Basics of Git and GitHub

Diego Garrido Martín

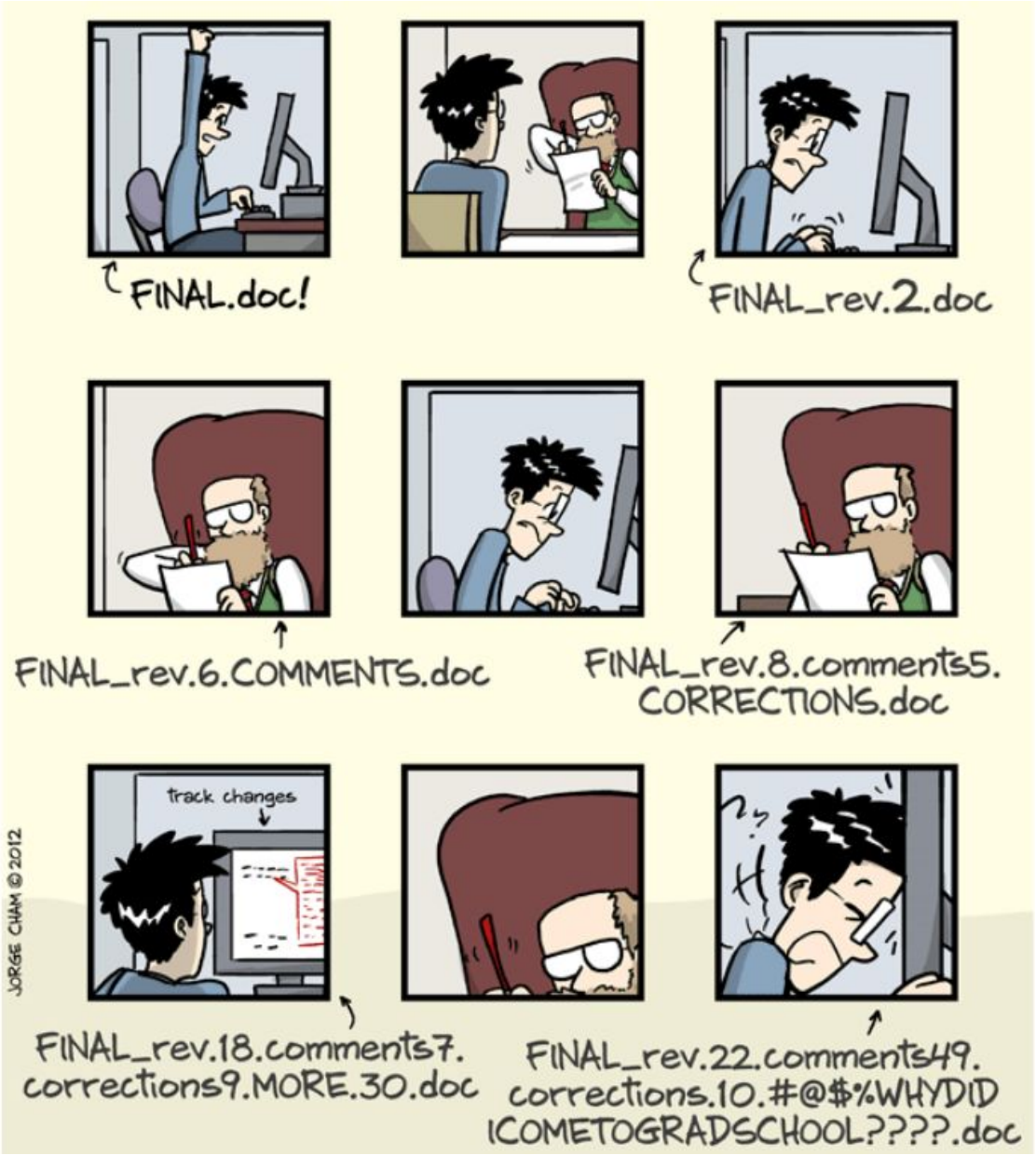
UST, University of Vic, Vic
Computational Biology of RNA Processing, CRG, Barcelona

Outline

- Version control and Git
- Basic Git workflow
- Some Git commands
- Branches
- Remote repositories

1. Version control

Version control



Version control

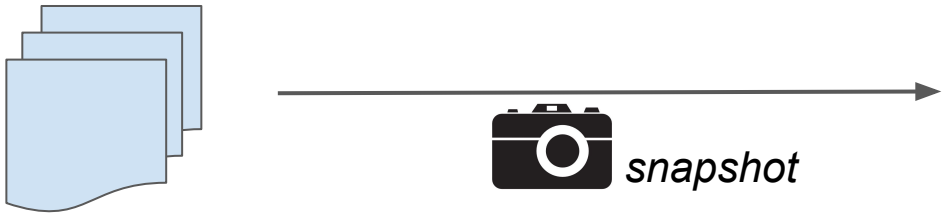
- Trying to *manually* keep track of your files (file naming, etc.), is incredibly error prone!

