

Git & Github

A Gérad Tutorial

Antoine Prouvost

with help from

Viviane Rochon Montplaisir, Mathieu Besançon, Aurélien Serre

Polytechnique Montréal

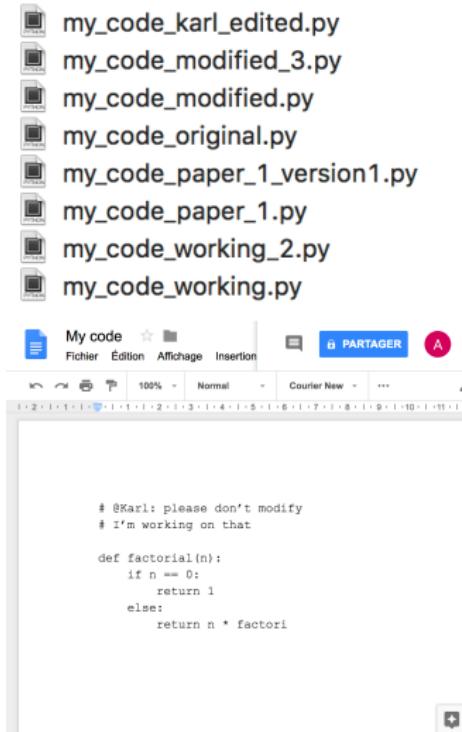
17th of May 2018

Motivation

Problem setting

- ▶ Organize
- ▶ Collaborate
- ▶ Back-up
- ▶ Publish

For code!



The image shows a file explorer on the left listing several Python files:

- my_code_karl_edited.py
- my_code_modified_3.py
- my_code_modified.py
- my_code_original.py
- my_code_paper_1_version1.py
- my_code_paper_1.py
- my_code_working_2.py
- my_code_working.py

Below the file list is a code editor interface with the following elements:

- File tab (highlighted)
- Fichier
- Édition
- Affichage
- Insertion
- B PARTAGER
- A button
- Zoom controls (100%, Normal)
- Type selection (Courier New)
- Font size controls
- Code preview area containing:

```
# @Karl: please don't modify
# I'm working on that

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factori
```

Git

Manage code history, versions, and collaborate on projects !



- ▶ Version control system
- ▶ Free & open source
- ▶ Powerful! Used by most software developers on the most complex projects.
- ▶ Yet easy(ish) to use for simpler projects

Git

Plenty of (great) resources!

- ▶ [Git website](#) (doc, tools, community)
- ▶ [Cheat-sheet from Github](#)
- ▶ [Visualization of commands and states](#) (ndpsoftware)
- ▶ [Try Git: Interactive commands](#)
- ▶ [Atlassian Tutorial](#)
- ▶ And so many more!

“



ShareLaTeX

= LATEX +



”



GitHub

= git +



Big picture

- ▶ Multiple repository on one computer
- ▶ Repo on same machine are not related
- ▶ One repo per project



Project Management

Git commands

```
$ git <command>
```

```
$ git <command> -h
```

The help on the command.

Create a new empty local repository

```
$ git init  
Initialized empty Git repository in ...
```

One folder (with subfolders) \leftrightarrow one git repository.

```
$ ls -a  
.  ..  .git
```

All the repository info are stored in the .git folder. Delete this folder to the repository (but not the files).

Repository status

```
$ git status
On branch master

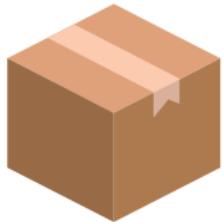
No commits yet

nothing to commit (create/copy files and use
 "git add" to track)
```

Very verbose, use often (and read the output) !

The commit

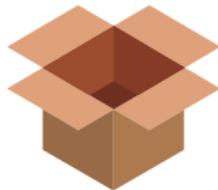
- ▶ Smallest modification in the project,
- ▶ Checkpoint in your project,
- ▶ Save the current changes to the tracked¹ files
- ▶ Always possible to come back to a commit
(immutable²)!



