Introductory course in Git

Git as a tool for distributed version control

Franziska Peter, OpenSystems UB

23 Sept 2021

Universitat de Barcelona

1. Why version control, why git?

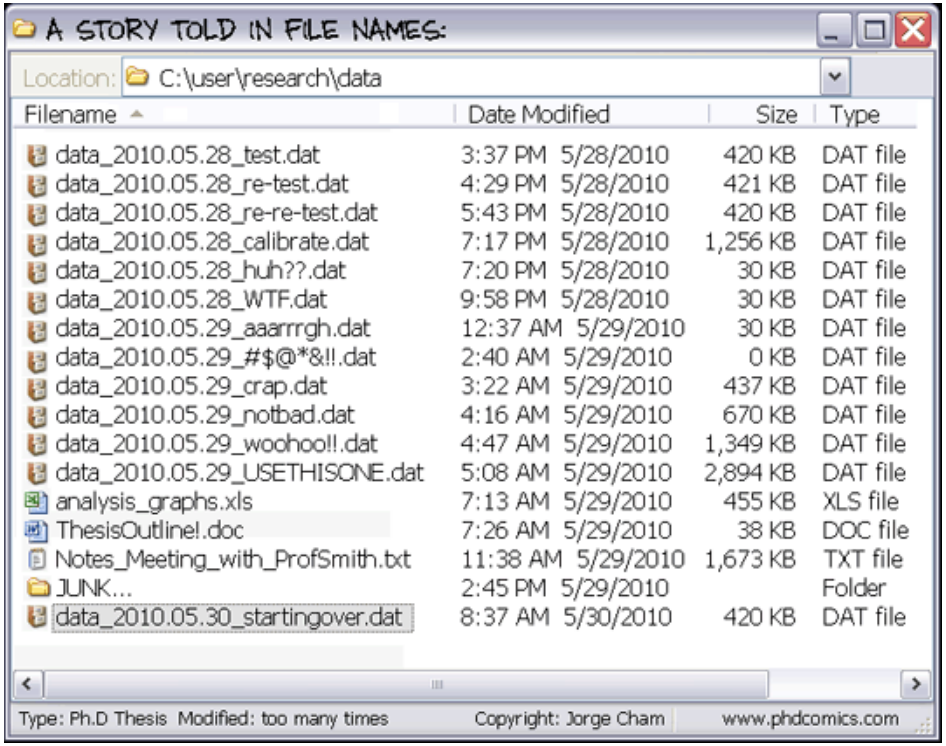
2. Two Basic Principles of Git

3. Getting Started (hands-on)

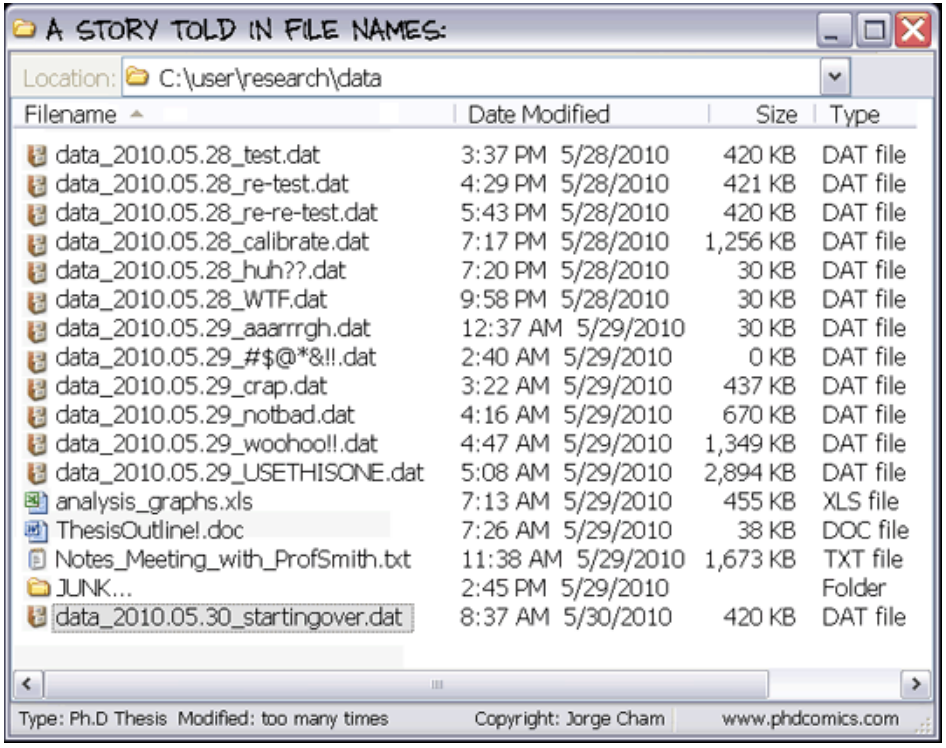
4. By the way...

