Version Control Systems

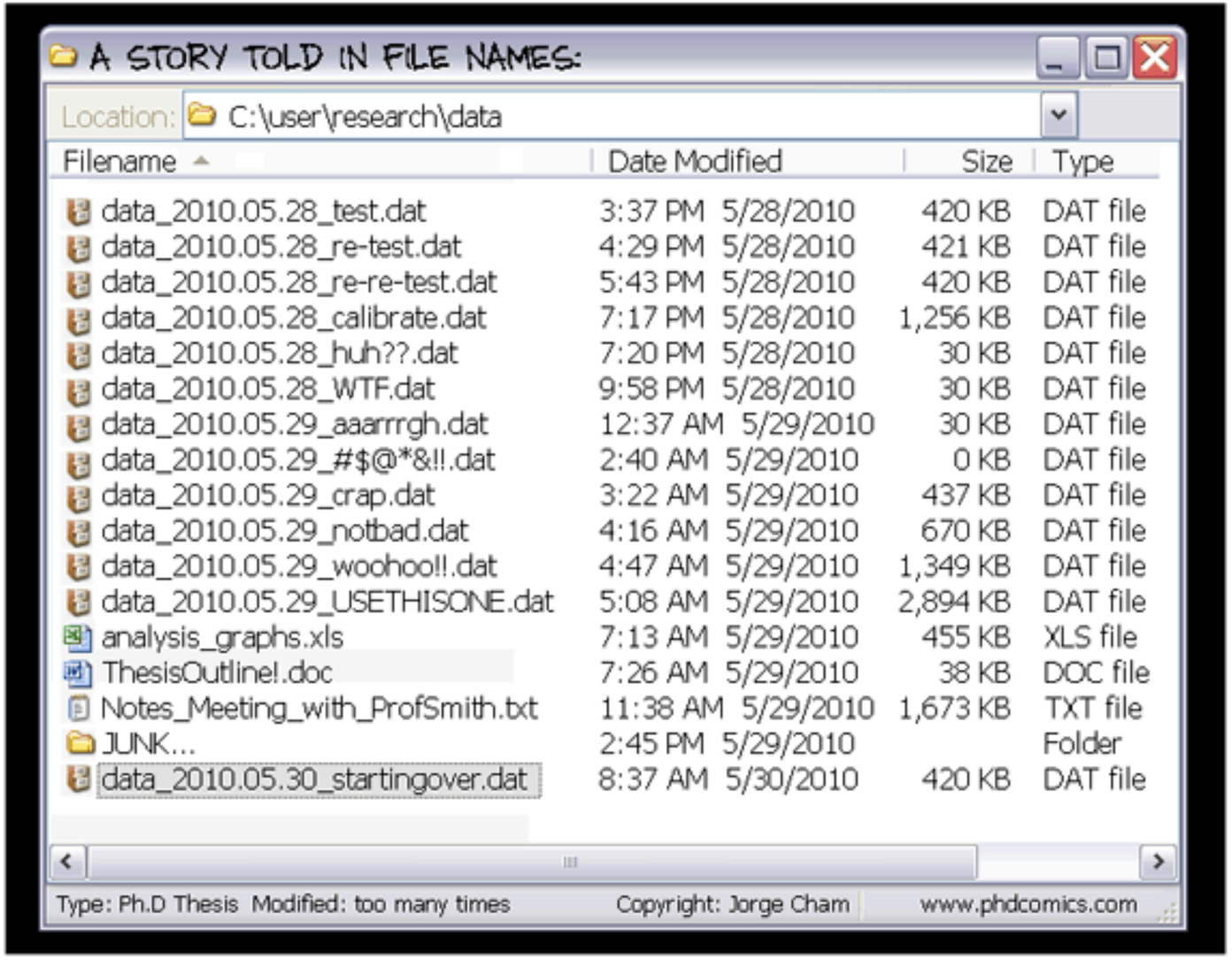
Dorota Jarecka (she/her)

MIT, Cambridge, MA, USA; ReproNim

Version Control Systems (VCS): what do they do?

- track changes in your files
    - can be files with code (scripts, source files), configuration files, images, documents, data
- keep all the versions of the files
    - you can access any previous version at any later point in time

…wait, I got this!



experimenting with suffixes

- probably the most well known improvised version control
- not sure if should be called a system

even if you keep all versions of your file, would you be able:

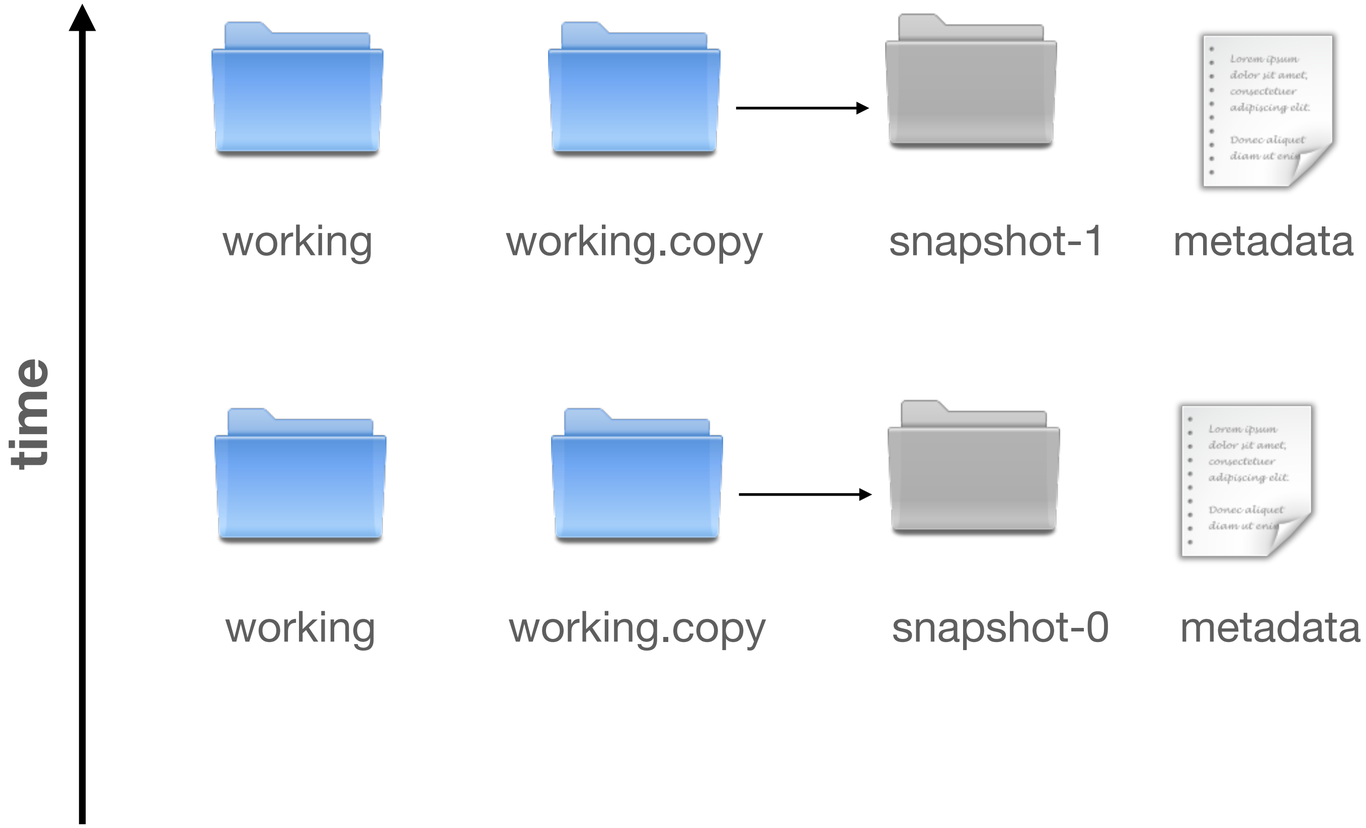
- tell the difference between *re-test* and *re-re-test* in 2 months or 2 weeks?
- are you sure you didn't use *woohoo!!* version instead *USETHISONE* in your last publication?
- are you able to pass your knowledge about the suffixes to your colleagues?