¹Not all files are tracked by git.

²Possible to mutate history but you need to want it.

The staging area



- ▶ The staging area is the place we put changes,
- ▶ The only point is to make a commit,
- ▶ Sealing the box is making a commit.

staging area commands

```
$ git add
```

```
$ git rm
```

```
$ git mv
```

```
$ git status
```

Again, use this one, a lot.

Actually making a commit

```
$ git commit
```

```
$ git commit -m "A useful description"
```

- ▶ All commit must have a description
- ▶ Choose appropriately
- ▶ All commit have a unique hash (identifier)

Commit names

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE.	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJ5DKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Interacting with history



```
$ git log
```

Show branch history

```
$ git diff
```

Show difference between commits / current modifications.

[Advanced] Rollbacks

```
$ git checkout <sha>
```

Go to specific commit - Enter detached HEAD state

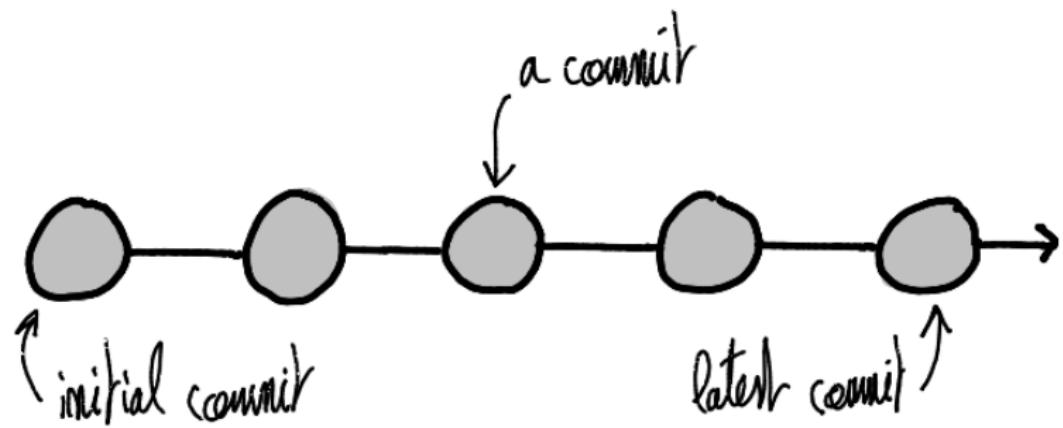
```
$ git revert <sha>
```

Make a commit that undoes another one.