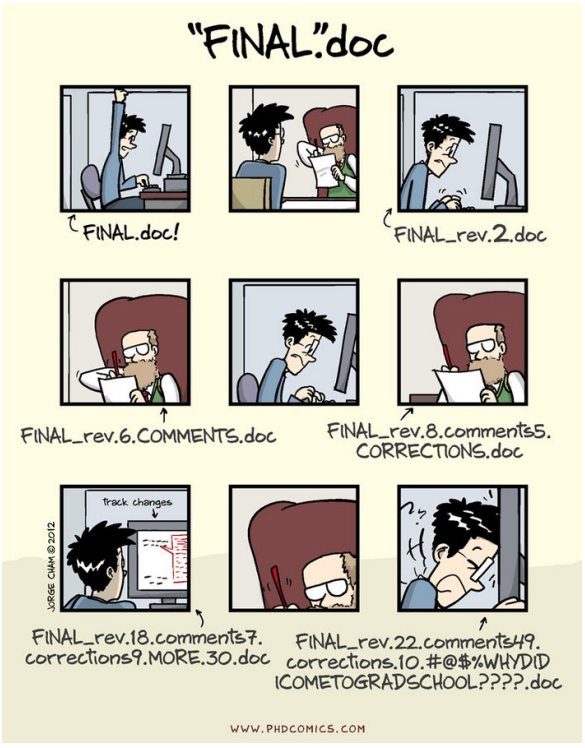
Why version control, why git?

