Introduction to Programming

Lecture 4-6: Version Control

Clément Mazet-Sonilhac
clement.mazet@sciencespo.fr

Sciences Po Paris

Disclaimer

- Most of the material is drawn from the excellent course prepared by software carpentry (adapted by Hugo Lhuillier for the last year course)
- 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.

⇒ 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

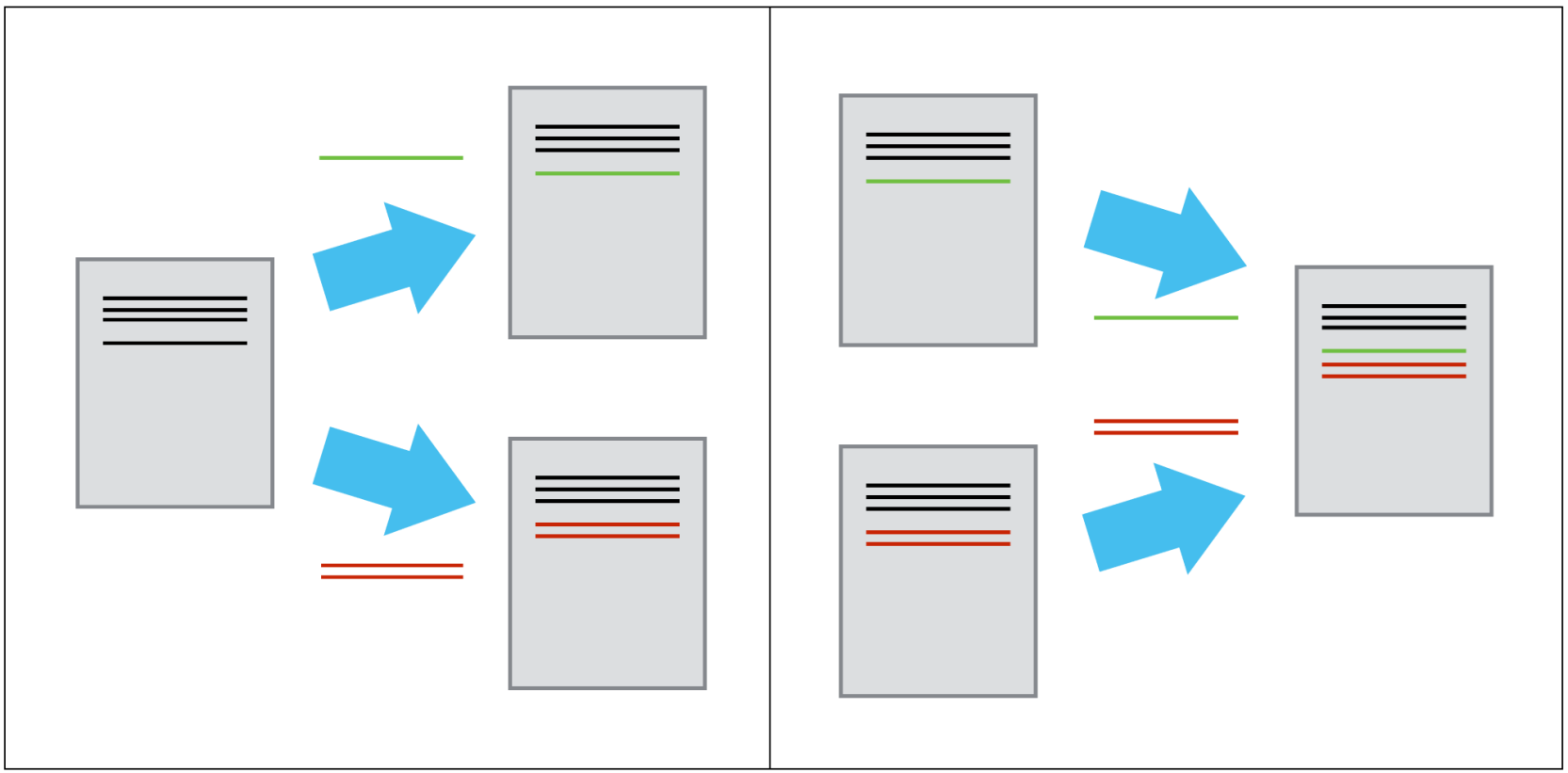
- A **version control** system is a tool that keeps track of these changes for us, effectively creating different versions of our files.

⇒ 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/GitHub
mkdir IP2019
cd IP2019
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 log` 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.

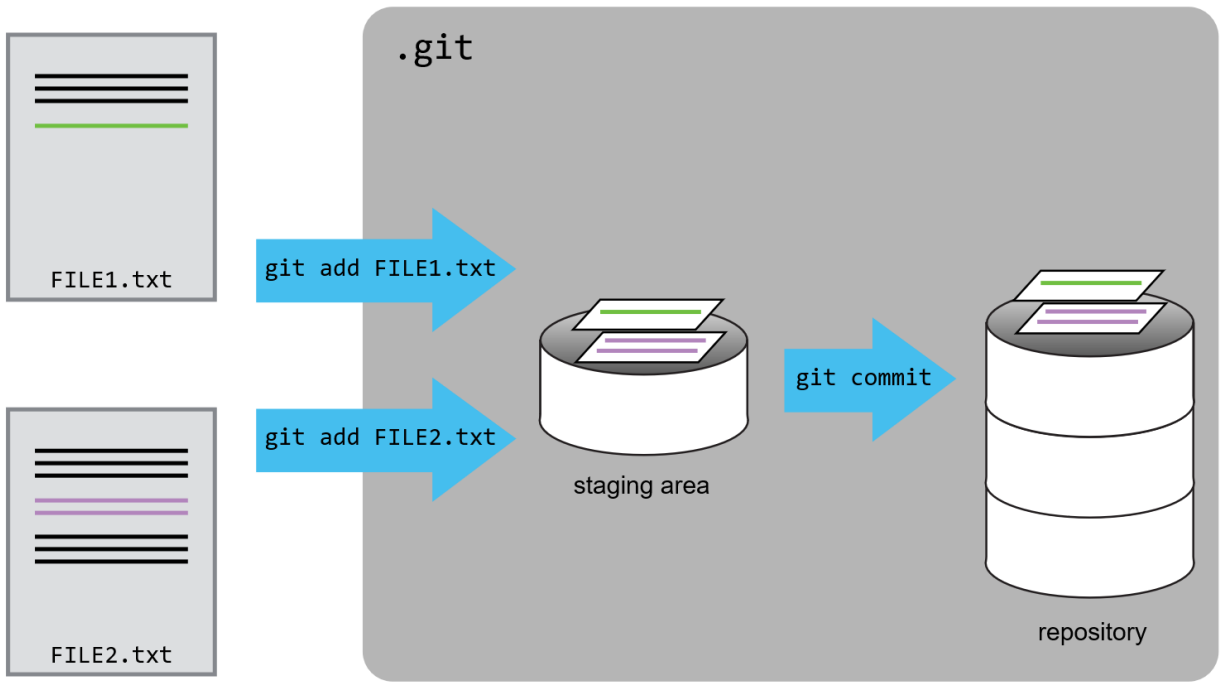
Git Fundamentals (II)

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 mars.txt`

Git Fundamentals (III)

Explore History

- **Exercise 3 : Ignoring things**

  1. create blank files a.dat and b.dat
  2. create a new directory called results, and in it, blank files, a.out and b.out
  3. tell Git to ignore these four files

- For the last step, you should create a file `.gitignore` (inside : *.dat results/). Do not forget to add and commit !