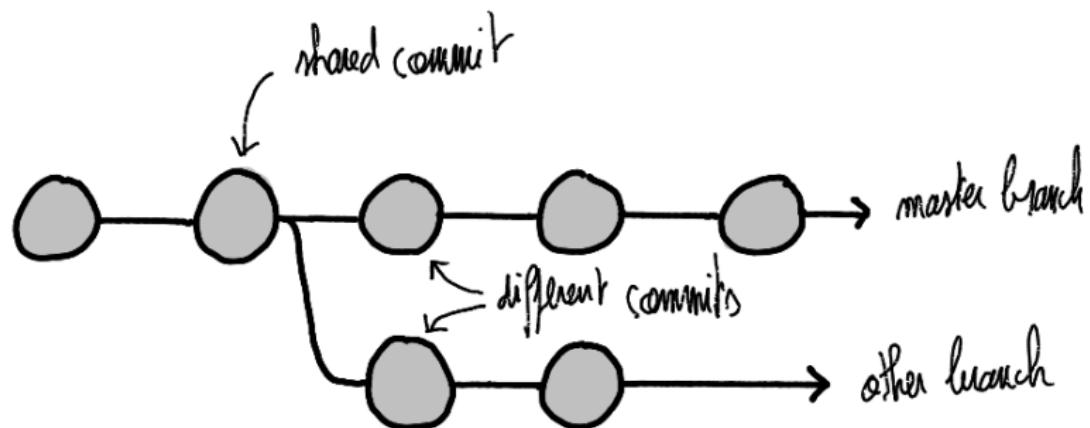
```
$ git reset
```

Multiple usages: unstage, rewrite history

Branch representation



Branches



Branches

```
$ git branch
```

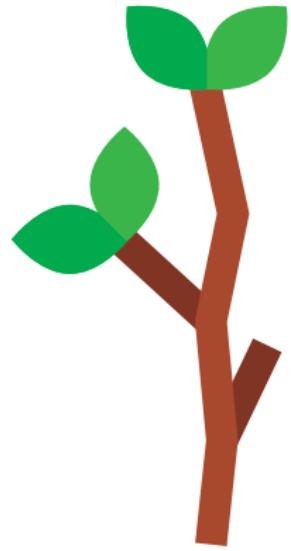
List branch

```
$ git branch <name>
```

Create new branch

```
$ git branch -d <name>
```

Delete branch



Branches

- ▶ You are on one branch at a time;
- ▶ The commit you make are appended to the current branch;
- ▶ You must not have uncommitted changes to be able to change branch.

```
$ git checkout <name>
```

Change branch

```
$ git checkout -b <name>
```

Create new branch and change to the created branch.

[Advanced] Stashing

- ▶ If you don't want to commit changes, you can tell git to put them aside (called stashing);
- ▶ You can retrieve them later in the same branch or another one;
- ▶ You can stack up stashes (like branches).

```
$ git stash
```

