Introduction to Git

IN104: Projet Informatique[2]

Natalia Díaz Rodríguez

ENSTA Paris, Institut Polytechnique Paris

April 8, 2020

[2]♡Acknowledgment: Florence Carton, François Pessaux. [Slides extended from:] Antonin Raffin & Ugo Vollhardt

Table of contents

What are Version Control Systems (VCS)?

- A VCS tracks the history of changes as people and teams collaborate on projects together.
- As the project evolves, teams run tests, fix bugs, and contribute new code
  - with confidence that any version can be recovered at any time.
- Developers can review project history to find
  - Which changes were made?
  - Who made them?
  - When?
  - Why were they needed?

Distributed Version Control Systems (DVCS)

- Git: an example of a DVCS commonly used for open source and commercial software development.
- DVCSs allow full access to
  - Every file, branch, and iteration of a project
  - A history of all changes.
- Git and other VCSs:
  - Help team members stay aligned through a **unified and consistent view of the project** while working independently.
  - **Don't need constant connection** to a central repository: Developers can work anywhere and **collaborate asynchronously** from any time zone.
- Without version control, team members are subject to:
  - Redundant tasks
  - Slower timeline
  - Multiple copies of a single project.

Git

Why Git?

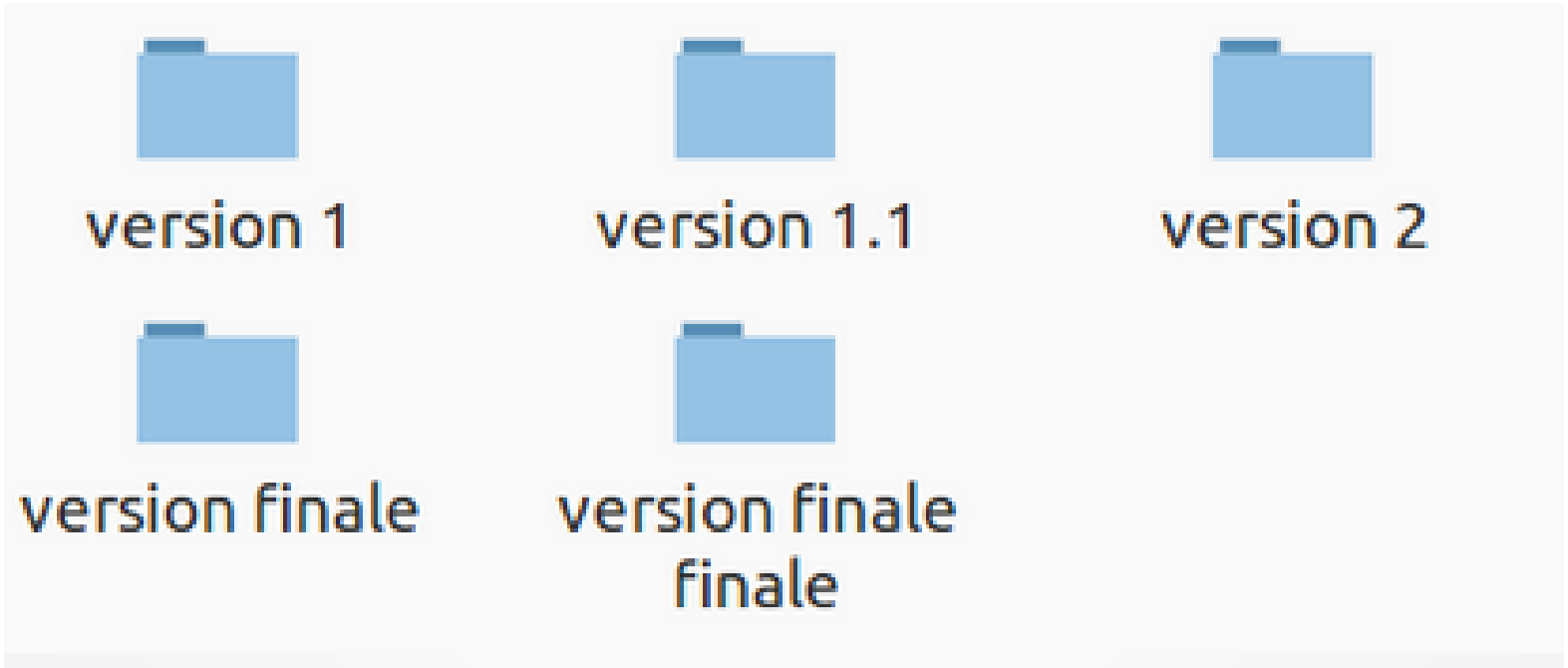


Figure: Avoiding the nightmare

Git

Many revision control systems: **Why Git?**

- Need a place to store code when team size > 1
- Git has over 10M repos
- Github offers free private repos (now for everyone!)
- Allows every developer to work on the same file (and have a local copy)