Version Control Systems (VCS): what do they do?

- track changes in your files
    - it can be files with code (scripts, source files), configuration files, images, documents, data
- keep all the versions of the files
    - you can access any previous version at any later point in time
- allow for easy collaboration
- improve efficiency (not right away, but pretty quickly)

Version Control System (VCS): concept

Creating snapshots of entire working directory



Git
Introduction and working with files

What is Git?

- the most popular VCS
- free and open source
- distributed system
  - no need for the central server
  - easier for collaboration and offline work
- handle everything from small to very large projects
- high performance
  - doesn’t copy entire content of the files with each snapshot
- according to manual page*: “the stupid content tracker”*
  - doesn’t think for you, only does what is asked to do

Git command line and initial configuration

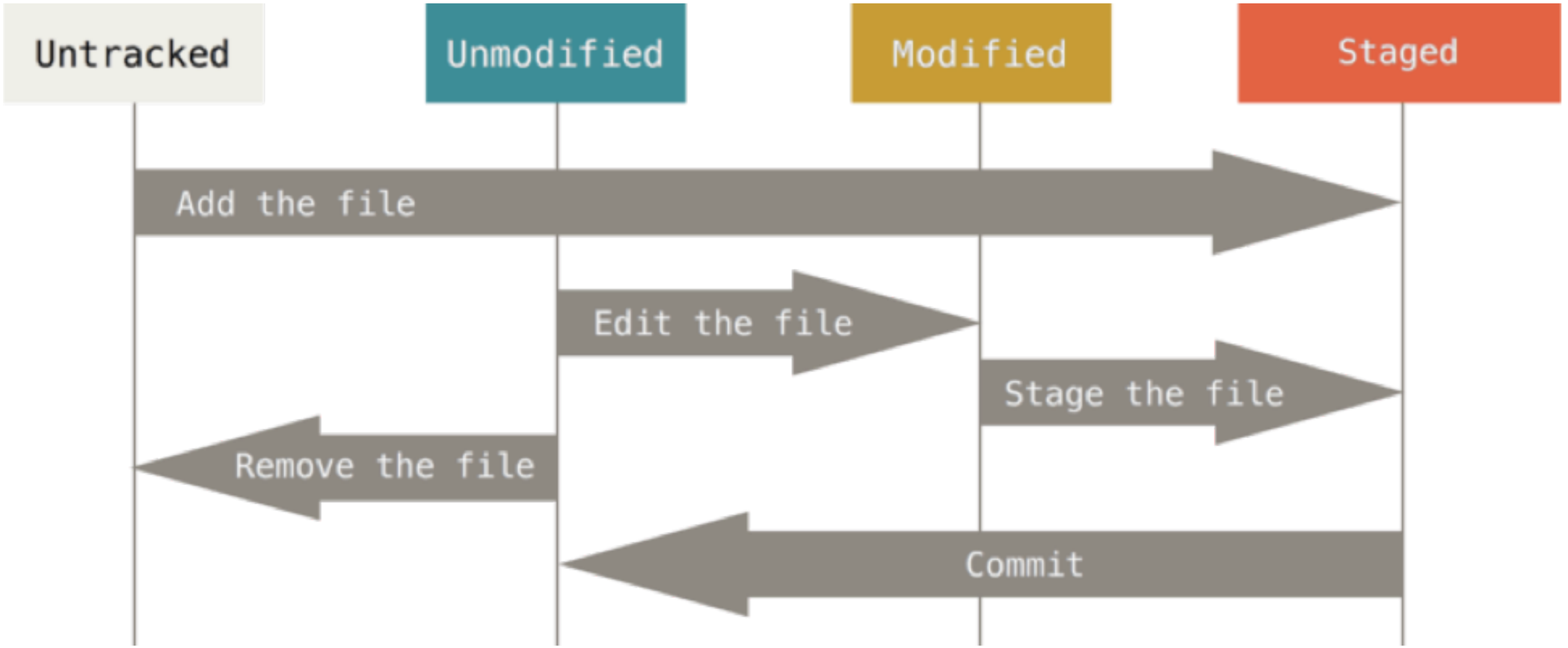
- checking local installation
  - *git --help*
  - *man git*

- basic initial configuration (only once)
  - *git config --global user.name "Your Name Comes Here"*
  - *git config --global user.email you@yourdomain.com*

- checking your settings
  - *git config --list*

Working with Git (a.k.a. the stupid content tracker)?

The life of files in the git repository

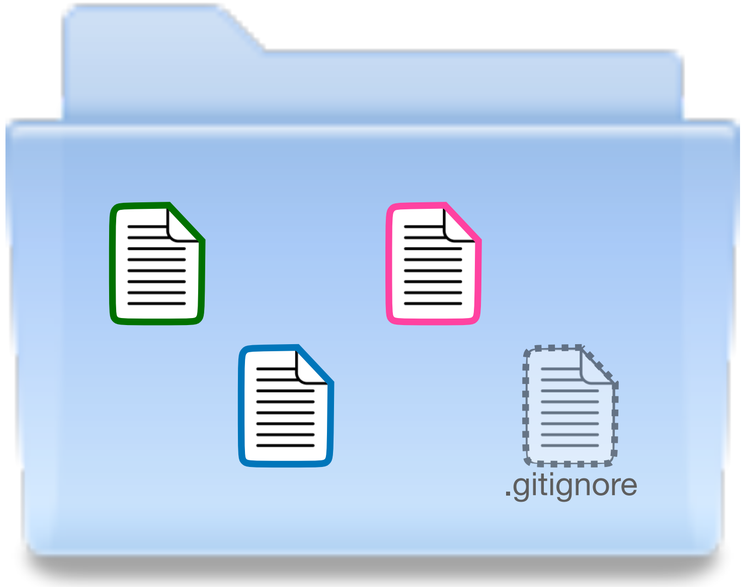


https://git-scm.com/book/en/v2

Working with Git (a.k.a. the stupid content tracker)?