- Version control is a system that allows you to **record changes** to a file or set of files over time so that you can **recall specific versions** later.

- They allow you to revert selected files back to a previous state, keep different versions of the same project, compare changes over time, collaborate with others, review the changes they introduce, and much more!

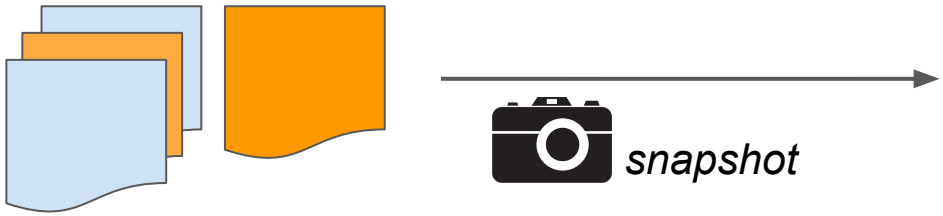
Version control

01/11/18 - **Initial set of 3 files (version 1)**



Repository (*repo*)

15/11/18 - **File 2 changed, new file 4 (version 2)**

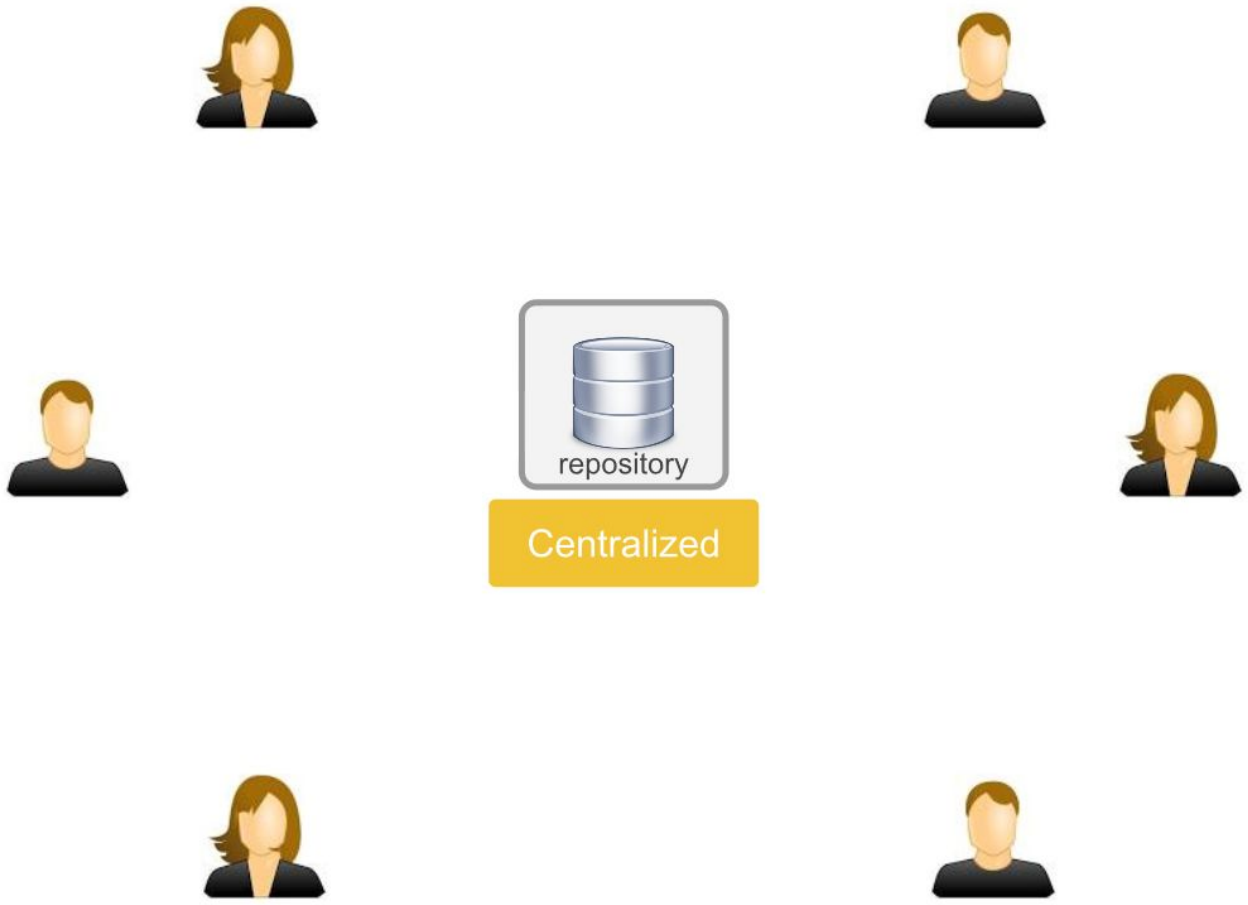


[...]

A ***snapshot*** is a set of changes. Only new changes from one commit to the next one (not full versions) are stored in the *repo*.

