# Introduction to Version Control using Git and Gitlab

Rémi Emonet – 2016-09-01

Université Jean Monnet Laboratoire Hubert Curien

## Version Control with Git: Overview

- Introduction to Version Control and Git
- Git basics
- Schyzophrenic Git
- Collaborating using Git and GitLab (or github)
- Summing it up

## About You

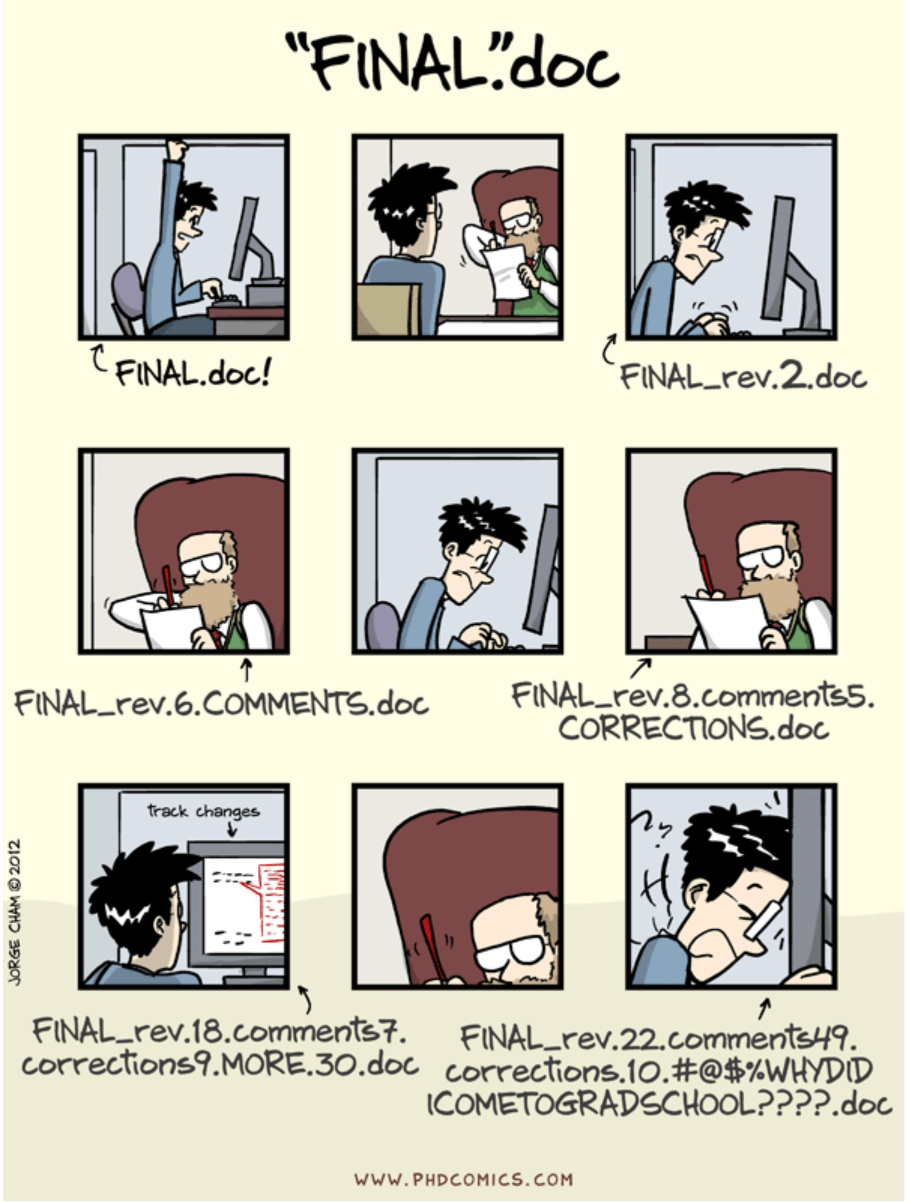
- Who already knows Git?
- Who knows any of these?
    - CVS, Subversion,
    - Mercurial, Baz, GnuArch

## About This Presentation

- Objectives
  - get convinced by version control systems
  - learn practical Git skills
  - learn about GitLab for collaboration
  - hands on with some “code” : a LaTeX paper
- Don't Hesitate
  - to ask questions
  - to interrupt me
  - to ping me after, when trying to practice

## Version Control with Git: Overview

- Introduction to Version Control and Git
- Git basics
- Schyzophrenic Git
- Collaborating using Git and GitLab (or github)
- Summing it up


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

## Why?

@davclark (PhD comics)

# Version Control: What?

- A version control system (VCS)
  - records what you and your collaborators have done
  - allows easy replication across machines
  - allows you to easily see changes
  - allows you to easily experiment new things
- Why dropbox/google drive/… is not sufficient
  - safety of your data
  - ownership of your data
  - semantics of your changes
- Why CVS/Subversion might not be sufficient
  - centralized : a host of the repository
  - working in the train/plane/countryside
  - speed limit