Creating new git repository

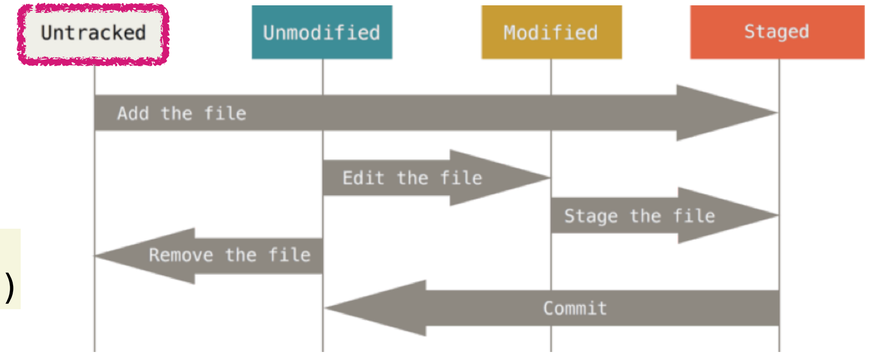
- create a git repository from the existing working directory
  - *cd my_project*
  - *git init*



my_project

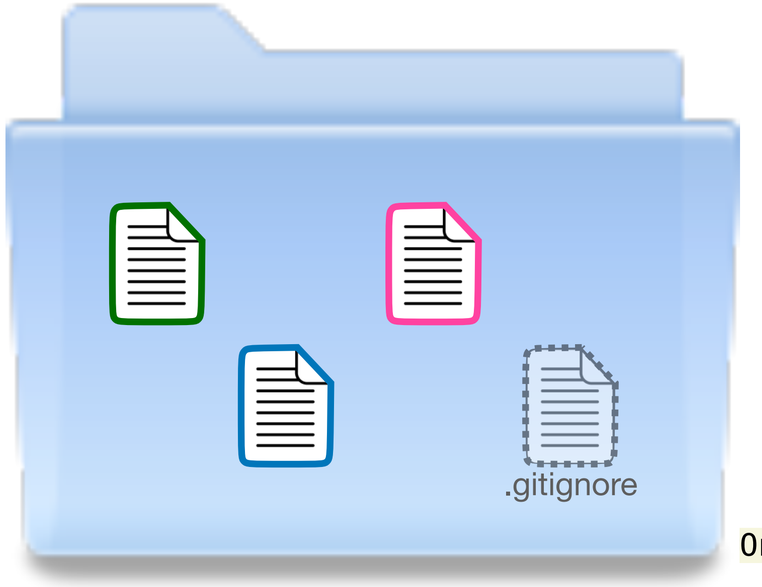
- checking the status
  - *git status*

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will
be committed)

  blue
  green
  pink

nothing added to commit but untracked
files present (use "git add" to track)
```



Working with Git (a.k.a. the stupid content tracker)?

Adding new files to the repository



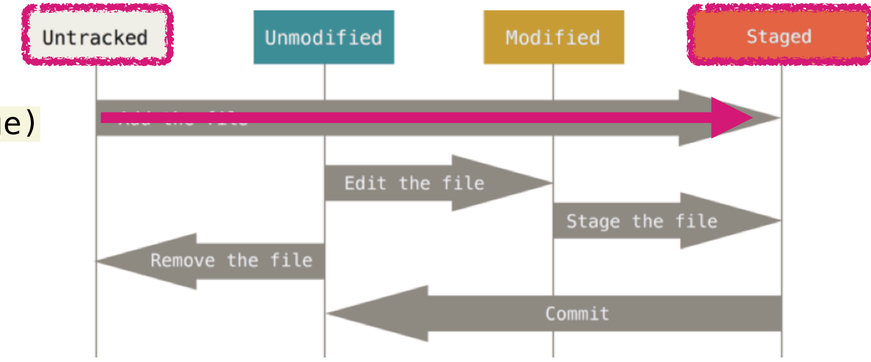
my_project

- adding file you want to track (git will not assume anything)
  - *git add green blue*

- file that shouldn't be tracked, e.g. cache directory (or pink files), can be added to *.gitignore*
  (the file would have to be added as well: *git add .gitignore)*

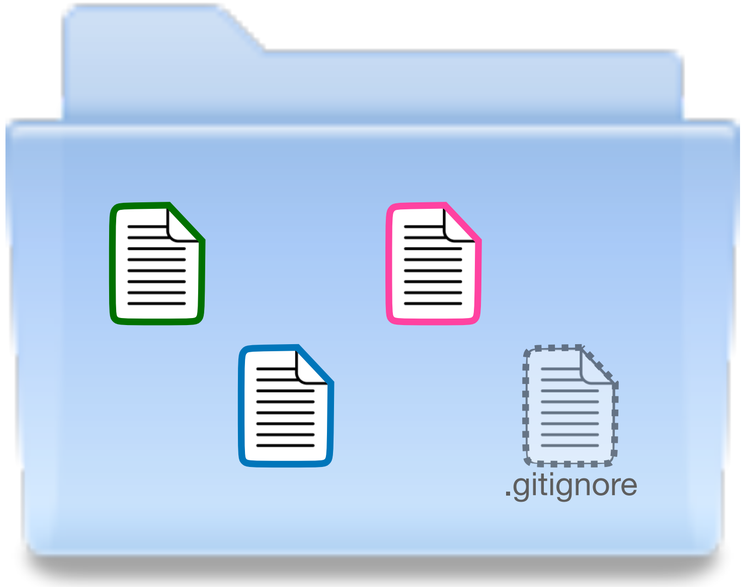
- checking the status: *git status*

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:   .gitignore
  new file:   blue
  new file:   green
```



Working with Git (a.k.a. the stupid content tracker)?