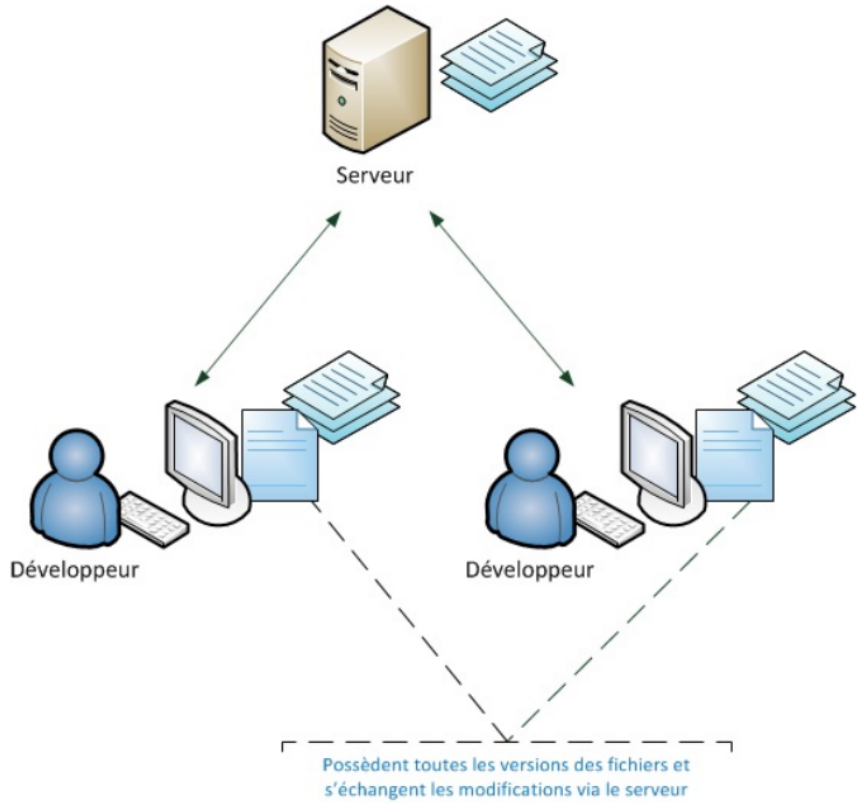


Figure: Git
`www.openclassrooms.com/courses/gerer-son-code-avec-git-et-github`

Initialization: Follow the steps carefully if you want to save time!

We follow steps in the Github guide *Generating a new SSH key and adding it to the ssh-agent*[3]

- SSH Key
  1. Generate an SSH key (accept parameters by default, Don't introduce pass code)

```
$ ssh-keygen -t rsa -C "name.surname@ensta-paris.fr"
```

  2. Show the generated public key

```
$ cat ~/.ssh/id_rsa.pub
```

  3. Paste the generated key in the Github interface, section 'My SSH Keys'. (one key required per computer you link to your github account)
- One time config

```
$ git config --global user.name "Diaz Natalia"
$ git config --global user.email "name.surname@ensta-paris.fr"
```

[3] https://help.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

A) Creating a project (when you have local work already)

1. Create a folder in your computer and initialize it

```
$ mkdir project_folder
$ cd project_folder
$ git init
```

2. Create a new project in GitHub.com GUI[4]
3. Add a new file

```
$ touch README.md
$ git add README.md
$ git commit -m "first commit"
```

4. Link your local folder to the Git project

```
$ git remote add origin git@github.com:your_username/your_repo_name
```

5. Push (*upload*) the README.md over Git

```
$ git push
```

or (if first time -see FAQ if issues-):

```
$ git push --set-upstream origin master
```

[4] https://help.github.com/en/articles/adding-an-existing-project-to-github-using-the-command-line

A) Creating a project (when you have local work already)

- At this point, the project is created and initialized.
- Each person joining this project must be added as *collaborator* member through the Github web interface, and simply should clone the project
  - Prefer the SSH url address against the HTTPS one.
  - The folder will be created in your current location, launching this command:

```
$ git clone git@[srv_url]:[username]/[project].git
e.g.:
$ git clone git@github.com:ndiaz/project.git
```

