Programming Course

Lecture 4: Version Control

Diego de Sousa Rodrigues
diego.desousarodrigues@sciencespo.fr

Sciences Po Paris

Disclaimer

- Most of the material is drawn from the excellent course prepared by software carpentry
- In particular, most exercises are drawn from it (If you really want to learn something, don't look up the answers)
- Other source of inspiration is the very complete QuantEcon website

What and why ?

Version Control, Git & Github

What and why ?

Version Control, Git & Github

- A **version control** system is a tool that keeps track of these changes for us, effectively creating different versions of our files.

What and why ?

Version Control, Git & Github

- A **version control** system is a tool that keeps track of these changes for us, effectively creating different versions of our files.

$\Rightarrow$ Imagine you drafted an excellent paragraph for a paper you are writing, but later ruin it. How would you retrieve the excellent version of your conclusion ? **Is it even possible ?**

What and why ?

Version Control, Git & Github

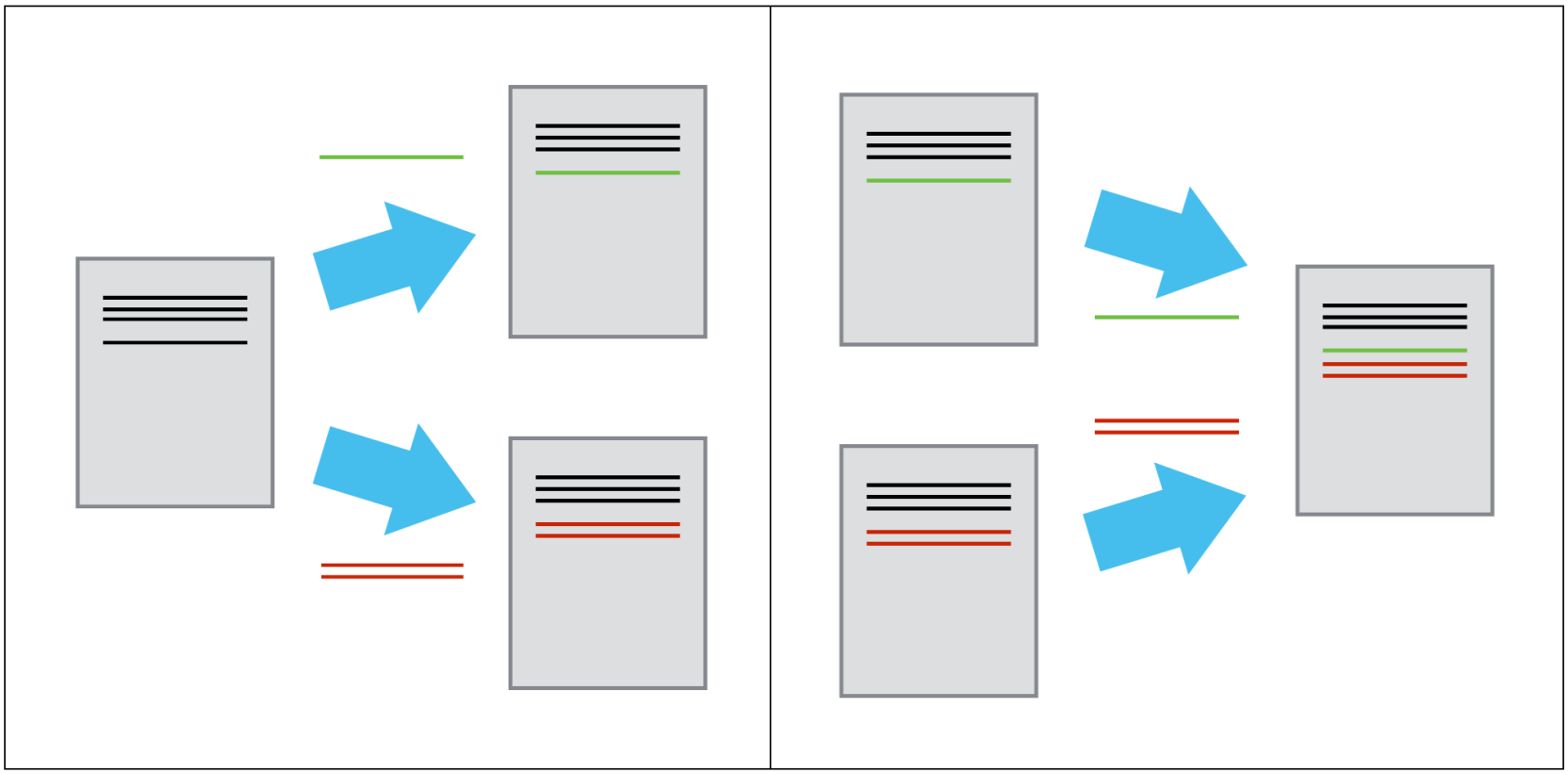
- A **version control** system is a tool that keeps track of these changes for us, effectively creating different versions of our files.