Creating the first snapshot of the repository

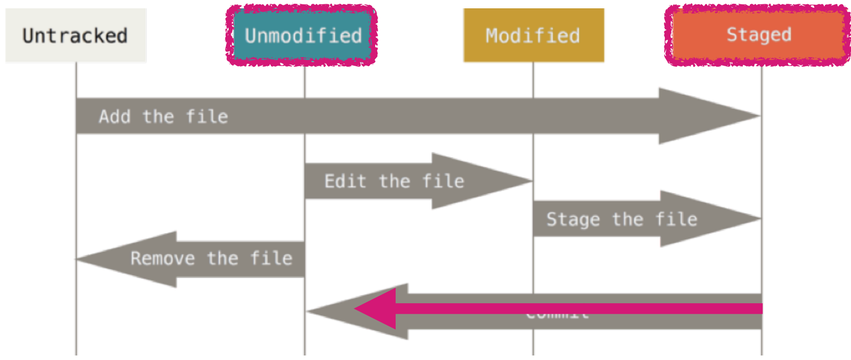


my_project

snapshot 0

- saving (creating a snapshot) of the initial version of the files
  - *git commit -m “initial version"*
- checking the status: *git status*

```
On branch master
nothing to commit, working tree clean
```



Working with Git (a.k.a. the stupid content tracker)?

Saving changes in a tracked file

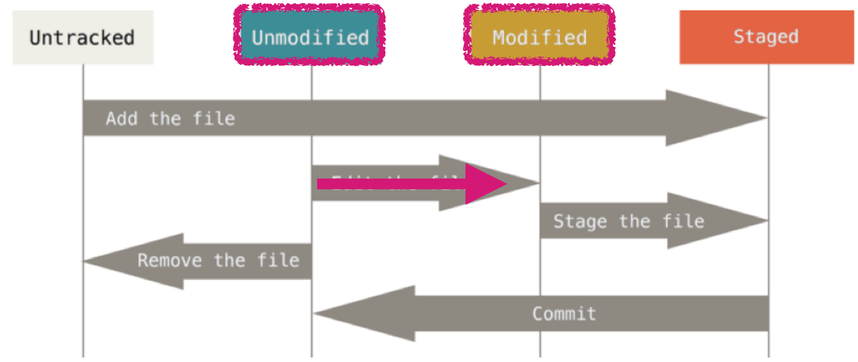
my_project

- editing one of the tracked files (the green one)
  - checking the status: *git status*

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   green

no changes added to commit (use "git add" and/or "git commit -a")
```

  - we can also check the difference
    - *git diff*

```
diff --git a/green b/green
index 11b3125..29cfdc3 100644
--- a/green
+++ b/green
@@ -1 +1,2 @@
-Initial content of the green file
\ No newline at end of file
+Initial content of the green file
+bla
\ No newline at end of file
```



Working with Git (a.k.a. the stupid content tracker)?

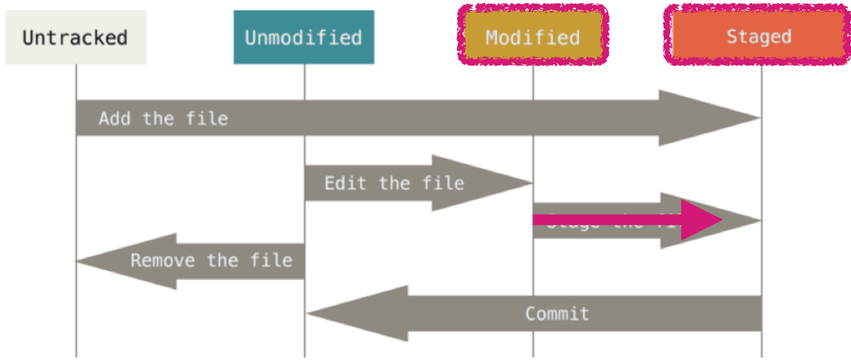
Saving changes in a tracked file

my_project

- git doesn't know when is the right time to move the file to the staged state, you have to explicitly add the file to the staged:
  - *git add green*

- checking the status: *git status*

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  modified:   green
```



Working with Git (a.k.a. the stupid content tracker)?

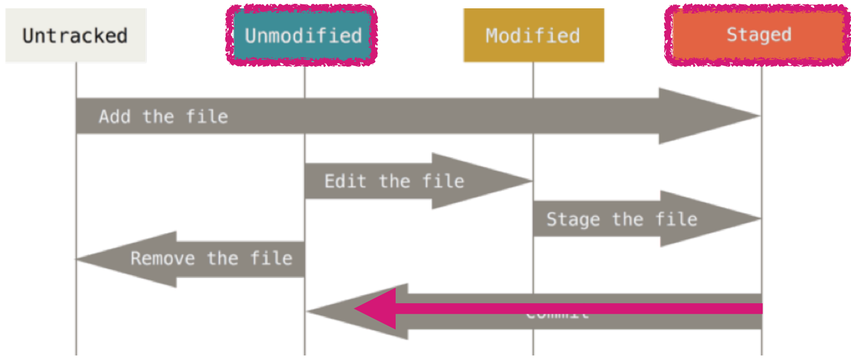
Saving changes in a tracked file

my_project

snapshot 1

- a new snapshot is not created until the changes are committed:
  - *git commit -m “my new thought"*
- checking the status: *git status*

```
On branch master
nothing to commit, working tree clean
```



Working with Git (a.k.a. the stupid content tracker)?