Put uncommitted changes aside

```
$ git stash pop
```

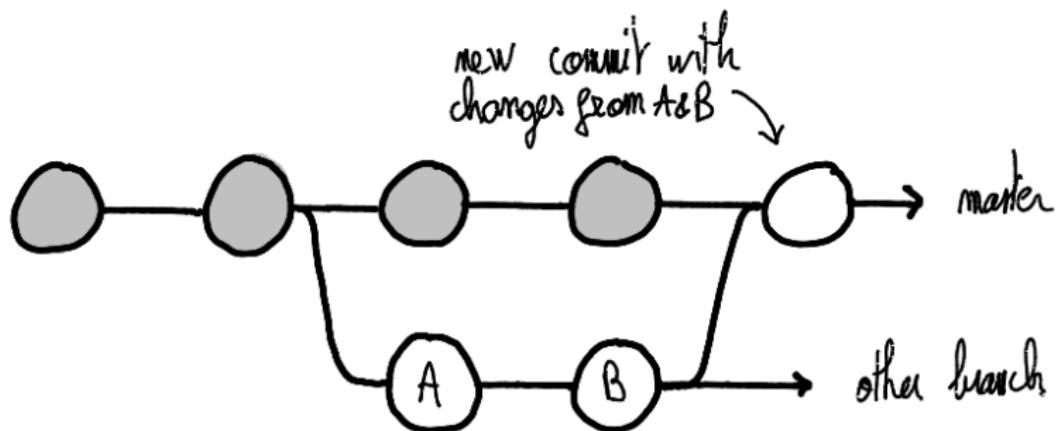
```
$ git stash apply
```

Retrieve the stashed changes

Merging branches

```
$ git merge <other_branch>
```

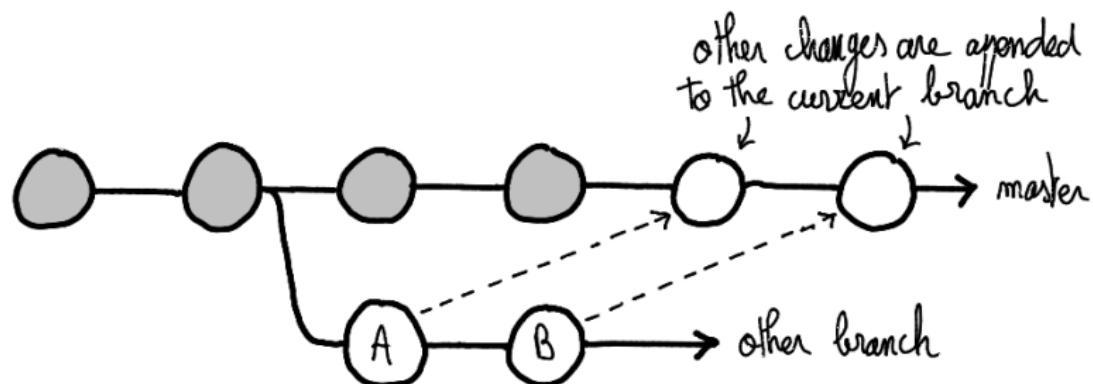
Merge other branch into current branch



Rebasing branches

```
$ git rebase <other_branch>
```

Rebase other branch into current branch



Conflicts

```
$ git merge <other_branch>
Auto-merging my_file.py
CONFLICT (content): Merge conflict in
my_file.py
Automatic merge failed; fix conflicts and
then commit the result.
```

If the same line has been modified in the two branches, we cannot merge automatically. We enter **conflict resolution mode**.



Conflicts

```
if __name__ == "__main__":
<<<<<< HEAD
    print("Bonjour le monde !")
=====
    print("Hello World!")
>>>>> other_branch
```

We need to choose the modification we want to keep

Conflicts

```
$ git add
```

Add the change (mark the conflict as resolved)

```
$ git merge --continue
```

Continue merging process

Trust git and git status to lead you through this process

Confused much?



- ▶ *How often do I commit?*
A commit is a *valid* contribution.
- ▶ *When do I make a new branch?*
When you're gonna break everything, or for horizontal work.
- ▶ *When do I merge branches?*
When the purpose of the branch is accomplished.
- ▶ Best: *branch and merge*.
- ▶ Branch management (and merge vs rebase) is a long and complex discussion.

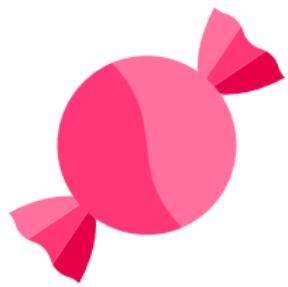
Recap



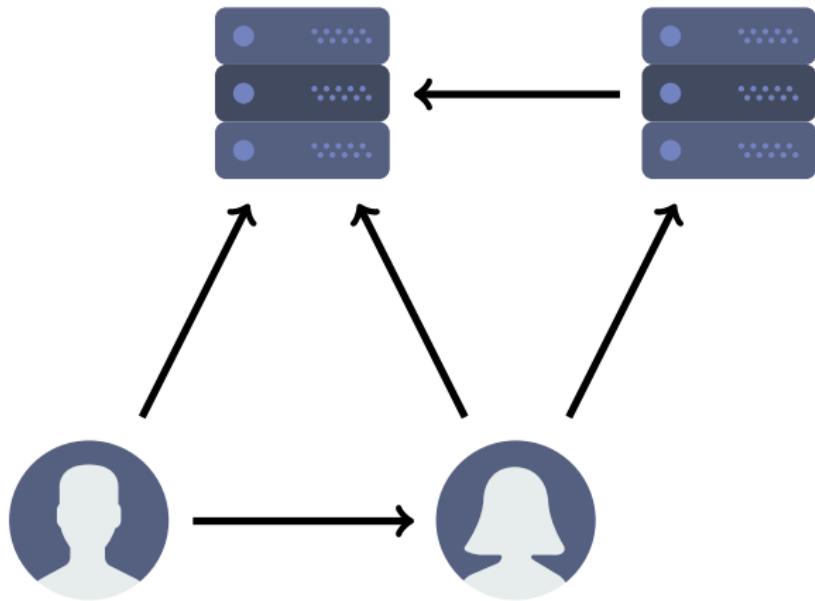
Collaboration

The idea

- ▶ Collaboration is just making interaction between branches on different repository;
- ▶ With syntactic sugar.