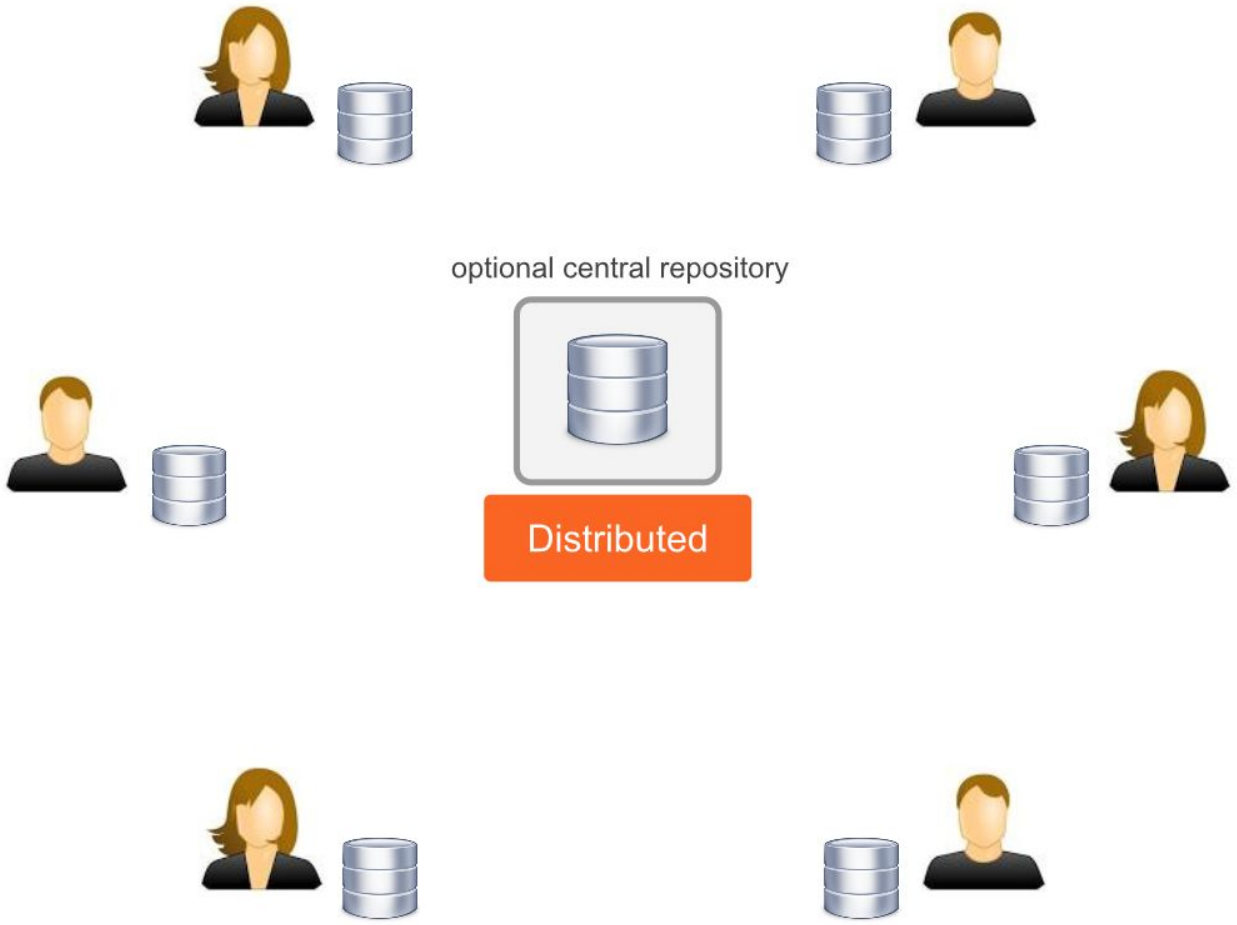
Version control

Centralized version control systems: There is just one central repository to work with (one user at a time).



Version control

Distributed version control systems: each user maintains a complete repository, although there may be a central repository.



Git