Checking the history

my_project

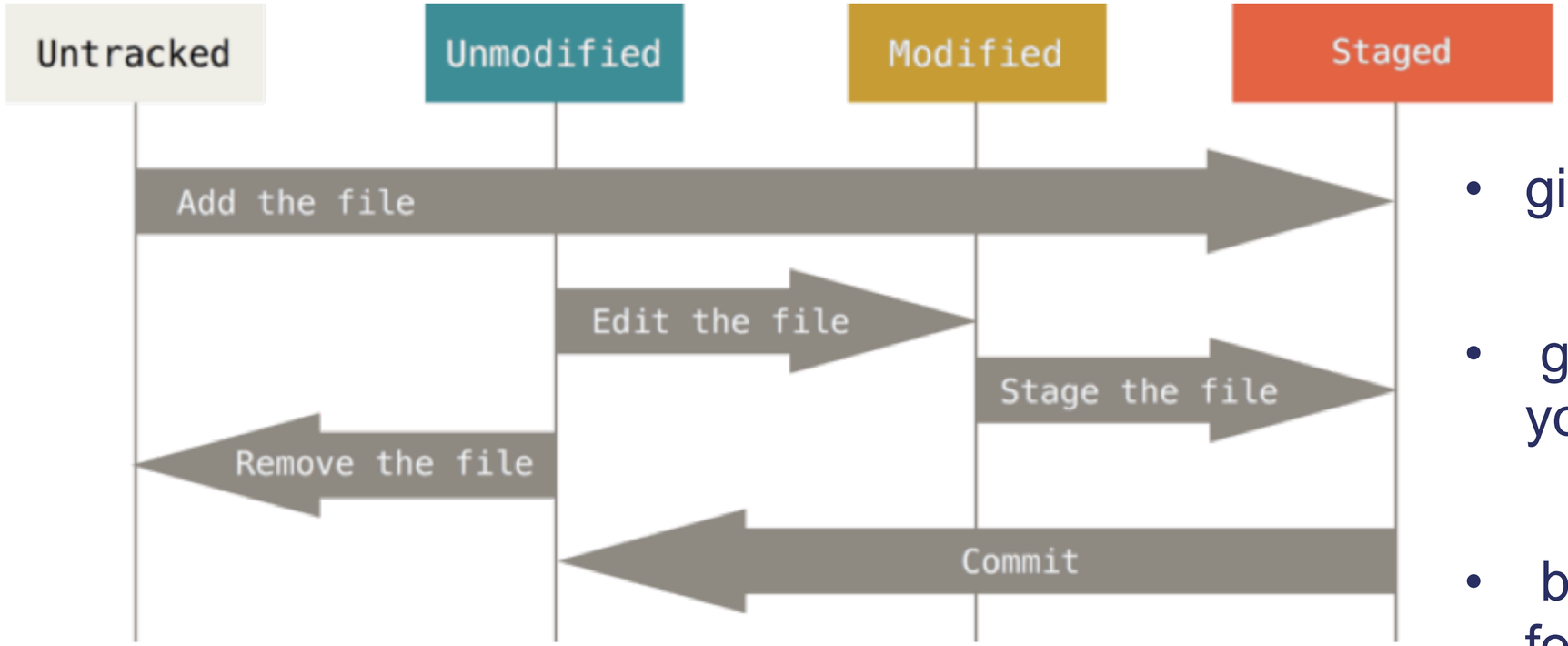
snapshot 1

- checking the history
  - *git log*

```
commit 95700c943d5152c1adc1a88e69b0d2e5cc9a0dbd (HEAD -> master)
Author: Dorota Jarecka <d***@gmail.com>
Date:   Mon Sep 14 12:39:06 2020 -0400

    my new thought

commit 446d5cc85c4a5ca526f3036b00c8f61910972722
Author: Dorota Jarecka <d***@gmail.com>
Date:   Mon Sep 14 12:38:27 2020 -0400

    initial version
```

Working with Git (a.k.a. the stupid content tracker)?

The life of files in the git repository



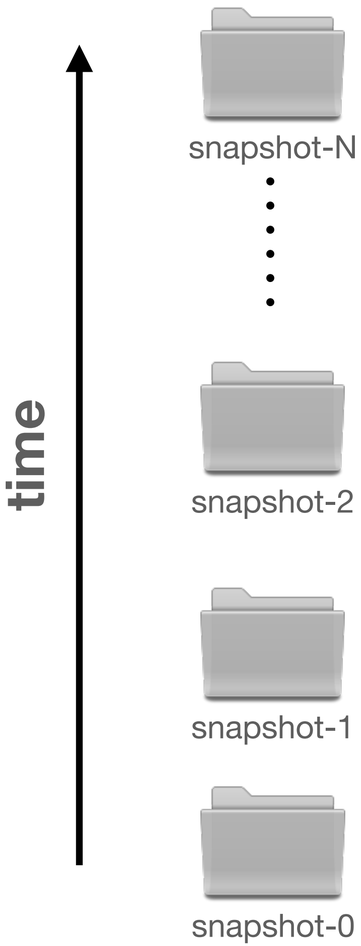
- git doesn't think for you
- git doesn’t guess what you want to do
- but git motivates (sometimes forces) you to think and organize your work

https://git-scm.com/book/en/v2

Git
Working with branches

Git: conceptual model of snapshots

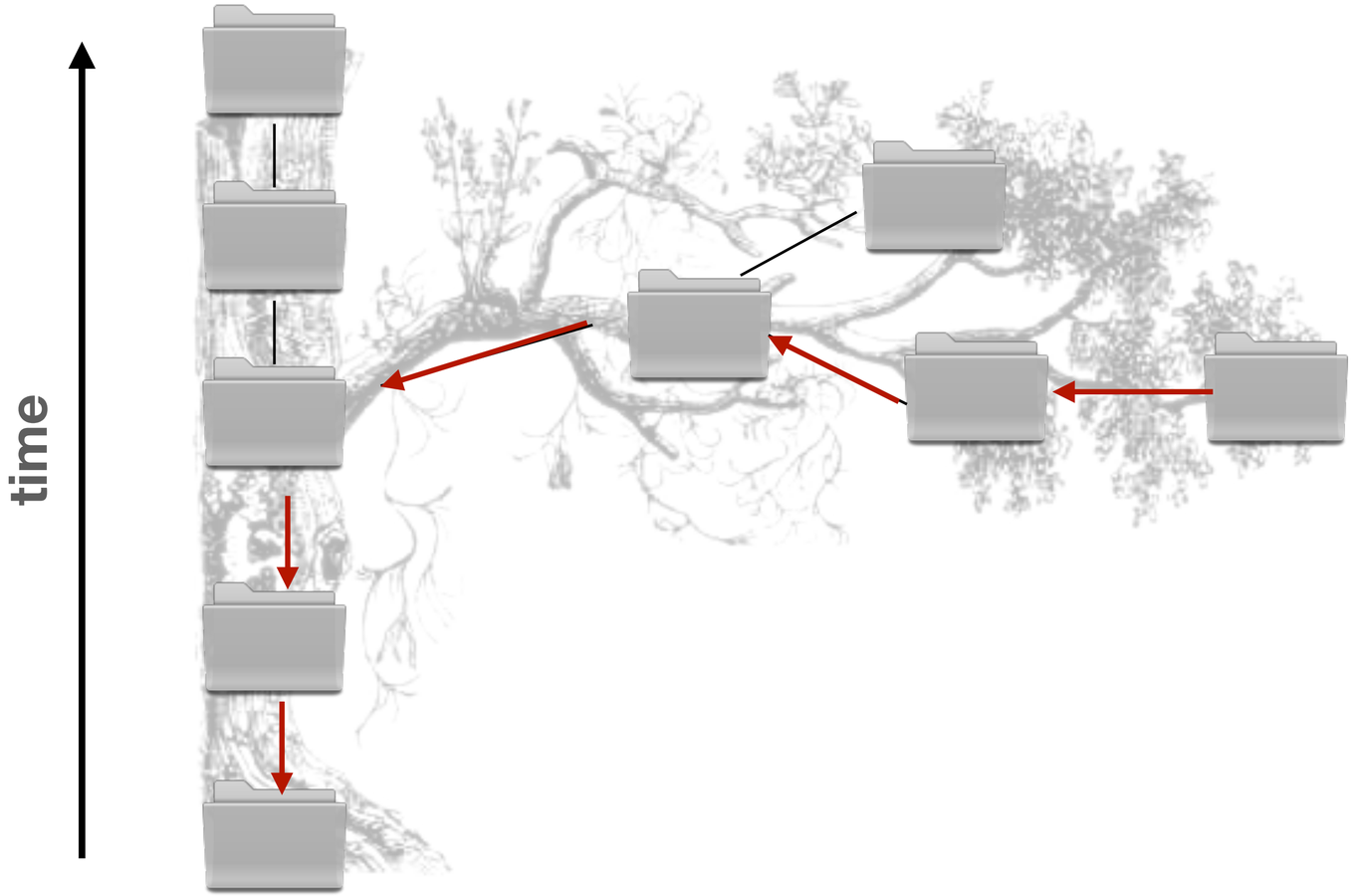
Linear model: every snapshot has one “child” snapshot



- is simple
- but often unrealistic:
  - assumes that you are working on one thing at a time

Git: conceptual model of snapshots

Non-linear model: a tree model



- looking at your code history as a tree
- you have multiple latest snapshots, one for each branch
- each snapshot can point to the previous snapshot (i.e., “parent” snapshot)
- pointers enable to easily trace the history of any given snapshot all the way back to the root

Git: conceptual model of branches

Why do you want to create new branches?