⇒ Imagine you drafted an excellent paragraph for a paper you are writing, but later ruin it. How would you retrieve the excellent version of your conclusion ? **Is it even possible ?**

⇒ Imagine you have 5 co-authors. How would you manage the changes and comments they make to your paper ? If you use LibreOffice Writer or Microsoft Word, what happens if you accept changes made using the Track Changes option ? **Do you have a history of those changes ?**

What and why ?

Version Control, Git & Github



What and why ?

Git vs. GitHub vs. GitHub Desktop

- To understand the relationship :
  - **Git** is an infrastructure for versioning and merging files (it is not specific to GitHub and does not even require an online server)
  - **GitHub** provides an online service to coordinate working with Git repositories, and adds some additional features for managing projects
  - **GitHub Desktop** is just one of many GUI-based clients to make Git and GitHub easier to use

What and why ?

Version Control, Git & Github

- Some (very importnt) definitions :
  - **Commit (v.)** : to record the current state of a set of files (the current changes) in a repository
  - **Repository (n.)** : a storage area where a version control system stores the full history of commits of a project and some metadata (e.g. who changed what, when...)
  - **Conflict (n.)** : a change made by one user that is incompatible with changes made by other users
  - **Merge (v.)** : to reconcile two sets of changes to a repository

Git Fundamentals (I)

Create and initialise a repository

- Create a directory and tell Git that this directory is a repository :

```
cd ./Desktop
mkdir IP2022
cd IP2022
git init
```

Git Fundamentals (I)

Create and initialise a repository

- Check the status of the repository with :

```
git status
```

Git Fundamentals (II)

Track changes

- Inside your repository, create a file `test.txt` with :

  `touch test.txt` or `nano test.txt`

- Then, write a sentence of your choice and check status

⇒ Git knows that it's supposed to keep track of test.txt, but it hasn't recorded these changes as a commit yet

Git Fundamentals (II)

Track changes

- To tell Git to record those changes in the repository, we need to (i) add these changes and (ii) commit them :

```
git add test.txt
git commit -m "start notes on Test as a base"
```

⇒ With this command, Git stored a copy **permanently** inside the special .git directory. Good commit messages start with a brief summary of changes made in the commit.

Git Fundamentals (II)

Track changes

- **Exercise 1 :** Add a line to your file `test.txt`. Use `git diff` to see the difference. Then add + commit this change. Check status at each step. Use `git log` to get an overview of the file history.

Git Fundamentals (II)

Track changes

- `git diff` output :
  - The first line tells us that Git is producing output similar to the Unix diff command comparing the old and new versions of the file.
  - The second line tells exactly which versions of the file Git is comparing
  - The third and fourth lines once again show the name of the file being changed.
  - The remaining lines are the most interesting, they show us the actual differences and the lines on which they occur. In particular, the + marker in the first column shows where we added a line.