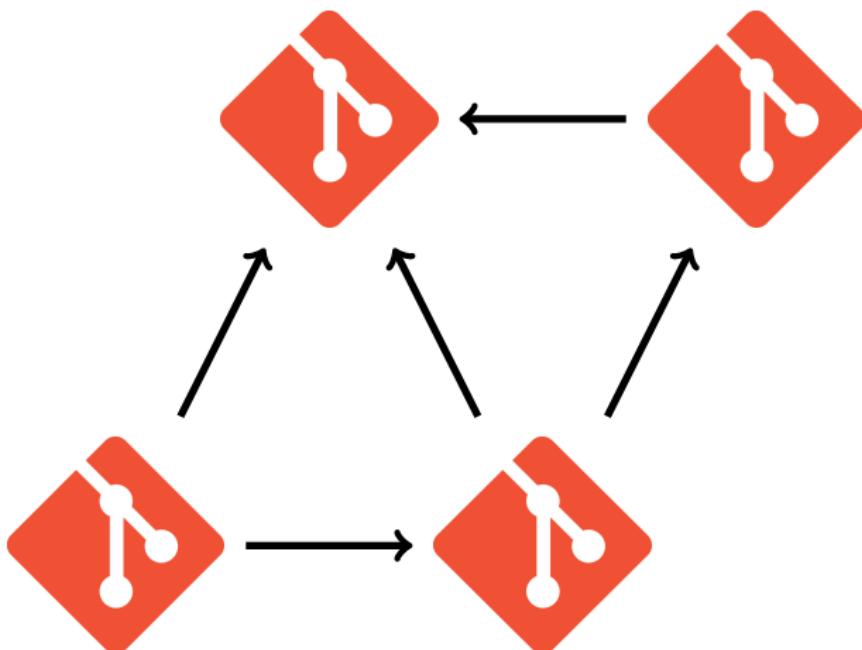


Remotes



In practice you cannot do user to user because we don't know the users computer's address.

Remotes



Remote commands

```
$ git remote -v
```

Show remotes (verbose)

- ▶ Remotes are other git repository¹;
- ▶ You can get (fetch) or send (push) commits to the remote.

¹ Actually aliases for repository URL.

Remote branches

- ▶ More precisely, we can track specific branches;
- ▶ The idea is to have the “*same*” branch locally and on remote.

```
$ git branch -a -vv
```

Show all branches, verbose mode

```
$ git remote show <remote>
```

Information on the remote

¹Actually aliases for repository URL.

Clone

```
$ git clone <url>
```

Show all branches, verbose mode



- ▶ Create a new repository by copying all information;
- ▶ Define the remote;
- ▶ Set the local branches to track the remote ones.
- ▶ **This is what you use in practice**

Getting remote changes

```
$ git fetch
```

Download remote commits
(not merged in your branch yet so harmless command)



```
$ git merge <remote>/<branch>
```

Merge remote commits in current branch



```
$ git pull
```

Download and merge (use this one in practice)

Pushing local changes

```
$ git push
```

Push your local commits to the remote(s).

- ▶ You cannot merge in remote repository from a local repository;
- ▶ Hence, to push commits you need to be up to date

Github

Overview



- ▶ Github is an online site to host git repository;
- ▶ Because your computer is not a dedicated server, people use a remote server to be the immutable reference for the project.

Being social

- ▶ A dramatic amount of open source project are hosted on Github;
- ▶ You can follow people, trends, and project;
- ▶ You can use Github to publish your code;
- ▶ People sometime use it when they just want to put something online (renders pdf, markdown and jupyter-notebooks);
- ▶ Free static website hosting via **Github Pages**



Public code

- ▶ People cannot modify it without your permission;
- ▶ Free private repos for student (**Student pack**)

