The Hitchhiker's guide to not (severely) screw up

Lecture 2 : Git and GitHub

Tommaso Ronconi

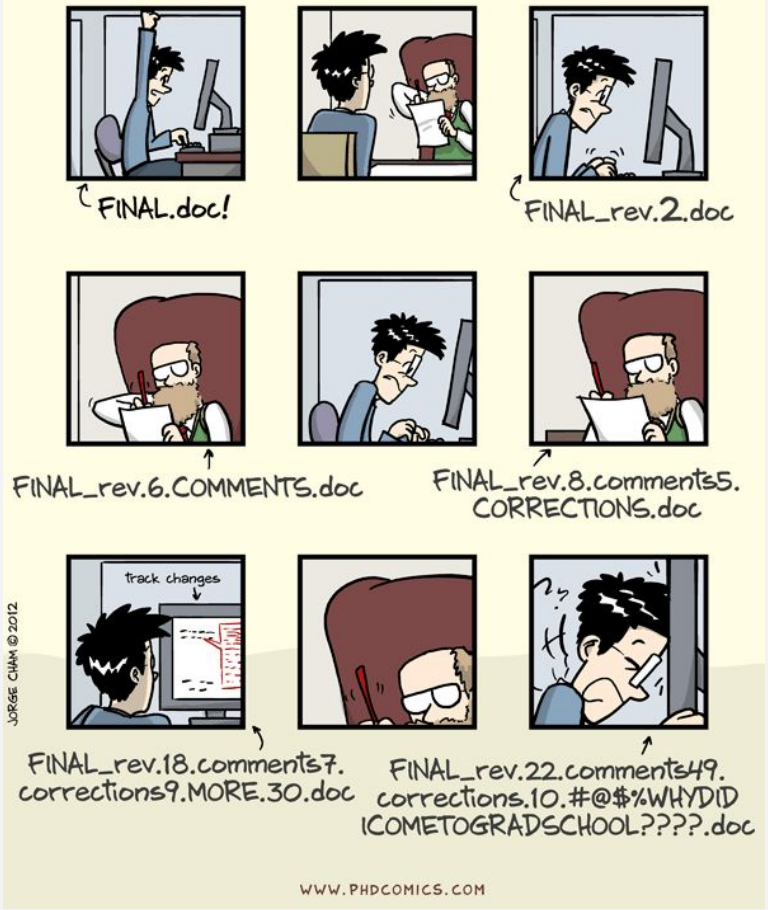
Outline for today

Version-control and Collaborative Development

- ➔ How to secure your progress and keep track of your work with **Git**
- ➔ Back-up and co-op with online git repositories: the hosting service **GitHub**