- keep the main branch clean
- experiment with new feature safely on new branches
- work on multiple features at the same time

Working with branches on Git?

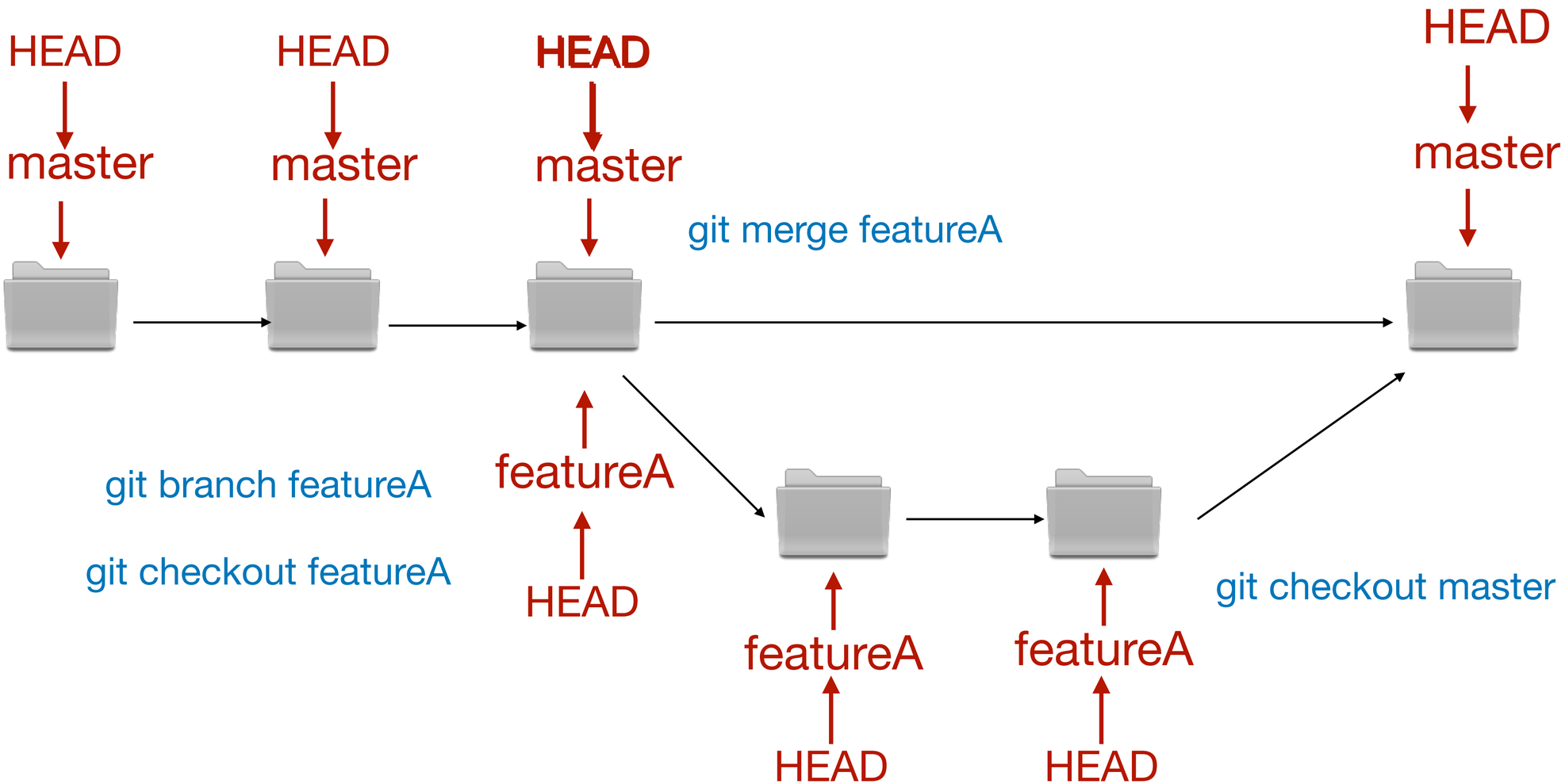
- *master* (or *main*) is the default branch that is created when project is initialized
- you can create as many branches as you want
    - *git branch featureA*
- you can jump between branches
    - *git checkout featureA*
- you can always check all of your branches:
    - *git branch*

```
  featureA
* master
  tests
```

- three branches
- the HEAD is pointing on master

Working with Git branches on Git?

- branch - movable pointer to one of the commits
- HEAD - special pointer that always points to the current branch



Git
GitHub and working with remotes

What is GitHub?

- website provides you with public (or private) storage for your Git repositories

- allows for easy sharing and collaboration

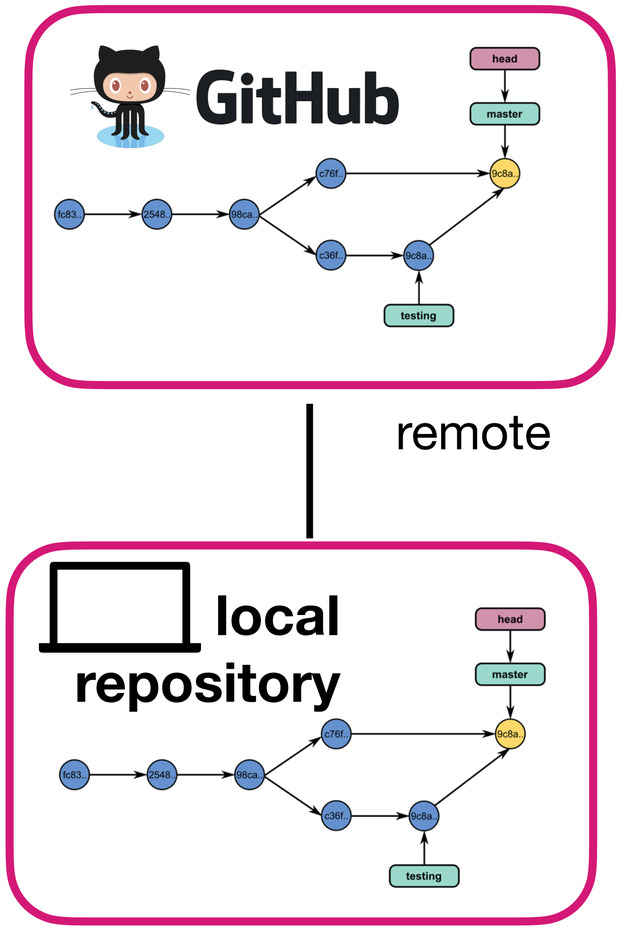
- allows third-party websites to interact with your repositories to provide testing, publishing, etc.