> *SVN-Git migration in progress. 8h to retrieve full SVN history, less than 1min to push full history to Git (same network)!*

## Git

- > *Git (/gɪt/) is a distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. Git was initially designed and developed by Linus Torvalds for Linux kernel development in 2005, and has since become the most widely adopted version control system for software development.*
  >
  > **wikipedia**

- History of Git
  - open source
  - initiated by Linus Torvalds
  - first release: 7 April 2005
  - version 2.9.3: 12 August 2016
  - fast and efficient
  - most used
- Good alternative: mercurial (hg)

## Version Control with Git: Overview

- Introduction to Version Control and Git
- Git basics
- Schyzophrenic Git
- Collaborating using Git and GitLab (or github)
- Summing it up

## Starting with Git

- Initializing your project

```
git init
```

- What's up?

```
git status
```

- Deciding what is relevant

```
git add file1 file2 …
git commit
```

- first: introduce yourself

## Let's try it

```
cp -r base mypaper ; cd mypaper

git init

git status
git add     mypaper.tex  cvpr.sty
git status
git commit
git status

... and more
```

...

## Recap

- Beginning

```
git init
git add ...
git commit [-m ...]
```

- Working

```
git status
git add ...
git commit [-m ...]
```

## Recap 2

- Keep your project clean: ignoring files
  - **.gitignore** file(s)
  - **blabla.*** , **!blabla.my_precious**, * ~**
- What did I just modify?

```
git status
git diff [...]
```

- What happened?

```
git log
```

# Nota Bene (vs CVS, Subversion)

- You have the complete repository
  - have all commits locally
  - commit often, fast and everywhere (train, plane, here)
  - merge with 0-stress
  - warning: commit ≠ backup
- Need to “**git add**” modifications
- Repository == project
  - SVN has a big tree-shaped repository
  - SVN allows to "checkout" any subtree
  - Git works at the repository level
  - you'll have a set of repository
  - commits are at the repository level

## Tools for Git

- GUI
  - Bundled with git: `git gui` and `gitk`
  - Many others (gitg, qgit, GitX, tortoisegit, Netbeans, ...)
  - graphical user interfaces for Git
  - huge list of frontends and tools
- Help for installation
  - for Windows, use "gitbash"
  - details and video helper in the software carpentry instructions

## Customizing Git

- Introducing yourself

```
git config --global user.name  "John Doe"
git config --global user.email john@doe.com
```

- Fancy colors and shortcuts

```
git config --global color.ui true

git config --global alias.st status
git config --global alias.ci commit
```

- Configuration in **`~/.gitconfig`**
- Software Carpentry help about configuring git

## Version Control with Git: Overview

- Introduction to Version Control and Git
- Git basics
- Schyzophrenic Git
- Collaborating using Git and GitLab (or github)
- Summing it up

## About History

- Remember **git log**?
- Each commit is written in stone
  - parent(s) commit
  - modifications
  - sha1sum (e.g. cb6dc3cb1f4f5eb15c1d9b2b25ae741cd73c0554)
- can be diff'ed against

```
git diff cb6dc3...
```

- can be retreived

```
git checkout cb6dc3...
```

# Back to the Future: parallel universes

```
git log
gitk    # or gitg

git checkout 41474a33e098689b...

emacs paper.tex
git commit
gitk
gitk --all

... and more
```

...

# Recap

- Branch
  - a label for a commit
  - automatically follows on new commit (**git commit**)
- Always commit before merging
  - commit is cheap, easy and local
  - you never loose anything when merging
- Use of "sha1" or branch-name (e.g. brrrr)
- Shortcuts (about ^ and ~)

```
cb6dc3, brrrr, HEAD,
HEAD^, HEAD~, HEAD~~, HEAD~2, HEAD~42,
HEAD^2, cb6dc3^42, tagggg
```

## Recap 2

- Moving in the history

```
git checkout sha1-or-branch-name
```

- Creating a new branch at current position

```
git checkout -b new-branch-name
```

- Merging “brrrr” into “master”

```
git checkout master
git merge brrrr
```

## Recap 3

- Non-conflicting **git merge** ⇒ automatic commit
- Conflicting **git merge**
  - (partial merge)
  - non-conflicting changes are automatically added
  - you: solve conflict
  - you: `git add`
  - you: `git commit`
- Exploring history
  - **git log**
  - **gitk [--all]**
  - **git log --graph --decorate --oneline --all --color**

## Best Practices

- commit early and often
- always commit before merge (or pull)
- use meaningful commit messages
  - first line is the short message (50 character)
  - recommended reading on commit messages
- avoid committing
  - binary files that change often (NB: word/excel/... are binary)
  - generated files (that can be regenerated in a reasonable time)
  - temporary files
- keep your git status clean
- don't put git repositories inside git repositories
- more