$ git clone git@github.com:TR/knowledge.git

"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

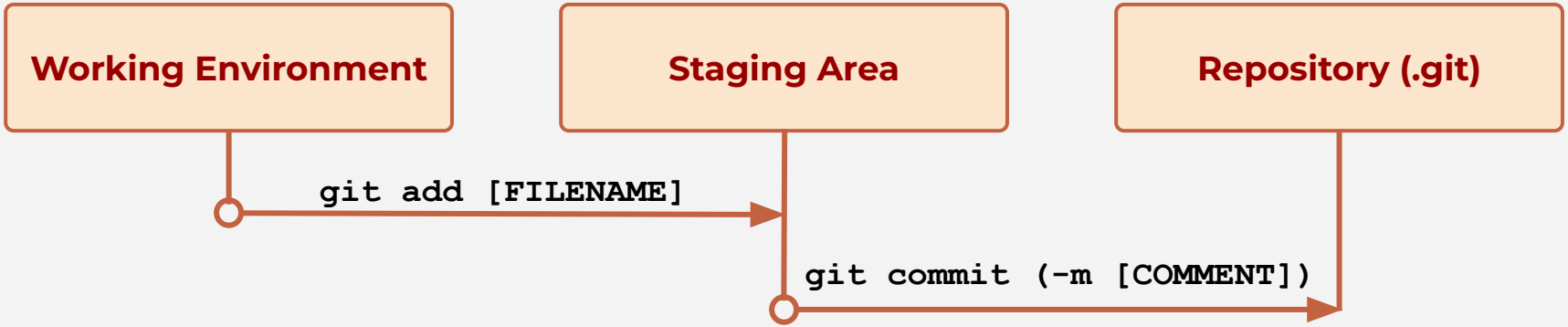
GIT - the stupid content tracker

Git: noun [C], UK informal. *A person [...] who is stupid or unpleasant.*
(Cambridge Dictionary)

- Default on GNU Linux system, easy to install and use on Unix
- Basically **THE(!!!) COLLABORATIVE VERSION CONTROL SYSTEM** standard.
- Easy to setup and run:
  - Change configs:
    ```
    $ git config --list
    $ git config --global user.name "Tommaso"
    $ git config --global user.email "tronconi@sissa.it"
    ```
  - make a directory:
    ```
    $ mkdir new_project && cd new_project
    ```
  - tell git you want to track the content: **`$ git init`** and that's it.

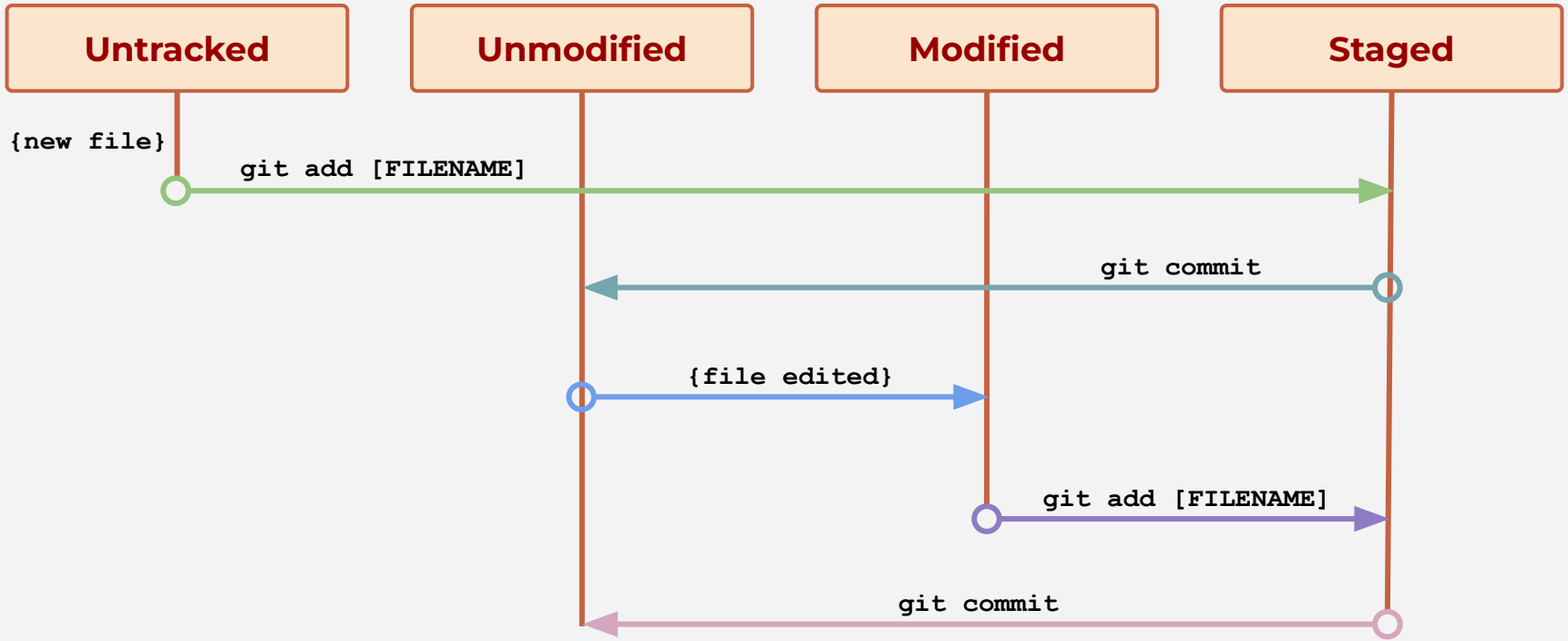
Exercise:
We can also make things easier, remember the PS1?
check out this file: `/usr/share/git/git-prompt.sh`
we want it to be executed every time we open the terminal.

