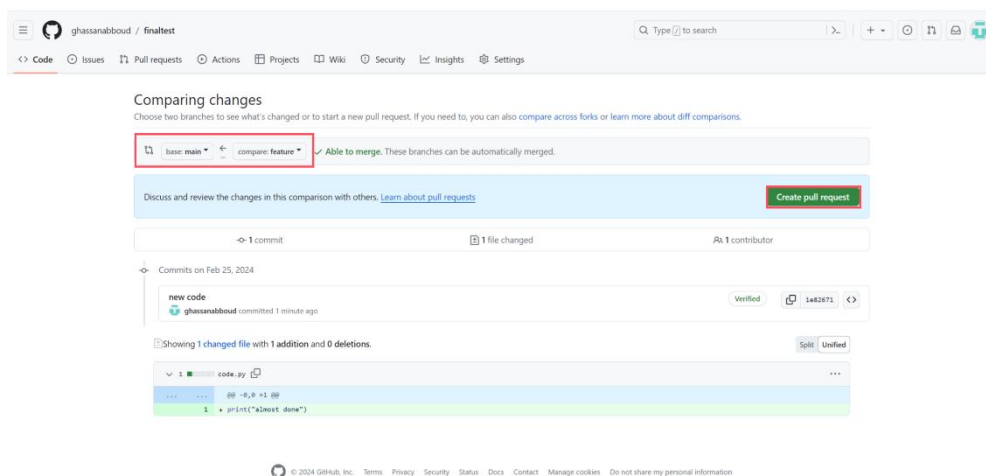
    git push --set-upstream origin feature

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

ghass@Ghassan_Desktop MINGW64 ~/projects/genorobotics/finaltest (feature)
$ git push --set-upstream origin feature
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 232 bytes | 232.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ghassanabboud/finaltest.git
 * [new branch]      feature -> feature
branch 'feature' set up to track 'origin/feature'.

```

- b) Now let's create a pull request to merge the changes of our new "feature" branch into the main, this time using a pull request. Go to the online repository on the github website, then "pull request" > "new pull request". You should then choose which branch is getting merged into which and finally create the pull request. You can add a description to the pull request with the changes you are merging



- c) Once created, you can request feedback from other teammates, discuss the request and check whether it can be merged without merge conflicts. Finally, "merge pull request" finishes the job, the changes are now on main.

## VIII- Further resources

- More about Git's data model, snapshots and how they relate to each other, from Anish Athalye's excellent lecture on git at MIT: <https://missing.csail.mit.edu/2020/version-control/>
- The comprehensive git book: <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
- Git commands cheat sheet: <https://education.github.com/git-cheat-sheet-education.pdf>