Git is a free, open source, distributed version control system, probably the most widely-used. It was created to help in the development of the Linux kernel.

2. Basic Git workflow

Basic concepts

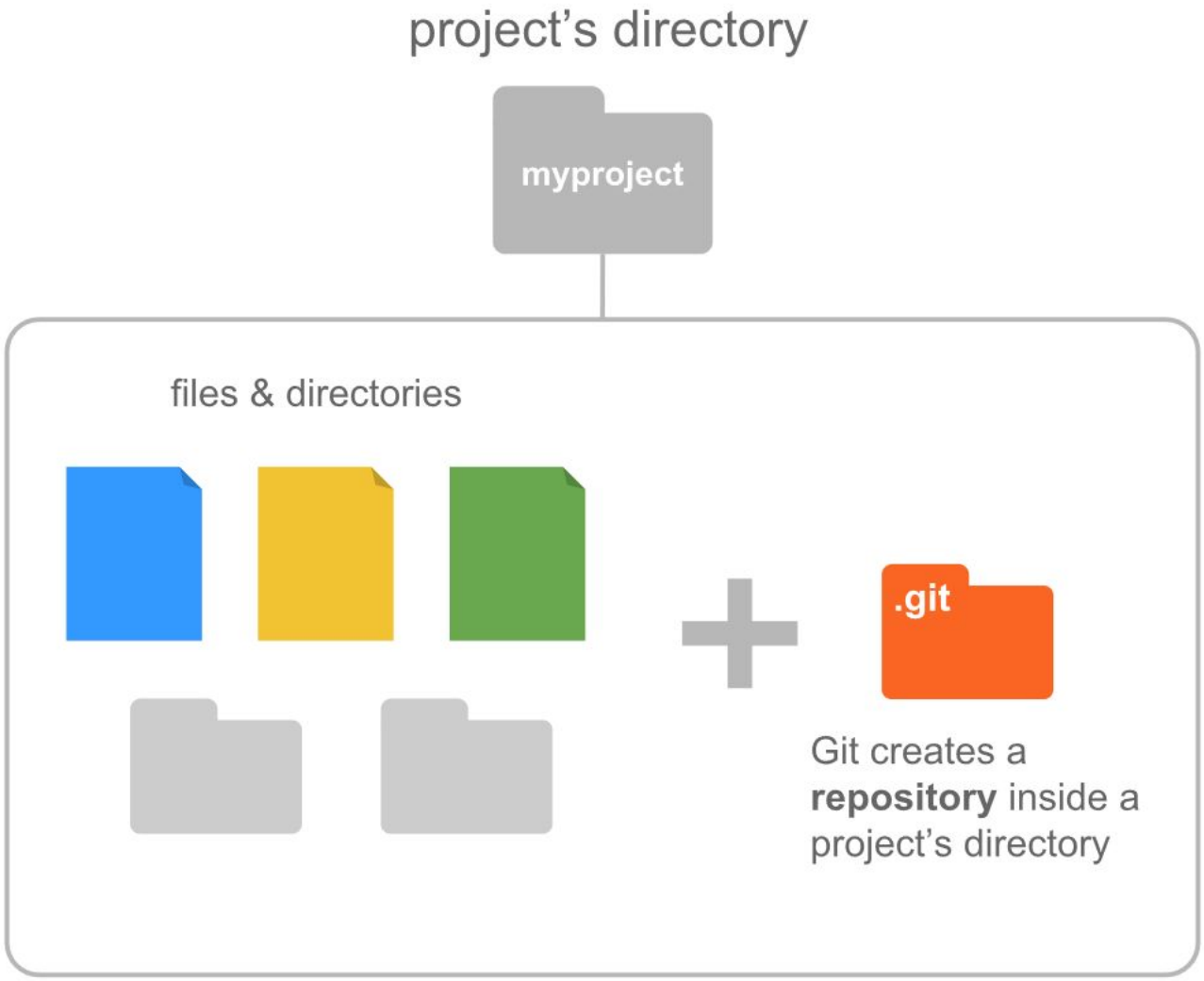
- Your **working directory** (or *working copy*) is the folder that contains -usually- the most recent version of your project files and directories.