Version control - Motivation



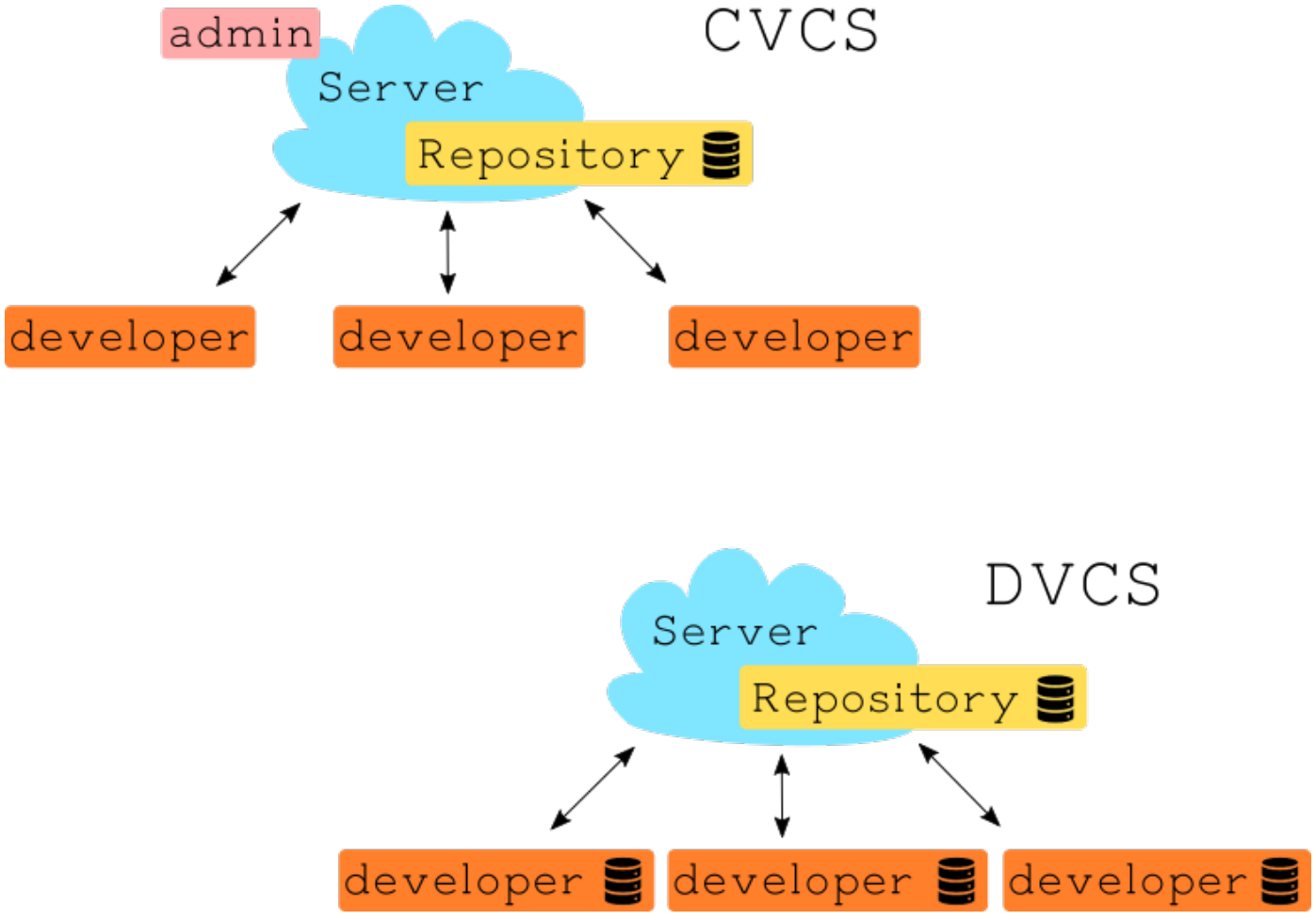
Version control - Motivation





a good VCS should have:
documentation (log book) / rollback/ (external) backup/ simplicity/ allow work in parallel with others/ allow work in parallel on different versions/ speed & efficiency

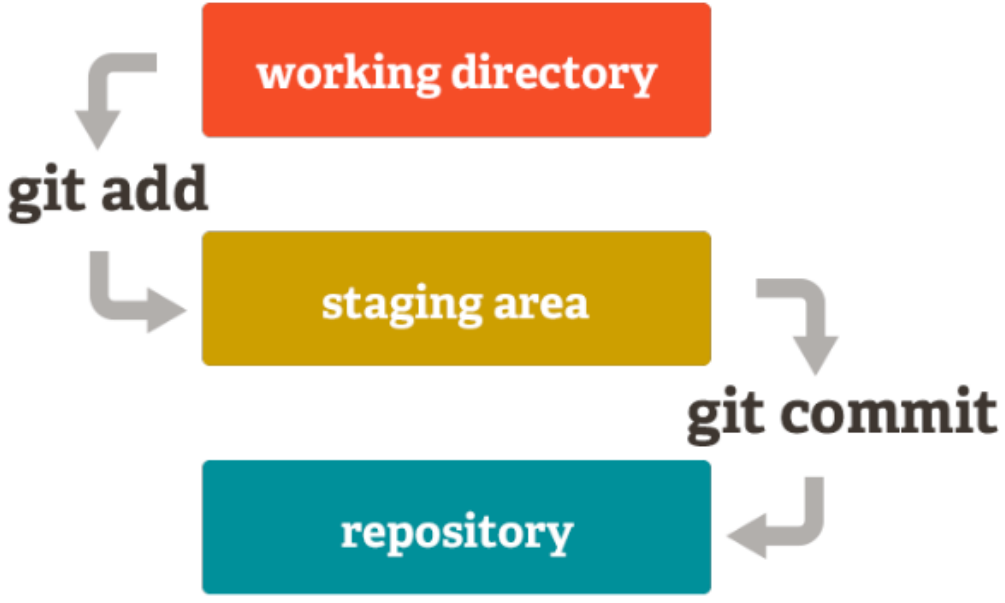
Centralized vs. Distributed Version control