- basic services are free of charge

- similar platforms: Bitbucket, GitLab (open-source, can be installed locally)

Git - working with remotes

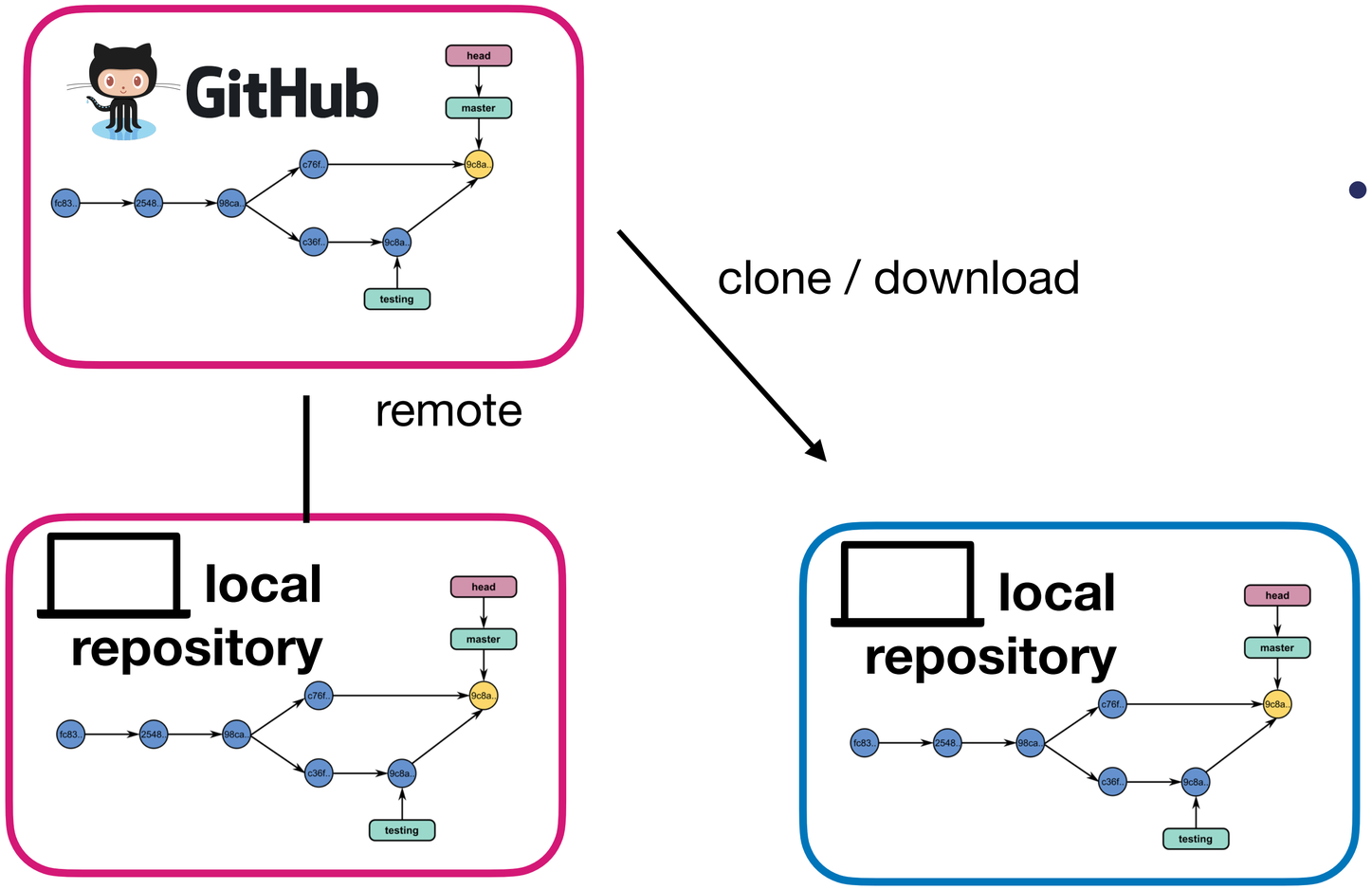
Single developer-user



- GitHub can be set as a remote
    - remote repositories - versions of your project that are hosted on the internet or other networks
- changes can be pushed to the remote or pulled from the remote