- A **repository** is a *database* that contains the history (the different *snapshots* over time) of your project. It often lives in the ".git" subdirectory of your working directory. It can be *local* or *remote*. In Git every copy of a repository is a complete repository.

- The **staging area** or *index* stores information about what will go into your next commit. (It is indeed a file within ".git")

- A **commit** is a point in the Git history. It is a snapshot of the changes present in the staging area, stored in your Git repository.

Basic concepts



Basic concepts

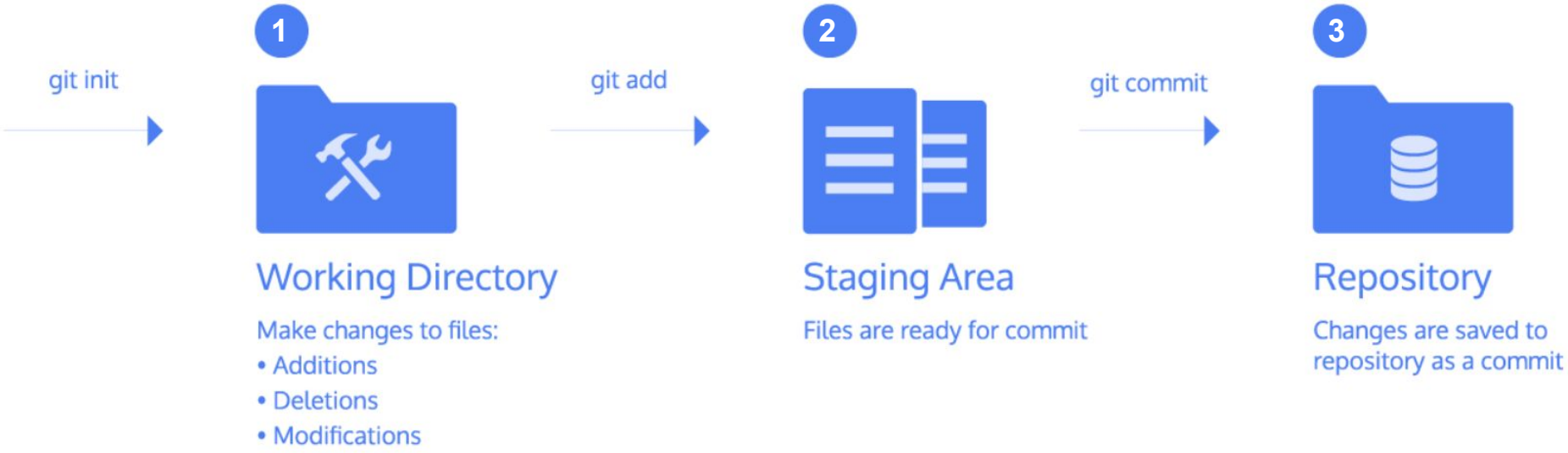
- **Tracked** files are files that were in your last snapshot (either modified or not) or staged. In short, tracked files are files that Git knows about. **Untracked** files are any files in your working directory that were not in your last snapshot and are not in your staging area.

- A **branch** is an active line of development. The default development branch is called ***master***. When using Git, we often work with **multiple branches**, several **collaborators**, etc.