Two Basic Principles of Git

Everyday Mantra: stage and commit (local)

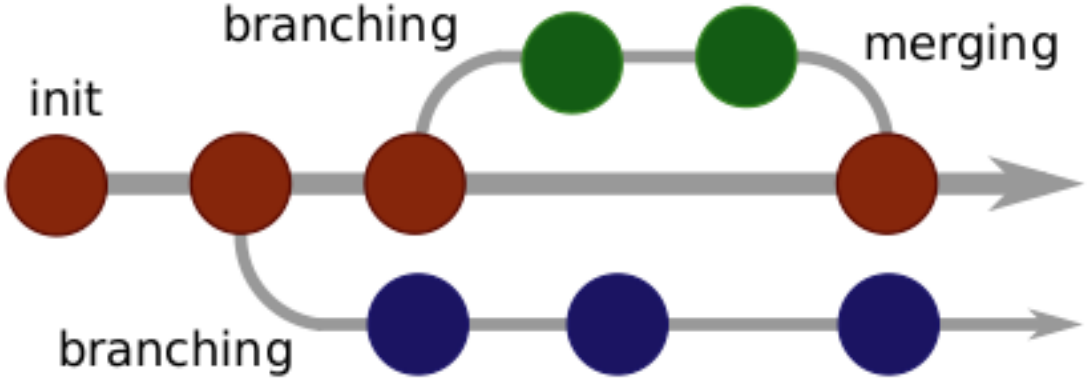
The three states of git: **modified - staged - committed**



```
git add .
git commit -m "implemented calculation of mu"
git push origin master
git pull
```

Working in parallel on different versions: branching

Nonlinear development:
working on several versions **in parallel**, with the possibility of **reuniting** the different resulting versions




```
git checkout -b feature
git checkout master
git merge feature
```

Getting Started (hands-on)

Installation

Recommendation: Use your command line for git.

- Linux/ Mac: first try `git --version`, otherwise install as usual with yum/ apt-get/ zypper/ brew etc.
- Windows: https://git-scm.com/download/win

Create your first own LOCAL git repository

Create a folder for today's course. Create a subfolder called `hello_git`. Run the following commands in your favourite shell via command line from within folder `hello_git`:

- `git init` now your subfolder is a repository

info Linux/ Mac: `ls -a` Windows: `dir a`

info `git status` which branch are you on?, staged/committed?

info `git log` show all commits

- create file with content inside `hello_git/` (e.g. `hello_world.py`), e.g. with vim or nano

Create your first own LOCAL git repository

Create a folder for today's course. Create a subfolder called `hello_git`. Run the following commands in your favourite shell via command line from within folder `hello_git`:

- `git init` now your subfolder is a repository

info Linux/ Mac: `ls -a` Windows: `dir a`

info `git status` which branch are you on?, staged/committed?

info `git log` show all commits

- create file with content inside `hello_git/` (e.g. `hello_world.py`), e.g. with vim or nano
- `git add .`
- `git commit -m "added file hello_world.py"`

Create your first own LOCAL git repository

Create a folder for today's course. Create a subfolder called `hello_git`. Run the following commands in your favourite shell via command line from within folder `hello_git`:

- `git init` now your subfolder is a repository

info Linux/ Mac: `ls -a` Windows: `dir a`

info `git status` which branch are you on?, staged/committed?

info `git log` show all commits

- create file with content inside `hello_git/` (e.g. `hello_world.py`), e.g. with vim or nano
- `git add .`
- `git commit -m "added file hello_world.py"`
- change file, e.g. `hello_world.py`, then run `git add .` , `git diff HEAD` , and `git commit -m "removed typo"`

Publish (?) your repository on GitHub

- Sign Up on https://github.com/
  - create a **token** on github.com under Settings>Developer settings>Personal access tokens

Publish (?) your repository on GitHub

- Sign Up on https://github.com/
  - create a **token** on github.com under Settings>Developer settings>Personal access tokens
- "upload" your local repository to github:

  Instead of `git remote add origin https://github.com/Chaotique/learn-git-a-bit.git`
  Use git remote add origin https://**<token>**@github.com/Chaotique/learn-git-a-bit.git
  `git push -u origin master`

Publish (?) your repository on GitHub

- Sign Up on https://github.com/
  - create a **token** on github.com under Settings>Developer settings>Personal access tokens
- "upload" your local repository to github:

  Instead of `git remote add origin https://github.com/Chaotique/learn-git-a-bit.git`
  Use git remote add origin https://**<token>**@github.com/Chaotique/learn-git-a-bit.git
  `git push -u origin master`
- alternative: create a repo on github and clone from there:

  `git clone https://github.com/Chaotique/test-repo.git`
  And then git remote **set-url** origin https://**<token>**@github.com/Chaotique/test-repo.git

Working with remotes

info `git remote -v` list remotes with url

- `git remote add foo <url>` add a remote with alias "foo"
- `git remote set-url foo <url>` change url of remote foo
- `git push origin master` send updates to origin (origin is the default alias for the default remote)
- `git clone <url>` download complete copy of a repo (with full history) from a platform and make it a remote
- `git pull` = `git fetch` + `git merge` get updates

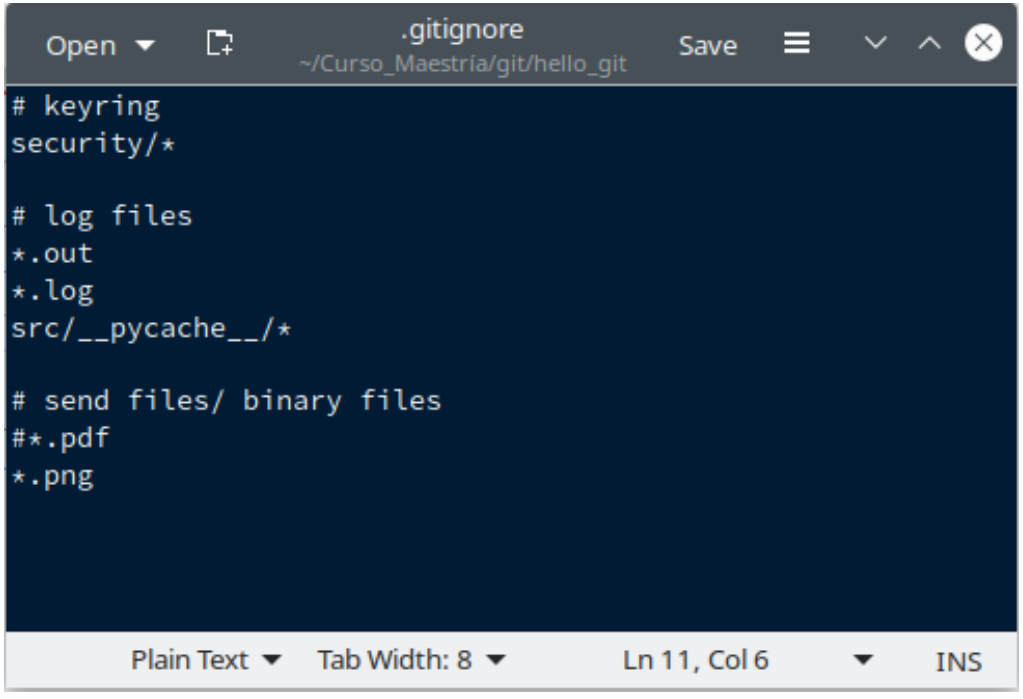
Working in parallel: branching

- `git branch developer` create a branch called "developer"
- `git checkout developer` switch to branch "developer"
- `git checkout -b developer` create a branch called "developer" and switch to it
- `git checkout master` and `git merge developer` switch to master and merge developer branch into master branch
- if necessary, resolve merge conflicts
- `git branch -d developer` delete branch "developer" after successful merge

By the way…

By the way... the first thing to do after `git init`

Adding a `.gitignore` file (using glob patterns) to your `hello_git/` folder (or any of its subfolders) makes git ignore the specified files and folders.