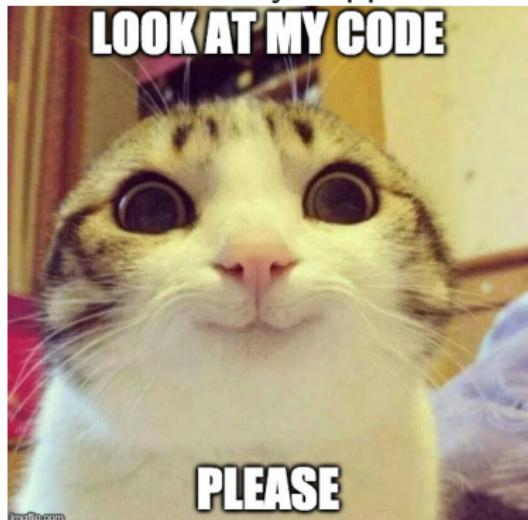
What I thought would happen



YOUR CODE SUCK

memegenerator.net

What really happens



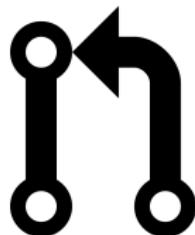
PLEASE

Powerful integrations

- ▶ Large ecosystem of Github App ([Marketplace](#))
From continuous integration to testing, and so on.
- ▶ Team and project management.



The pull request



- ▶ Merge a branch into the master on Github (on the remote repository);
- ▶ Gives an interface to comment, run test, and decide before merging;
- ▶ Best practice to control the contributions

[Advanced] Forks

- ▶ Copy another Github public repository (from anybody) to make a new one;
- ▶ Two fetch remote & one push remote;
- ▶ Possibility to make a pull request from forked to original.



[Advanced] Github authentication



You need to authenticate into Github in order to be able to push (pull).

- ▶ **Easy** set up credentials using [Github Desktop](#)
- ▶ **Moderate** Cache your password for git ([how to](#))
- ▶ **Advanced** Set up SSH authentication ([how to](#))

Other

Other online provider

 Bitbucket



And more

Graphical interfaces



Github Desktop

- ▶ Free
- ▶ Simple



Git Kraken

- ▶ Free and paying options
- ▶ Advanced



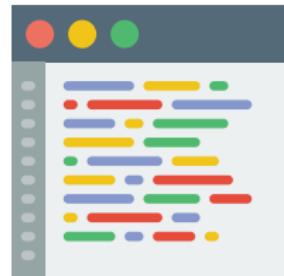
Tower

- ▶ Paying
- ▶ Advanced

More [here](#)

Text editor compatibility

- ▶ All decent editor/ IDE should have git compatibility;
- ▶ Able to pass git commands;
- ▶ Show modified files and lines.



Clean Master

- ▶ Think of the master branch as the last version of your project;
- ▶ It's the version you want people to see / use

[Advanced] Git Hooks



- ▶ Have a action (script) execute whenever you commit / merge;
- ▶ Not specific to Github, builtin in git.

Bad practices

- ▶ **Do not commit** passwords, encryption keys (AWS for instance);
- ▶ **Do not commit** anything that is private (even on private repos);
- ▶ **Do not commit** large files
Commit a very small sample¹ if needed or consider [Git Large File Storage](#);
- ▶ **Do not commit** machine generated files, such as binaries.

¹ Make sure you comply with sharing policies if applicable.

.gitignore file

- ▶ Project file to tell git which files (file type) not to track;
- ▶ Exists for every language (you can add it when you create a repo on Github);
- ▶ E.g. for Python, avoids committing *.pyc, __pycache__, ...
- ▶ All available [here](#)

Git config

```
$ git config
```

- ▶ A number of configurations (commit username)
- ▶ Exists at different level: repo, global

Command line stuff

- ▶ PS1: show current repository information in you prompt
([git-prompt.sh](#))
- ▶ Shell completion: allow to have completion (when hitting tab) for the git sub commands, branches, tracked objects...
(choose you shell at [git completion](#))

Attributions

Icons made by [Smashicons](#), [Good Ware](#), and [Freepik](#) from [Flaticon](#) are licensed by [Creative Commons BY 3.0](#).

Icons made by [Sensible World](#) from [Shareicon](#) are licensed by [Creative Commons BY 3.0](#).