B) Creating a project (fastest)

- Create a new repo in Github.com GUI once logged in (Upper right '+' button)
- Add *collaborator* members through the Github web interface, and simply clone the project[5]:

```
$ git clone git@[srv_url]:[username]/[project].git
e.g.:
$ git clone git@github.com:ndiaz/project.git
```

- Now you can create files inside the *project* folder

[5] As in case A, Prefer the SSH url address against the HTTPS one (the folder will be created in the location where you are located when launching this command)

Commands

- `Add`: adds file(s) for the next commit

```
$ git add my_file1 my_file2
$ git add --all
```

- `Commit`: *saves* files added previously

```
$ git commit -m 'Comment over the performed changes'
```

- `Pull`: get the changes others made

```
$ git pull
```

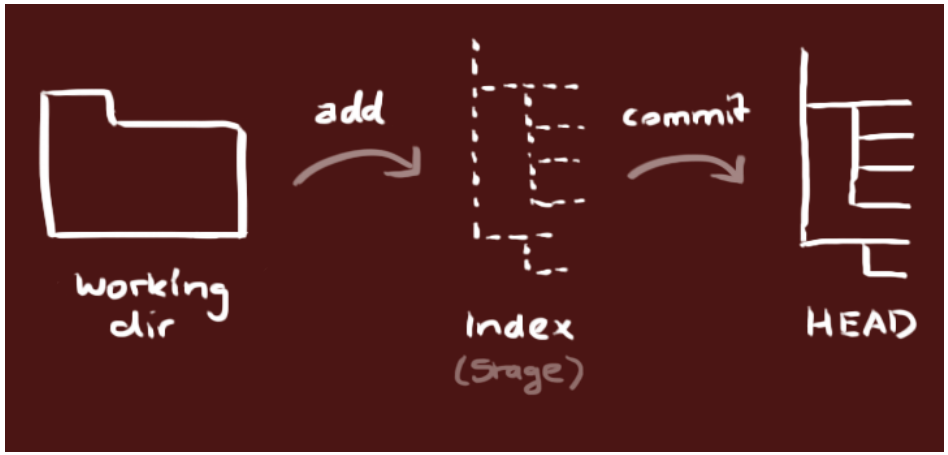
- `Push`: upload all changes on Git

```
$ git push
```

- `Status`: shows the status of the git local folder (modified/to add/staged files...)

```
$ git status
```

Reminder



ALWAYS do *pull* before *push*!![6]