Git Fundamentals (II)

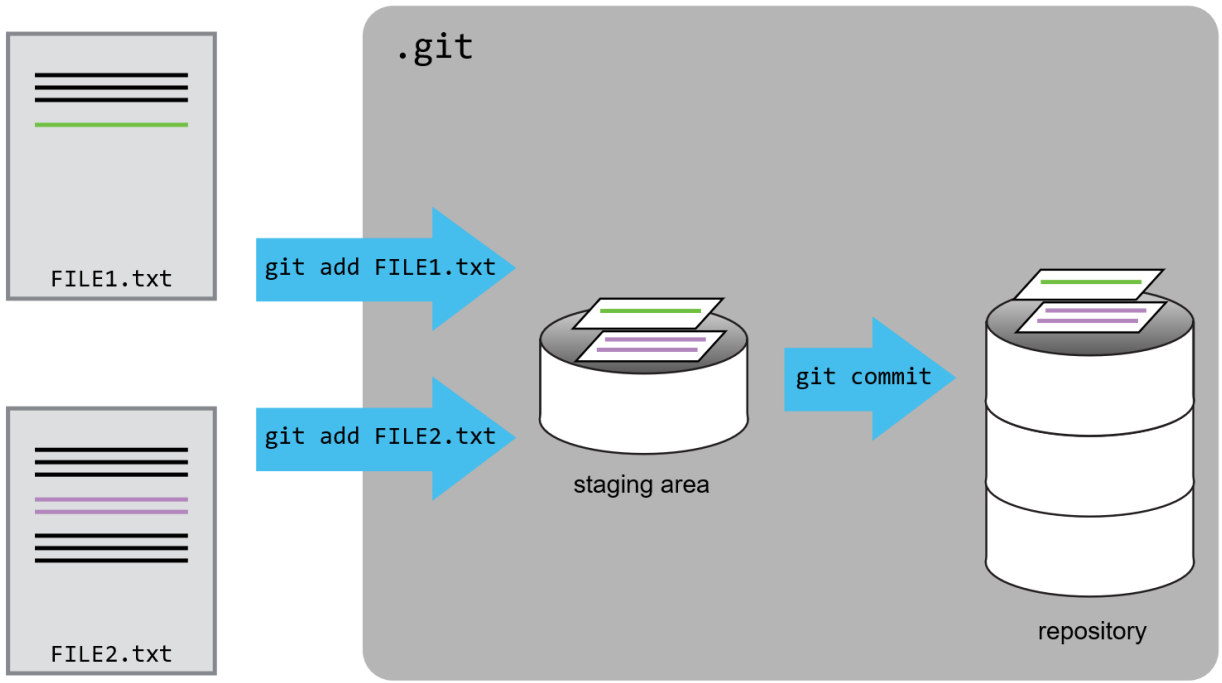
Track changes

- **Exercise 1 bis :**

  1. Add another line to your file `test.txt`
  2. create a second file `second-test.txt`
  3. check `git status` and commit only the changes made to test.txt

Git Fundamentals (II)

Track changes



Git Fundamentals (II)

Track changes

- **Important** : Git tracks only files, not directories per se
- BUT : if you have a sub-directory and want to commit the changes made in all the files in this directory, you can run :

```
git add <name-of-that-sub-directory>
```

Git Fundamentals (III)

Explore History

- Every commit has its own identifiers, the most recent is called `HEAD`

⇒ Try, `git show HEAD test.txt` and then, `git show HEAD~1 test.txt`

Git Fundamentals (III)

Explore History

- Every commit has its own identifiers, the most recent is called `HEAD`

⇒ Try, `git show HEAD test.txt` and then, `git show HEAD~1 test.txt`

- Instead of `HEAD`, can also refer to commits via their actual identifiers (only the first few characters are necessary)
- How to get the identifier ? ⇒ `git log -2`

⇒ Try `git show xxxxx test.txt`

Git Fundamentals (III)

Explore History

- **Exercise 2 :**
  1. Delete everything in your `test.txt` and replace by the line "oupsie"
  2. Use `cat test.txt` to check the file current version
  3. Use `git checkout HEAD test.txt`
  4. Re-use `cat test.txt` : what is the result ?

- Warning : If the changes were committed, use `git checkout HEAD 1 test.txt`

Git & GitHub

What and why ?

Git vs. GitHub vs. GitHub Desktop

- To understand the relationship :
  - **Git** is an infrastructure for versioning and merging files (it is not specific to GitHub and does not even require an online server)
  - GitHub provides an online service to coordinate working with Git repositories, and adds some additional features for managing projects
  - GitHub Desktop is just one of many GUI-based clients to make Git and GitHub easier to use

What and why ?