Basic Git workflow

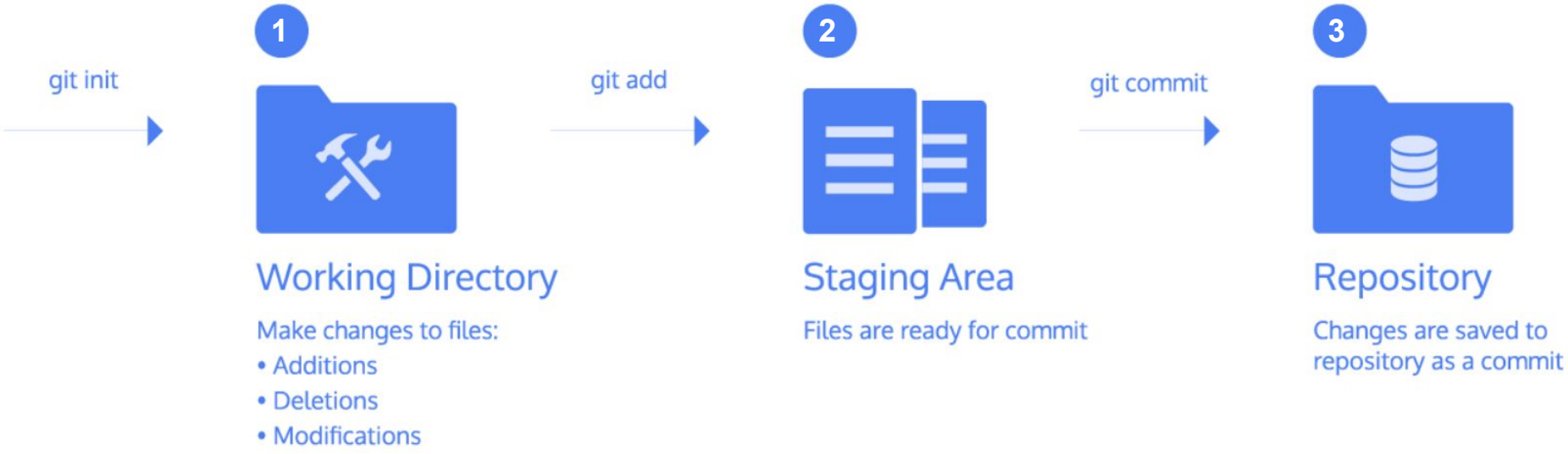
The basic Git workflow goes something like this:

1. **Modify** files in your working directory.
2. Selectively **stage** just those changes you want to be part of your next commit, which adds *only* those changes to the staging area.
3. **Commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git repository.