[6] Anyway, impossible to push before pull if modifications exist remotely

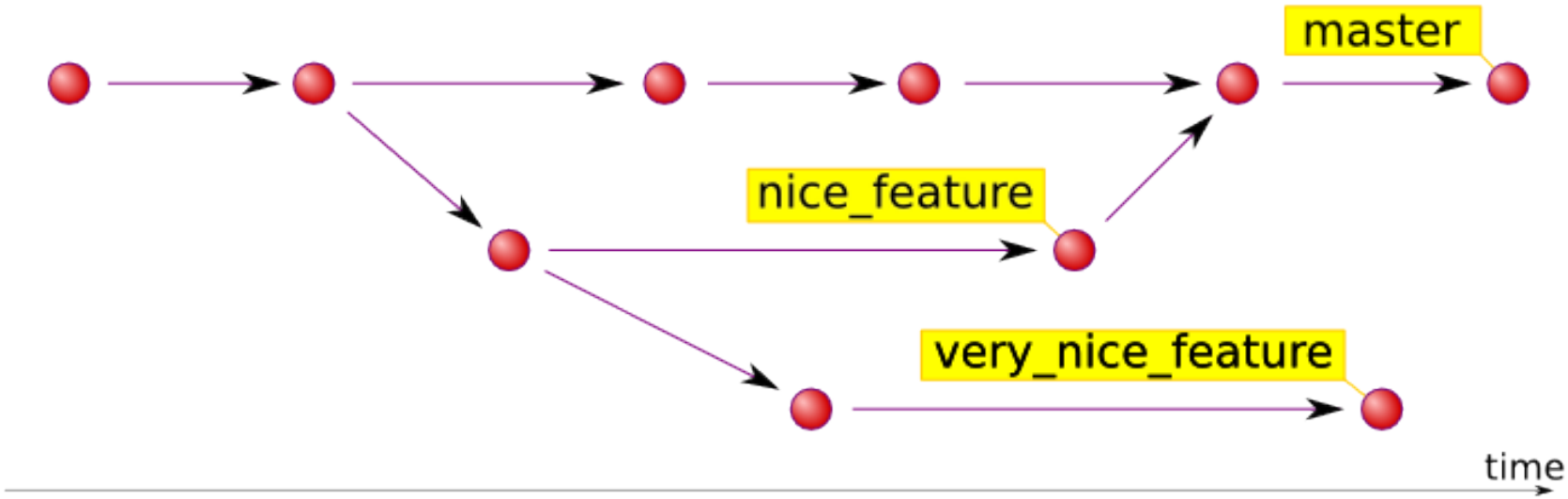
Example: common situation

Example: 2 bugs to solve:
bug 1: requires modifying file a.py and b.py $\rightarrow$ bug 1 solved
bug 2: requires modifying file c.py $\rightarrow$ bug 2 solved

```
$ git add a.py b.py
$ git commit -m 'bug 1 solved!'
$ git add c.py
$ git commit -m 'bug 2 solved!'
$ git pull
$ git push
```

Branches: pointers to commits

They allow you to work on different features to later **merge** your work:



Merge conflict

```
$ git merge my_branch
Auto-merging test_file.md
CONFLICT (content): Merge conflict in test_file.md
Automatic merge failed; fix conflicts and then commit the result.
```

Inside test_file.md you will see:

```
<<<<<<< HEAD
# some modifications here created conflicts
=======
# blablabla! breaking some code blabla
>>>>>>> my_branch
```

→ **HEAD**: modifications in *master* branch

→ **my_branch**: modifications of *my_branch*

Edit it to keep the right changes. Once problems are solved:

```
$ git add test_file.md
$ git commit -m 'Solved merge conflict in test_file.md'
$ git push
```

Going back in time: recovering a past version

- Abandon changes done in a particular file

```
$ git checkout -- my_file
```

- Cancel the changes done in last commit

```
$ git revert
```

Going back in time: recovering a past version

Panic mode?

If you get stuck with a bunch of unintentional merge errors and want to reset your repo:

```
git fetch origin
git reset --hard origin/master
```

Note that you will lose EVERYTHING unsaved (or maybe even saved) in your repo! Keep a backup copy.

Practical time! The Lab session consists of:

You need to:

1. Learn GIT through the excellent GitHub Hello World Guide[7], GitHub Flow Guide[8] and GitHub Handbook Guide[9].
2. Find a classmate and form a team of 2 (if you really are alone, join a team of 2, but never alone!).
3. Create a PRIVATE repository called *IN104_NameA_SurnameA-NameB_SurnameB* (include all team members), add as collaborators your team mate(s) and your Teaching Assistant (TA).
4. Create a folder inside called "GIT". Inside, each of you will create a Python program (*hello_world.py* and *bye_world.py*, respectively) that your mate needs to retrieve, modify and commit. You need to retrieve the changes your mate did to the file you created after his commit.
5. Send the link to your repository to your TA within 1 week max[10].

[7] `https://guides.github.com/activities/hello-world/`
[8] `https://guides.github.com/introduction/flow`
[9] `https://guides.github.com/introduction/git-handbook/`
[10] Your TD email is in Lecture 0 (associated to the project you were assigned to)

Practical time! What is essential to pass this TD?

1. COMPULSORY: What we will evaluate: 1 commit each team member, for each of the files requested.
2. OPTIONAL: If you already master git, and want to learn about branches, you can do 2 commits each, the first saying "hello/bye world", the second saying "merge the pull-request".
   In this case, we would also check how many of you were curious and did a little extra, e.g:
   - customize your gitconfig a little,
   - learn a bit of markdown on the way,
   - navigate the wiki or issue tabs in github,
   - read the 7 rules of a commit message: `https://chris.beams.io/posts/git-commit/#seven-rules`, etc.

Practical time! What is essential to pass this TD?

1. The same game of GIT commits in your collaborative team project will be evaluated in your final repository.
2. If you finish on time, play more advanced GIT in `https://gitexercises.fracz.com` and `https://www.codecademy.com/courses/learn-git/lessons/git-branching/exercises/branching-overview`.
3. Q: Should I use Gitlab or Github?
   A: We strongly encourage the use of GitHub
   (If you really really want to use Gitlab, use `gitlab.ensta.fr` and set up your account and SSH Keys -as in `https://gitlab.com/help/ssh/README`)