Git vs. GitHub vs. GitHub Desktop

- To understand the relationship :
  - Git is an infrastructure for versioning and merging files (it is not specific to GitHub and does not even require an online server)
  - **GitHub** provides an online service to coordinate working with Git repositories, and adds some additional features for managing projects
  - GitHub Desktop is just one of many GUI-based clients to make Git and GitHub easier to use

Git & GitHub

Remote repository

- Instead of storing all the info on our laptop, can store it on GitHub :
  1. create a new repository
  2. need to **connect this repo** with the one we have on our local machine = **making the GitHub repository a remote** for the local repository

Git & GitHub

Remote repository

- Instead of storing all the info on our laptop, can store it on GitHub :
  1. create a new repository
  2. need to **connect this repo** with the one we have on our local machine = **making the GitHub repository a remote** for the local repository

```
git remote add origin
https://github.com/Diego-de-Sousa-Rodrigues/example.git
```

- `origin` is just a name we give to the remote repository
- remote (n.) : a version control repository connected to another, in such way that both can be kept in sync exchanging commits

Git & GitHub

Remote repository

- Check the remote repository settings :

```
git remote -v
```

Git & GitHub

Remote repository

- Check the remote repository settings :

  ```
  git remote -v
  ```

- **(Crucial Step)** Save your local changes on the remote repository :

  ```
  git push origin master
  ```

Git & GitHub

Remote repository

- What are we doing with `push` ?