GIT - different “areas”



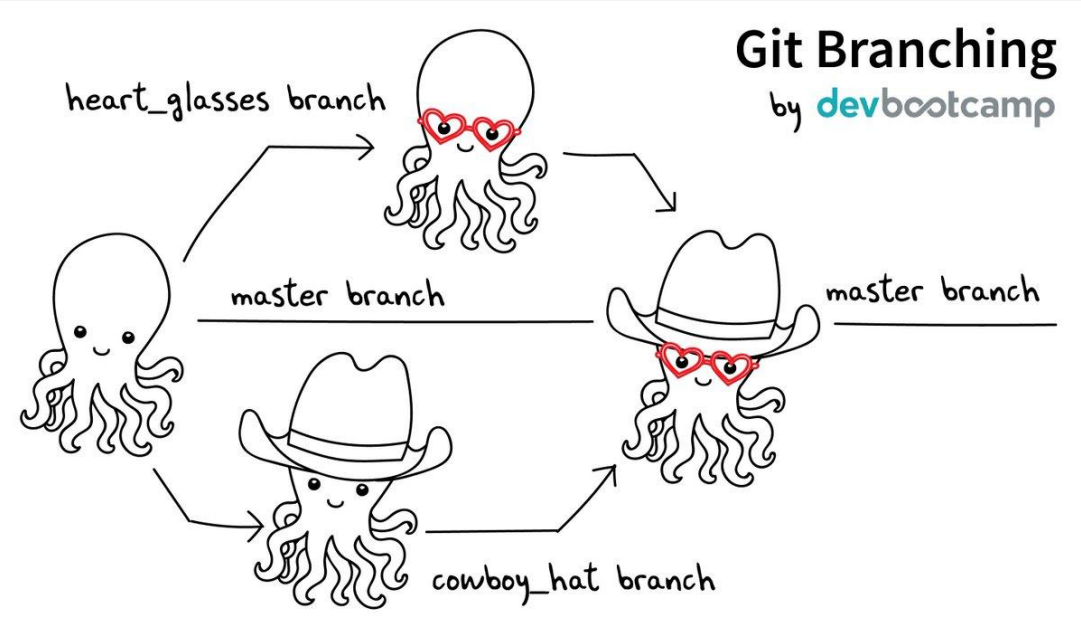
- `.git/` makes the directory a git repository: it’s a directory containing all the changes informations
  The position in your working environment containing this hidden directory is called
  root directory of the repository (not to be confused with the root of your filesystem, i.e. **/**)
- `.gitignore` define a set of files that will not be part of the repository (even though inside the dir)
  - you can also add a global gitignore:
    `$ git config --global core.excludesfile “/path/to/global_gitignore”`
- `HEAD` is a pointer to a position in the repository history: *you will always have just one head.*
  Your working environment will always be where your head is pointing to.