Useful links

- First time user/computer: Generating a new SSH key and adding it to the ssh-agent[11]
- GIT Cheat Sheets:
  `https://education.github.com/git-cheat-sheet-education.pdf`
  `https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet`
  In French: `https://github.com/UgoVollhardt/CheatSheetGit/blob/master/CheatSheet.pdf`
- How to undo (almost) everything in Git `https://blog.github.com/2015-06-08-how-to-undo-almost-anything-with-git/`
- Openclassroom: Manage your source code with Git and Github (in FR):
  `www.openclassrooms.com/courses/gerer-son-code-avec-git-et-github`

[11] `https://help.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent`

Useful links

Interactive tutorials to learn by doing:

- `https://gitexercises.fracz.com`
- `https://www.codecademy.com/courses/learn-git/lessons/git-branching/exercises/branching-overview`
- `https://learngitbranching.js.org/`
- `https://try.github.io/levels/1/challenges/1`

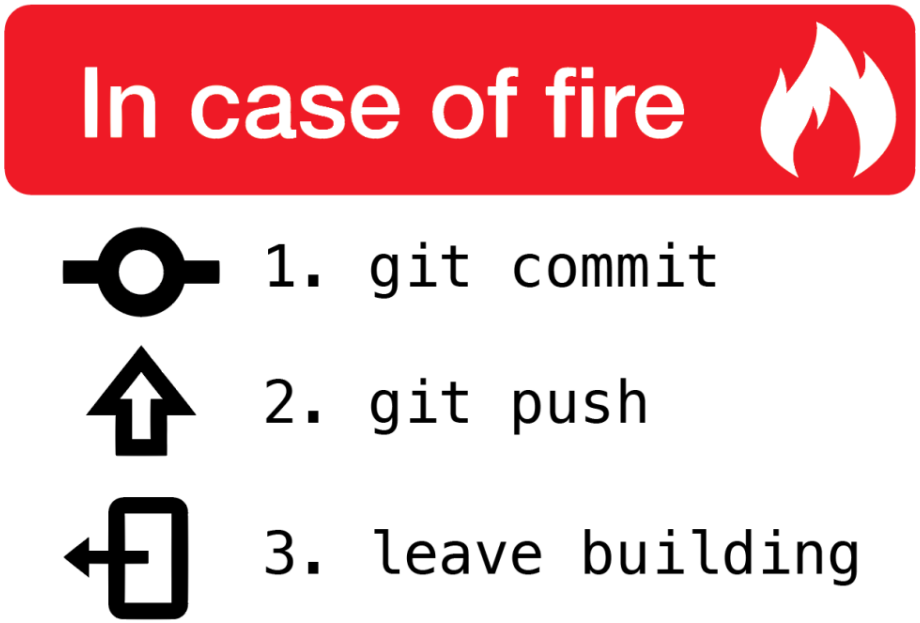
Per-command Atlassian guide (e.g. checkout vs fetch vs pull):

- `https://www.atlassian.com/git/tutorials/syncing/git-fetch`

Useful links

- Antonin Raffin tutorials - Intro to Git: http://slides.com/antoninraffin/git and Git intermediate: http://slides.com/antoninraffin/git-intermediate
- http://users.humboldt.edu/smtuttle/s12cis492/492guide-to-git.pdf
- https://github.com/git-tips/tips#everyday-git-in-twenty-commands-or-so
- https://tutorialzine.com/2017/11/10-useful-git-tips

To Conclude



Appendix


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

FAQ

- Q: git merge error

```
$ Merge branch 'master' of github.com:NataliaDiaz/repo-name
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

A: To solve it in linux: Ctrl+X (Exit). In Vim editor:
press "i"
write your merge message
press "esc"
write ":wq"
then press enter
You should see something like:

```
Merge made by the 'recursive' strategy.
 YourChangedFile.txt | 57 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 57 insertions(+)
```

FAQ

- Q: First time pull:

```
git pull
There is no tracking information for the current branch. Please specify
    git pull <remote> <branch>

If you wish to set tracking info for this branch you can do so with:

    git branch --set-upstream-to=origin/<branch> master
```

A:

```
$ git branch --set-upstream-to=origin/master  master

$ git pull --allow-unrelated-histories
```

FAQ

- Q: First time push when associating local repo to a remote:

```
$ git pull
fatal: refusing to merge unrelated histories
```

A:

```
$ git pull --allow-unrelated-histories

Merge made by the 'recursive' strategy.
```

FAQ

- Q: Associating local repo to a remote:

```
$ git remote add origin git@github.com:your_username/your_repo.git
fatal: remote origin already exists.
```

A: To reset your origin:

```
$ git remote set-url origin git@github.com:your_username/your_repo.git
```