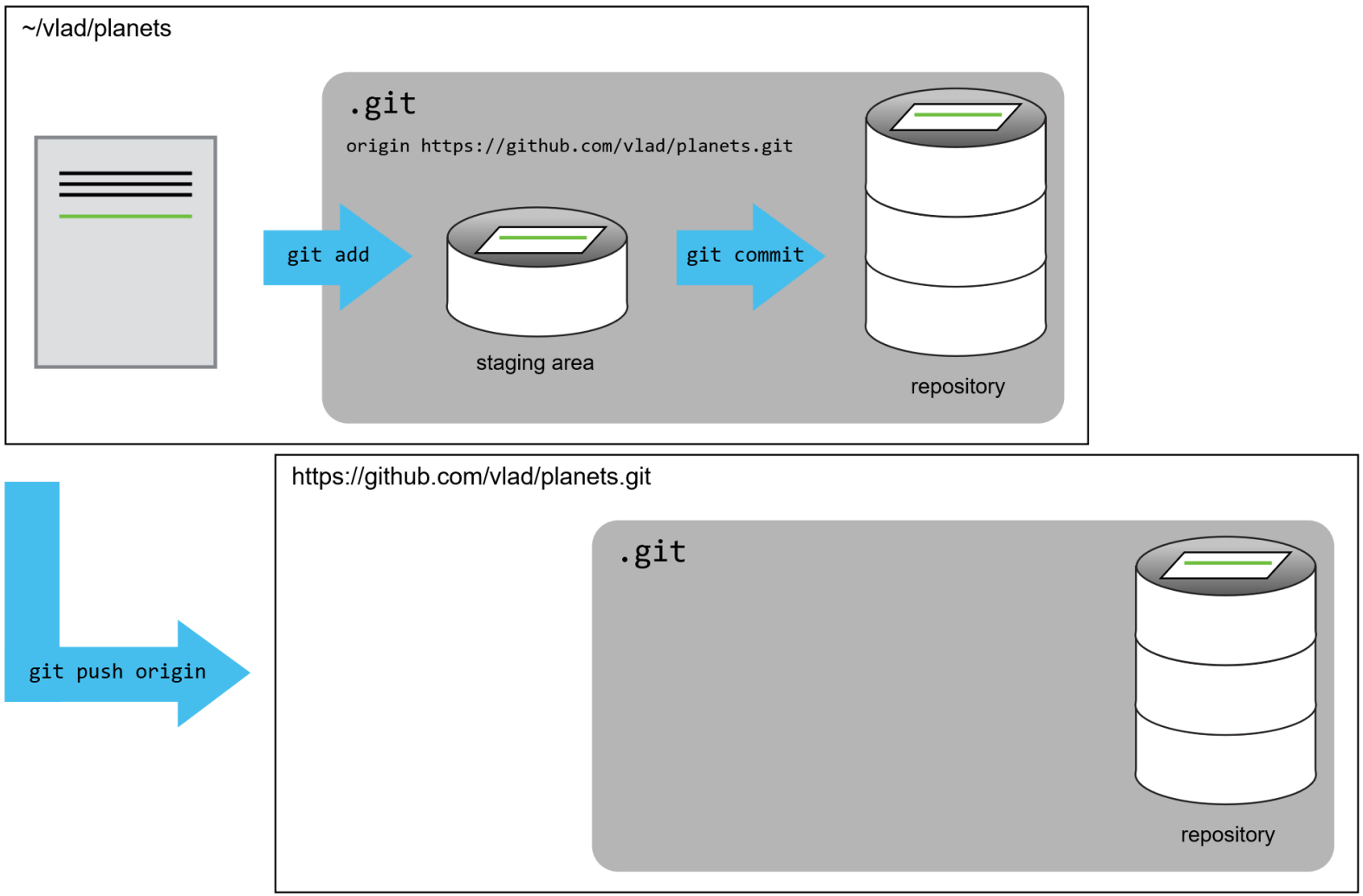
  ```
  git push origin master
  ```

  i push to the remote repository called `origin` ...

  ii ... the local branch called `master`

Git & GitHub

Remote repository



Git & GitHub

Remote repository

- If some changes are made on the remote repository (e.g. by you directly on Github, or by one of your collaborator), can pull these changes to your local repository via git pull

```
git pull origin master
```

Git & GitHub

Remote repository

- If some changes are made on the remote repository (e.g. by you directly on Github, or by one of your collaborator), can pull these changes to your local repository via git pull

  ```
  git pull origin master
  ```

- **Exercise 1 :** Try to `pull`, what do you obtain ? Then, modify the `README.md` directly on GitHub (the website), then retry `pull`.

Git & GitHub

Conflicts

- Particularly salient when working with other people !

Git & GitHub

Conflicts

- Particularly salient when working with other people !
- **Exercise 2 :**
  1. Go on your GitHub repo (website), and modify the content of README.md to "This is a remote change"
  2. Without pulling these changes, modify the README.md on your local repository to "This is a local change"
  3. Commit these changes (locally)
  4. Try to push these changes

Git & GitHub

Conflicts

- We get an error message : so far, not a conflict per se, simply that we haven't updated our local repository because both local and remote version of the file differ.
- Pull the remote version to solve issues locally :

```
git pull origin master
```

Git & GitHub

Conflicts

- We get an error message : so far, not a conflict per se, simply that we haven't updated our local repository because both local and remote version of the file differ.
- Pull the remote version to solve issues locally :

