Introduction to Git

IN104: Projet Informatique[1]

Natalia Díaz Rodríguez

ENSTA ParisTech

March 2019

---

[1]Acknowledgment ♡: Slides extended from Florence Carton & Antonin Raffin

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 the confidence that any version can be recovered at any time.
- Developers can review project history to find
  - Which changes were made?
  - Who made the changes?
  - When were the changes made?
  - Why were changes 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.
  - Dont need a 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

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!)
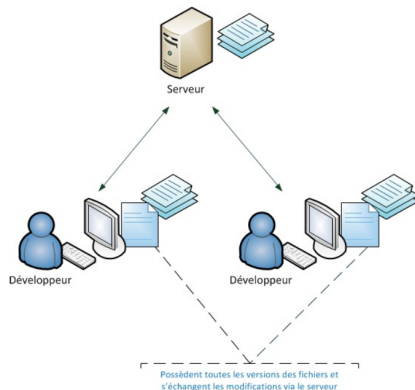- Allows every developer to work on the same file (and have a local copy)

Figure: Git[a]

---

[a] www.openclassrooms.com/courses/gerer-son-code-avec-git-et-gith

Git

Why Git?

Figure: Avoiding the nightmare

Git

From its creator, Linus Torvalds[2]:

```
        GIT - the stupid content tracker

"git" can mean anything, depending on your mood.

 - random three-letter combination that is pronounceable, and not
   actually used by any common UNIX command.  The fact that it is a
   mispronounciation of "get" may or may not be relevant.
 - stupid. contemptible and despicable. simple. Take your pick from the
   dictionary of slang.
 - "global information tracker": you're in a good mood, and it actually
   works for you. Angels sing, and a light suddenly fills the room.
 - "goddamn idiotic truckload of sh*t": when it breaks

This is a stupid (but extremely fast) directory content manager.  It
doesn't do a whole lot, but what it _does_ do is track directory
contents efficiently.
```

Figure: GIT: *Global information tracker*

---

[2]Source: https:
//github.com/git/git/blob/e83c5163316f89bfbde7d9ab23ca2e25604af290/README

Initialization

We follow steps in the 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-paristech.fr"
     ```

  2. Show the generated 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-paristech.fr"
  ```

---

[3]https://help.github.com/en/enterprise/2.16/user/articles/
generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Creating a project

1. Create a folder in your computer and initialize it

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

2. Create a new project in Git

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:ndiaz/project.git
   ```

5. Push (*upload*) the README.md over Git [4].

   ```
   $ git push -u origin master
   ```

---

[4] `-u/ --set-upstream` adds an upstream (tracking) reference so to set origin as the upstream remote in your git config (this way you don't have to manually specify the remote every time you run git push, and so you can run git push without arguments)

Creating a project

- At this point, the project is created and initialized.
- Each person joining this project must be added as *collaborator* member through the Github interface, and simply must do:

```
$ git clone ssh_not_https_link_of_the_project
$ git clone git@github.com:ndiaz/project.git
```

- The folder will be created over his PC (same location where it was located when launched command)

Commands

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

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

- Commit: 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
```

Example:
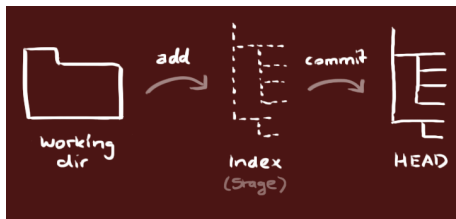
2 bugs to solve:

bug 1: file a.py and b.py modified $\rightarrow$ bug 1 solved

bug 2: 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
```

Commands

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

```
$ git status
```


Note: ALWAYS do *pull* before *push*!