By the way... know the difference(s)

You can compare any version on any branch with any version in any state. Most typical examples:

- `git diff`: working directory vs. staged
- `git diff --staged`: staged changes vs. last commit (HEAD)
- `git diff HEAD`: HEAD vs. working directory

For any other versions: you'll find out when you need it :P
Also: Try `git log -p -3` to see log & changes from the last three commits.

By the way... how to undo things? DANGER

DANGER

Many steps can be undone on git. But be careful not to delete anything important!

- `git restore --staged example.py` to undo staging (> Git version 2.23.0), `git restore example.py` throws away your modifications, so careful!
- `git commit --amend` to add sth to the commit (use only locally!!)
- `git checkout -- example.py` to throw away all modifications you did since the last commit - DANGER!

By the way... how to delete or move stuff?

You can't simply remove a file, git will notice (tracked files), use `git rm example.py` instead.

Same story about moving files. Either use

```
mv example.py src
git add src/example.py
git rm example.py
```

or instead simply do

```
git mv example.py src
```

What's more?

- **tagging**: give special commits a special name
- **issues**: mark bugs and later refer to them in the fixing commit
- **stashing**: hold modifications (staged or unstaged, but not yet committed) in the air to make other operations first, then pop them back in
- **forks and pull requests**: branch from other peoples projects/repositories and later ask the to accept your proposed changes
- **rebase**: do clean up work on your forest of commits, or run messing up everything completely (experts command)

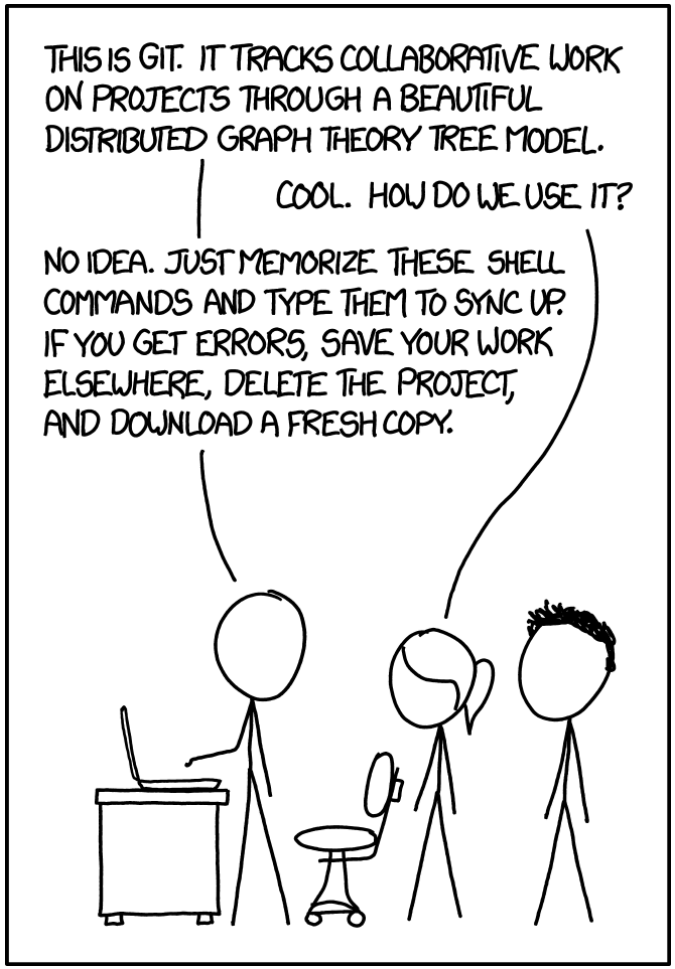
Further reading, Tutorials, Cheat Sheets

Scott Chacon, Ben Straub (2021)
Pro Git
https://git-scm.com/book/en/v2

Tutorials:
https://try.github.io
https://www.atlassian.com/git/tutorials

Cheat sheets: (several languages)
https://www.git-tower.com/blog/git-cheat-sheet/
https://training.github.com/

Have a lot of fun and joy with git!



Gràcies per l'atenció!

¡Gracias por la atención!

Thanks for your attention.