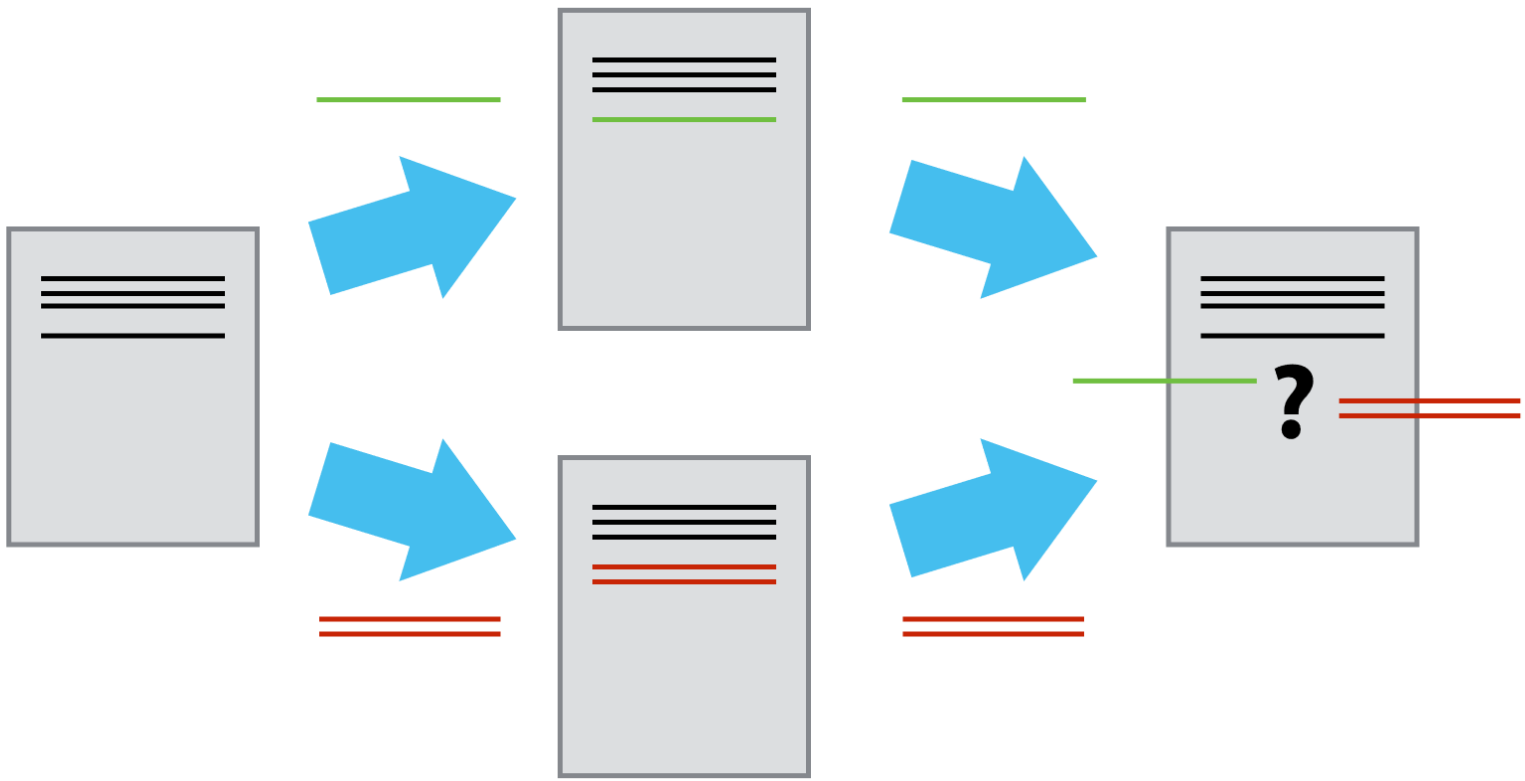
  ```
  git pull origin master
  ```

- Now we have a conflict (the changes made in one copy of the repo overlap with those made in the other copy)

$\Rightarrow$ We have to resolve it manually (with the help of Git)

Git & GitHub

Conflicts



Git & GitHub

Conflicts

⇒ How to solve manually a conflict ?

- Start to check the current file with `cat README.md`
  - Our change on the local repo is preceded by <<<<<<< HEAD
  - ======= separates the conflicting changes
  - >>>>>>> marks the end of the content download from Github (with the commit associated)
  - Up to you to remove these markers and resolve the conflict

Git & GitHub

Conflicts

- **Exercise 2b :** Try to resolve any conflict :
  1. modify locally the file(s) affected
  2. commit the changes
  3. push the new commit (= the merge)

Git & GitHub

Conflicts

- Resolving conflicts is a pain ! To minimize them :
  1. pull from upstream more frequently
  2. try as much as possible to break large files into smaller ones
  3. make smaller commits

Git & GitHub

Conflicts

- Resolving conflicts is a pain ! To minimize them :
  1. pull from upstream more frequently
  2. try as much as possible to break large files into smaller ones
  3. make smaller commits
- And also, when working in groups :
  1. clarify which collaborator is responsible for what part of the code
  2. establish a project convention to avoid stylistic conflict