Branches

- List the branches

```
$ git branch
* master
```

- Create a branch

```
$ git branch my_new_branch
$ git branch
* master
  my_new_branch
```

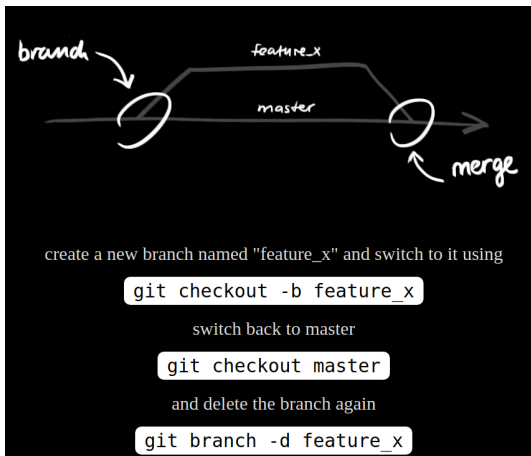- Place yourself in my_new_branch

```
$ git checkout my_new_branch
```

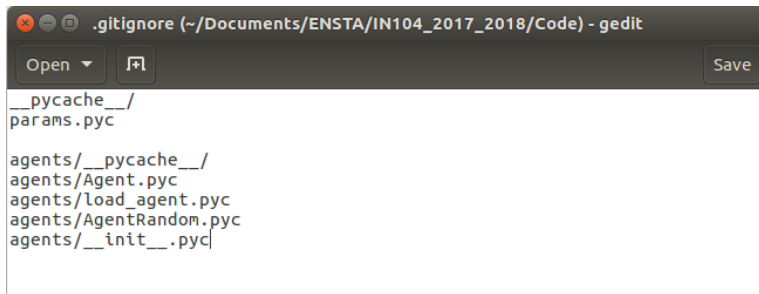- Fuse the branches: *merges* my new branch into master

```
$ git checkout master
$ git merge my_new_branch
```

branches

Creating and deleting branches: **Re-cap**:

create a new branch named "feature_x" and switch to it using

```
git checkout -b feature_x
```

switch back to master

```
git checkout master
```

and delete the branch again

```
git branch -d feature_x
```

Ignore files we don't want uploaded in Git: **.gitignore**

Git blame

Who introduced this bug?

Git blame

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.
```

file test_file.md

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

$\rightarrow$ **HEAD**: modifications in master branch
$\rightarrow$ **my_branch**: modif of my_branch (others)
Once the 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.

Going back in time: recovering a past version

Practical time!

In the lab you will:

1. Learn GIT through the excellent GitHub Hello World Guide[5], GitHub Flow Guide[6] and GitHub Handbook Guide[7].

2. Create a PRIVATE repository called *IN104_NameA_SurnameA_NameB_SurnameB* (for all team members, max 3 members), add as collaborators your team mate(s) and your TA. Create a folder inside called GIT that contains a sample hello_world.py Python program that your mate needs to modify, commit, and you need to retrieve the changes he did.

3. Show the program modified by both members and the commits in github by both team partners to your Teaching Assistant (TA)

4. Send the link to your repository to your TA

---

[5]https://guides.github.com/activities/hello-world/
[6]https://guides.github.com/introduction/flow
[7]https://guides.github.com/introduction/git-handbook/

Practical time!

In the lab you will:

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

To Conclude

In case of fire 🔥

1. `git commit`

2. `git push`

3. `leave building`

Action time!

Useful links

- First time user/computer: Generating a new SSH key and adding it to the ssh-agent[8]
- GIT Cheat Sheets:
  https://education.github.com/git-cheat-sheet-education.pdf
  https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet
- Oh shit git! http://ohshitgit.com/
- 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
- Learn by doing: https://gitexercises.fracz.com
  https://www.codecademy.com/courses/learn-git/lessons/git-branching/exercises/branching-overview

---

[8]https://help.github.com/en/enterprise/2.16/user/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Useful links: Git

- 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://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf`
- `https://github.com/git-tips/tips#everyday-git-in-twenty-commands-or-so`
- `https://tutorialzine.com/2017/11/10-useful-git-tips`

Useful links: Going beyond

- Install Python libraries and Master Python:
  http://musicinformationretrieval.com/python_basics.html
- Python Numpy http://cs231n.github.io/python-numpy-tutorial/
  and IPython tutorials http://cs231n.github.io/ipython-tutorial/
- Iterate fast installing Jupyter notebooks http://jupyter.org/install
  and get good at IPython: http:
  //musicinformationretrieval.com/get_good_at_ipython.html
- The quartet of NumPy, SciPy, Matplotlib, and IPython is a popular
  combination in the Python world. Numpy Basics:
  http://musicinformationretrieval.com/numpy_basics.html
- Numpy Tutorial: http:
  //scipy.github.io/old-wiki/pages/Tentative_NumPy_Tutorial