## Best Practices for Latex

- Reviewing your changes
  - check that it compiles and use git status (e.g., for missing images)
  - use git diff and latexdiff
  - use a spellchecker as you type
- Normalizing your writing (in latex)
  - Goals
    - easier search
    - easier reading
    - better versioning
  - the “one sentence per line” rule
    - never put more than one sentence on one line
    - avoid as much as possible to break a sentence on multiple lines (e.g., don't reflow/justify your tex file)
  - don't put a % alone on a line (if you need a break, probably a paragraph break is good)
  - remove spaces at end of lines
  - remove all "  " (double space, makes search difficult)
  - remove all " ~" (space tilda, typographically wrong)
  - remove all "~ " (tilda space)

# Version Control with Git: Overview

- Introduction to Version Control and Git
- Git basics
- Schyzophrenic Git
- Collaborating using Git and GitLab (or github)
- Summing it up

## What is GitLab (and GitHub)

- GitLab
  - a company providing support and advanced features
  - an open source project (Community Edition)
  - a web application
    - collaboration platform
    - hosting git repositories
    - visualizing repositories
    - managing issues/tickets

> *GitLab offers git repository management, code reviews, issue tracking, activity feeds, wikis.*

## Let's Go

- Create a repository on GitLab
- Push our content
    - link our repository to the remote repository (on GitLab)
    - push the changes to this remote repository
- On another machine
    - clone the repository
    - make changes, commit and push them
- On this machine
    - pull changes: fetch them and then merge

## Recap GitLab (and Git remotes)

- GitLab project == git repository (+ wiki etc)
- The local repository can be linked to some remote one(s)
- When we `git clone`, a remote named `origin` is set up
- When we `git fetch`, we download the remote changes
- … we can then use `git merge` to integrate them
- … or `git pull` that does fetch+merge
- When we `git push`, we upload our history of changes to the remote
- Reminder: always commit before merge or pull

## More GitLab (additions to git)

- Groups
  - groups of users (e.g., PhD student and supervisors)
  - automatic access to the projects of the group
- Forking
  - take a repository on GitLab
  - make a “personal” copy of this repository (still on GitLab)
- Merge requests (pull requests in GitHub)
  - ask for another repo to integrate changes from your fork
- Issues
  - bug
  - questions
  - feature requests
- Wikis
  - set of pages, in markdown
  - (also accessible as a git repository)

## Things to Know at UJM

- SSH access is disabled
  - always use "https://gitlab...." to clone your repository
- To avoid typing your login every time
  - add your user name and @ after https://
  - e.g.,
    `git clone https://`**`er1234h@`**`gitlab.univ-st-etienne.fr/remi.emonet/pyqtidoteach.git`
  - in case you forgot, you can edit **.git/config**
- In case of problems while pushing big sets of commits

  -
    ```
    error: RPC failed; result=22, HTTP code = 411
    fatal: The remote end hung up unexpectedly
    ```
  - just run **git config http.postBuffer 524288000**

## How to Get an Account on GitLab

- For people with a UJM account
  - open a ticket and asking “to get an account on GitLab”
  - wait until you're notified it has been done
  - visit http://gitlab.univ-st-etienne.fr/
  - use your UJM login/pass to connect
  - ignore the email you'll receive about validating your email
- For other people (interns, external collaborators)
  - a person from UJM needs to open a ticket asking for the account
    - providing an email and a name (for each collaborator)
  - upon creation, you receive your credentials
  - to log in, visit https://gitlab.univ-st-etienne.fr/
  - click on the “Standard” tab
  - use the credentials you've been given
- In the near future
  - LDAP users: automatic access
  - other users: unified management across all university services

## Version Control with Git: Overview

- Introduction to Version Control and Git
- Git basics
- Schyzophrenic Git
- Collaborating using Git and GitLab (or github)
- Summing it up

# Key Points

- Version control
  - keep track of what happened
  - store semantic snapshots/versions of your "code"
- Git
  - "distributed" version control: you have a complete repository
  - efficient and widely used
  - one repository per project
- GitLab : a place to share repositories (projects)
  - visualization of the repositories, wiki pages, issue tracker, …
  - groups of users (e.g., PhD student and supervisors)
- Links
  - interactive learning of branching in Git
  - official website
  - graphical user interfaces for Git
  - for Git by a git, ask Linus Torvald
  - Pro Git book (available online)

## Correspondence git ⇄ svn

- git commit ⇄ none
- git commit ; git push ⇄ svn commit
- git fetch ⇄ none
- git fetch ; git merge ⇄ svn update
- git pull == git fetch ; git merge
- NB: you can also use git to collaborate with SVN users, using "git svn"

## Going further

- git remote add
- git tag
- git rebase
- git commit --amend
- git reflog
- git ls-files
- git revert
- git bisect