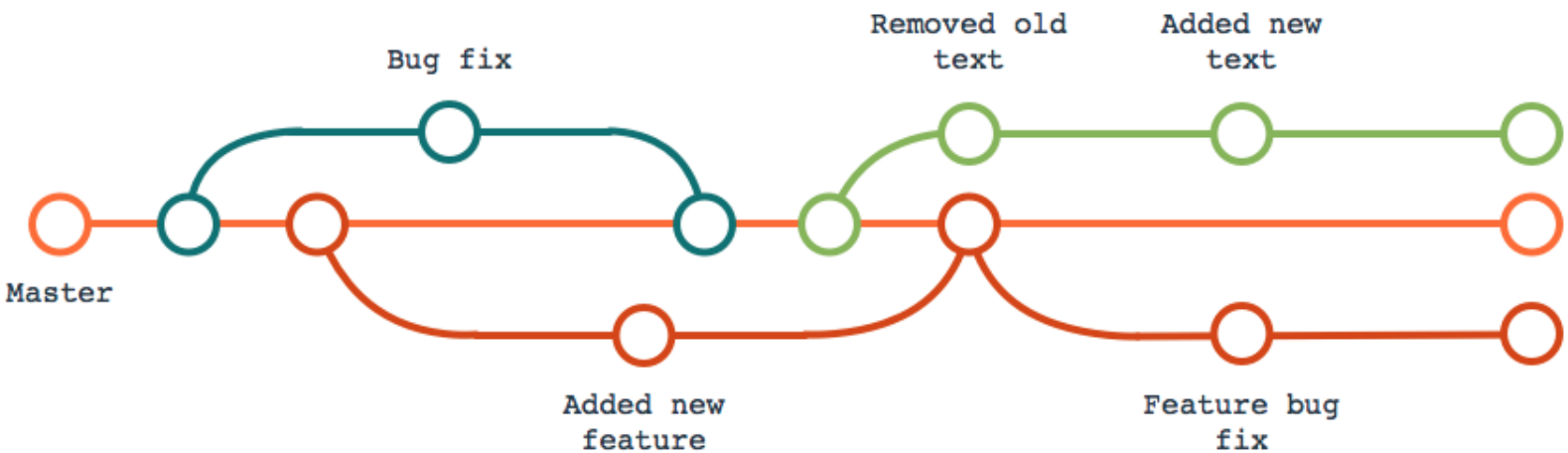
Git & GitHub

Branches

- **Branches** allow you to work on some part of the code **without affecting the main line** of development (the main branch)
- Branches are very handy for instance when want to introduce new features, but not certain if these are going to work
- To use a branch :
  1. Create a branch
  2. Work on it as much as you want (commits, push, pull etc. work similarly)
  3. When you are done, you can merge this branch to the main branch (= master) - with potential conflicts

Git & GitHub

Branches



Git & GitHub

Branches

- Some basic commands :
  1. **git branch** *# list all existing branches*
  2. **git branch <name>** *# create a branch named <name>*
  3. **git branch -m <name>** *# rename the branch we are on to <name>*
  4. **git branch -d <name>** *# delete the branch <name> IF merged*
  5. **git branch -D <name>** *# delete the branch <name> even if not merged*

- **git checkout <name>** tells Git to go to the branch name

Git & GitHub

Branches

- **Exercise 3 :** Create and work with a branch !
  1. Use `git branch` to check that you have only one branch called `master`
  2. Create a second branch with `git branch second-version` and go to this branch with `git checkout second-v`
  3. Modify your blank file (add a line), `add` and `commit` on this second branch
  4. Save your second branch version with `git push origin second-v`

Git & GitHub

Merge

- Two options :

i. own work : do a local merge via git merge, use `git merge <branch>`

ii. will add the changes made in to the branch we are currently on collab : do a pull request on Github

Git & GitHub

Merge

- Two options :

i. own work : do a local merge via git merge, use `git merge <branch>`

ii. will add the changes made in to the branch we are currently on collab : do a pull request on Github

- With our previous example

1. `git checkout master`
2. `git merge second-v`

Git & GitHub

Fork

- **Forking** : cloning a Github project and registering it under your Github username
- Why is it useful to fork ?
  1. Can make changes without affecting the original project
  2. Can update your version with changes made to the original project (= upstream)
  3. Can make changes upstream (= a pull request)

Git & GitHub

Fork

- How to fork a repository ?
  1. Go to the repository of your choice, and fork it
  2. Create a local clone of your work
  3. Setup an upstream repository

Git & GitHub

Fork

- How to fork a repository ? An example :
  1. Go to `https://github.com/Diego-de-Sousa-Rodrigues/IP2019` and click on *fork*
  2. In your directory of your choice, make a local copy of your repo (a clone). `git clone` will automatically setup your forked repository as the origin remote