GIT - File Status



GIT - Branching

So, you want to develop a **cool new feature**? Better use **git-branch.**



- Create a new branch:
  `$ git branch cowboy_hat`
  Move in the newly created branch:
  `$ git checkout cowboy_hat`
- Do the above in just one command:
  `$ git checkout -b cowboy_hat`
- Merge a branch into another
  `$ git checkout master`
  `$ git merge heart_glasses`
- Delete a branch
  `$ git checkout master`
  `$ git branch -d heart_glasses`
- List all available branches:
  `$ git branch -a`

Create and move into new branch: `git checkout -b new_b` » **Add new features** » **Move back to original branch:** `git checkout master` » **Merge new branch:** `git merge new_b` » **Delete new branch:** `git branch -d new_b`

GitHub - An online space for git repositories

An on-line service for **hosting** git repositories → **Collaborative development**

My account: https://github.com/TommasoRonconi

Some alternatives exist:

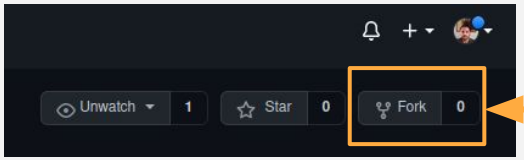
- GitLab (somebody prefers this, I do not judge)
- BitBucket (it is a bit old-fashioned now)

Actions possible with remote repositories

- Clone a remote repository: `$ git clone git@github.com/Author/repo_name.git`
- Connect local repository to a remote repository:
  `$ git remote add remote_name git@github.com/Author/repo_name.git`
- Fetch content from a known remote: `$ git fetch remote_source [branch_name]`
- Merge content of remote branch: `$ git merge remote_source/branch_name`
- Send changes to remote repo: `$ git push remote_source [branch_name]`

Hands on - fork+clone a repository

1. Go to this link: https://github.com/TommasoRonconi/crash_python_course



2. Click on the “fork” button (top right) and follow the instructions

3. Back to the terminal: go to your home and

```
$ git clone git@github.com/YourName/crash_python_course.git
```

Congrats! You have cloned your first **remote repository** into a **local repository**!!

Show remote repos connected to the local: `$ git remote -v`
Show branches of the current repo (including remotes): `$ git branch -a`

4. Connect also to my remote version of the repository:

```
$ git remote add tommaso git@github.com/TommasoRonconi/crash_python_course.git
```

Hands on - a dummy collaborative project

The instructions for the exercise are at this link:

`https://www.github.com/YourName/crash_python_course/exercise2_git`

[of course a local copy is also present on your system but go at the link]

To set-up secure-shell (aka SSH):
at this link (from checking blabla)

Git cheat sheet

The very basic “what’s going on?”-kit

man git[-command]	Man pages of git [or of the given git-command (e.g. **man git-status**)]
git status	See the status of the repository (what is new, what is staged, what has been moved/deleted)
git branch [-a]	List all available branches [include also the remote branches]
git remote -v	List all the remotes this repo is connected to and your rights on the remote (fetch/push)
git diff [--staged]	what is the difference between the working environment [staging area] and the repository?

Modify file status

git add file/pattern	Stage modifications in the provided `file` or in everything matching `pattern`
git commit [-m “comment”]	Commit staged changes to repository
git rm/mv file/pattern	Remove or move file or pattern (the change will appear as staged)
git reset HEAD staged_file[s]	Un-stage modifications in `staged_file[s]`
git reset HEAD [SHA1 code]	Go back to a previous commit

Juggle with branches

git branch new_branch [SHA1]	Create a new branch starting from HEAD [`SHA1` code] position
git branch -d branch_name	Delete `branch_name`
git checkout [-b] branch_name	[Create and] move to `branch_name`
git merge branch1 [branch2]	Merge `branch1` into HEAD pos. [into `branch2`]
git rebase branch_name	Permanently merge `branch_name` history with history in HEAD pos.

Inspect history

git reflog	Show history of where HEAD had pointed to
git log [--oneline|--graph|..]	Show history of all modifications of repository

Deal with remotes

git remote add name address	Add remote @`address` and name it `name`
git fetch remote_name [branch_name]	Fetch recent mods from remote [‘s branch]
git push remote_name branch_name	Push your latest commits to `remote_name`

And that’s all folks! (for today)