Git - working with remotes

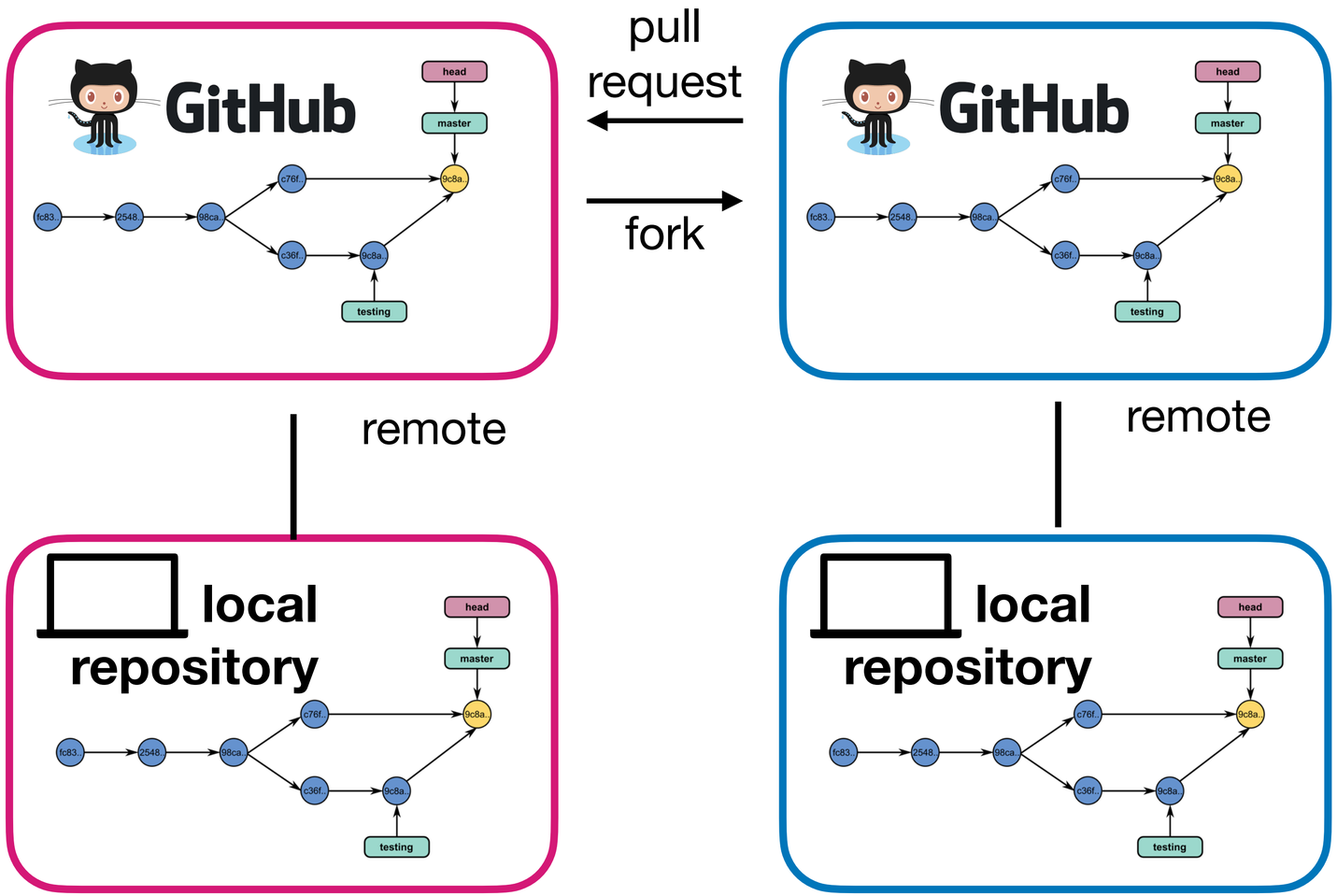
An additional user



- if the GitHub repository is public, anyone can clone (download) the repository
- users can modify the code, but can’t push to the GitHub repository

Git - working with remotes

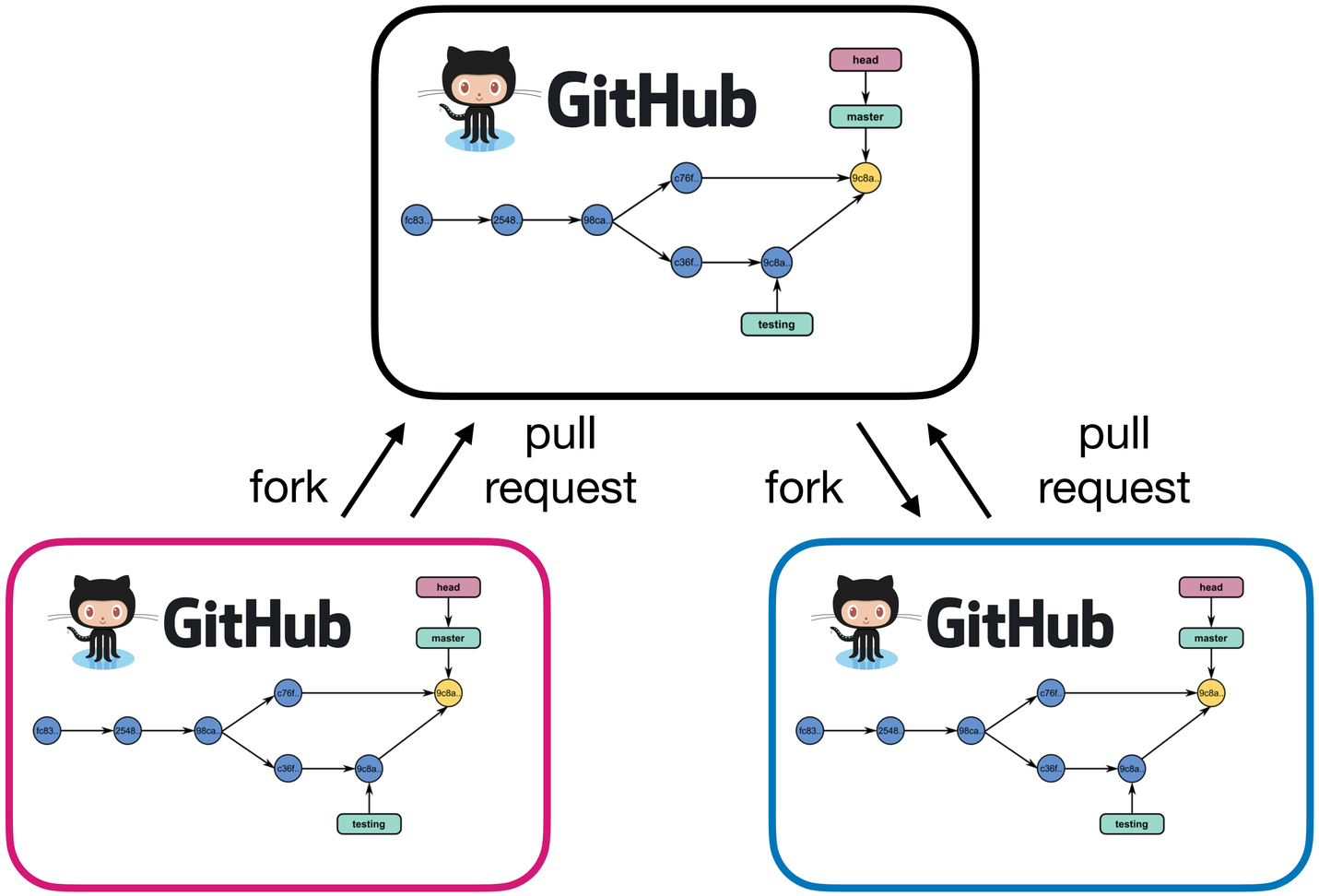
An additional collaborator (a small team)



- collaborators often create a new fork on GitHub and have their owns GitHub repositories
- all local changes can be pushed to the remote
- a pull request with the changes can be created from the forked repository

Git - working with remotes

A bigger team



- for bigger number of collaborators it's often better to set the main public repository (often under a specific organization)
- everyone can create a new fork and pull request
- some or all collaborators can approve changes suggested in pull requests

Git and friends

- Git is great, but...
    - It’s not design to handle big files (you don’t want to upload your big files to GitHub and keep a copy of them in every single location)
- git annex:
    - allows managing files with git, without checking the file contents into git (creates a symbolic links instead)
- DataLad
    - relies on Git and git-annex
    - more intuitive command-line interface and Python API
    - developed by members of the Neuroimaging community (and ReporNim)

You will hear more about DataLad during this course!