     ```
     git clone https://github.com/your-name/IP2019
     ```
  3. Configure Git to pull changes from the upstream repository :

     ```
     git remote add upstream
     https://github.com/Diego-de-Sousa-Rodrigues/IP2019.git
     ```
  4. Check it worked by running with `git remote -v`
  5. To sync your local remote : `git pull upstream master`

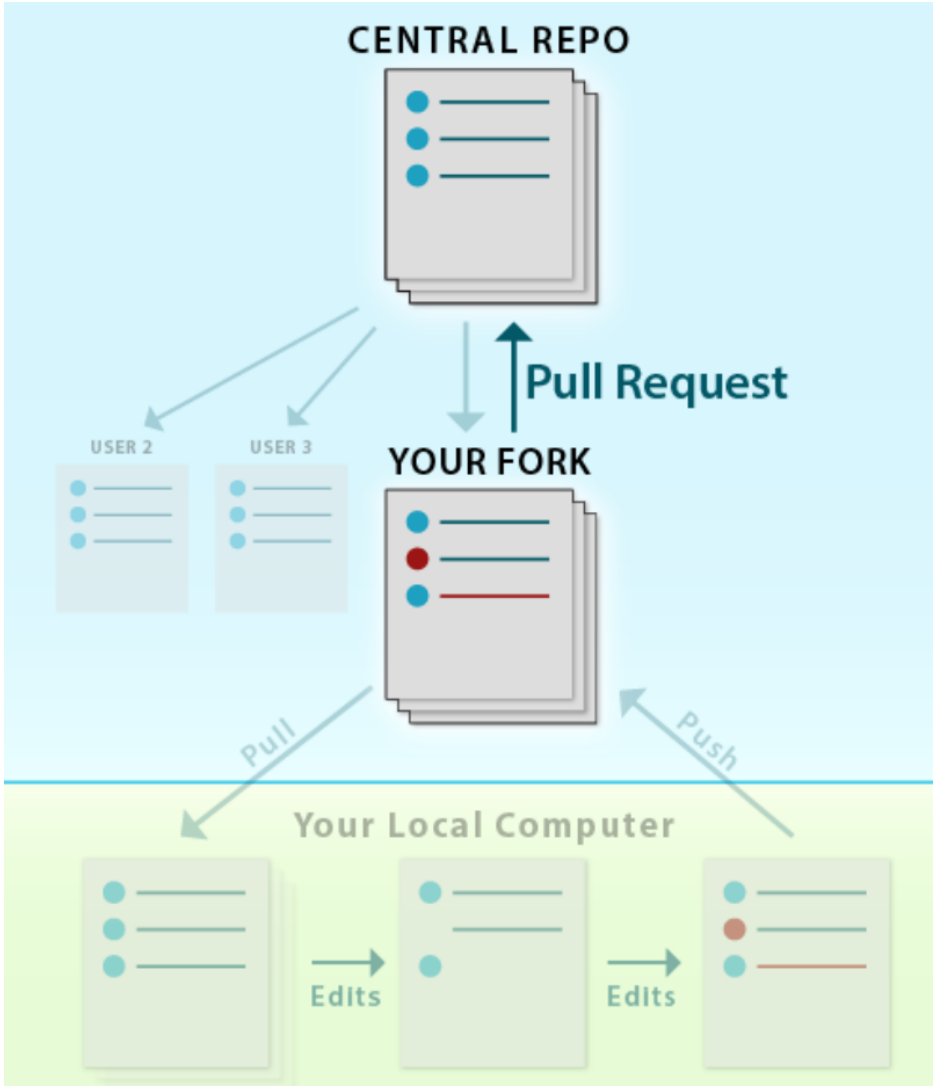
Git & GitHub

Pull Request

- Pull request : Similar to pushing to your remote repository, except that your pushing to somebody else's
- Allows to discuss and review the changes made, and add follow-up commit before the changes are merged
- To do a pull request :
  1. Push some changes to your remote personal repository
  2. Go on its Github page, and click on pull request
  3. Add comments etc.

Git & GitHub

Pull Request



Git & GitHub

Pull Request

- Homework : we will now use pull request for homework
- Idea :
  1. do the homework on your computer
  2. push it to your remote repository
  3. pull request to my repository so that I can see what you've done