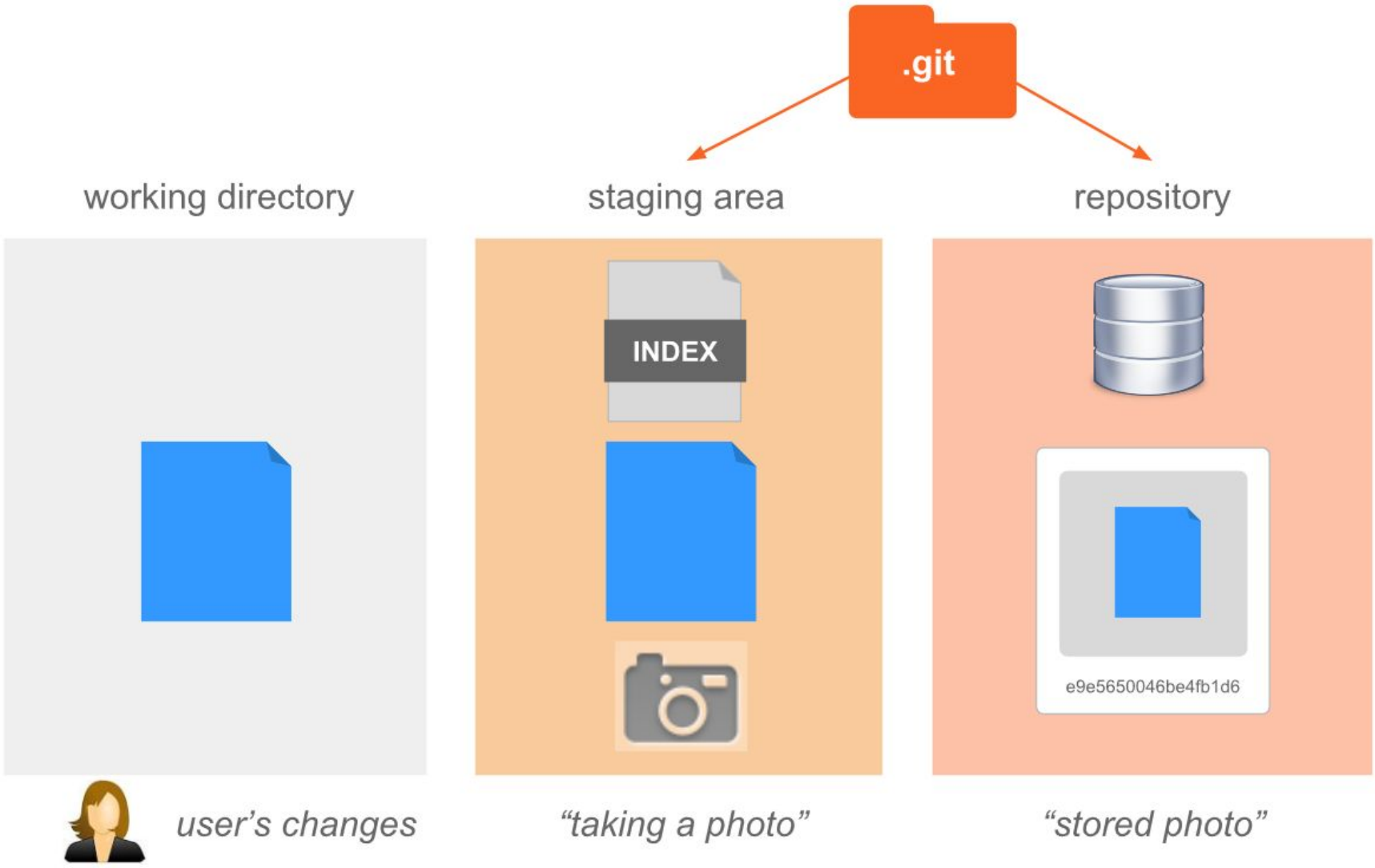


Basic Git workflow

If a particular file was changed but the changes have not been added to the staging area, it is **modified**. If it has been modified and was added to the staging area, it is **staged**, and if it is in the Git repository, it is considered **committed**.



Basic Git workflow

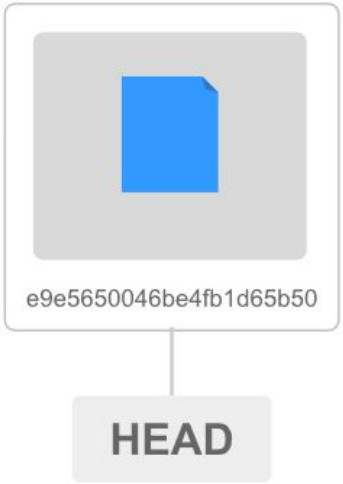


Basic Git workflow

Git stores commits

Each commit has a unique identifier or SHA

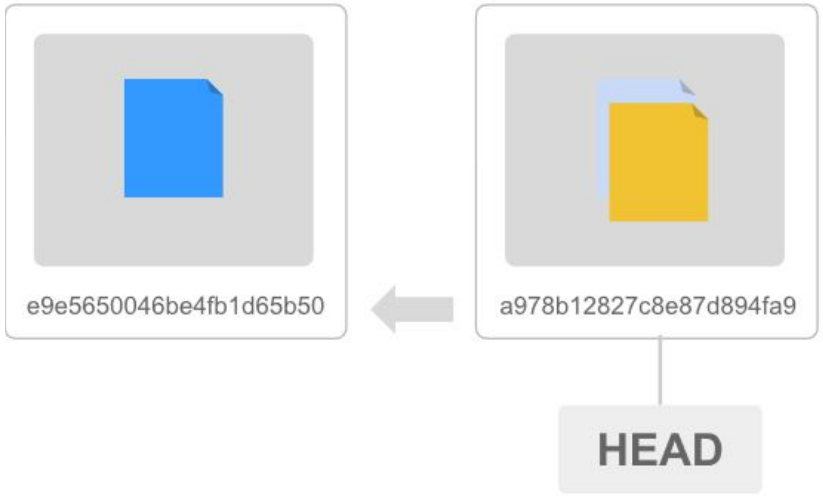
Basic Git workflow



HEAD is a pointer

Typically, HEAD points to the last commit

Basic Git workflow



HEAD is a pointer

Typically, HEAD points to the last commit

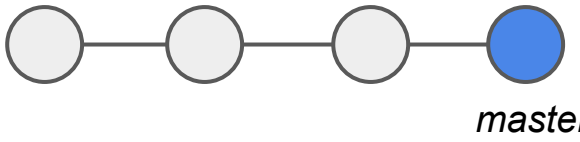
Basic Git workflow

HEAD is a pointer

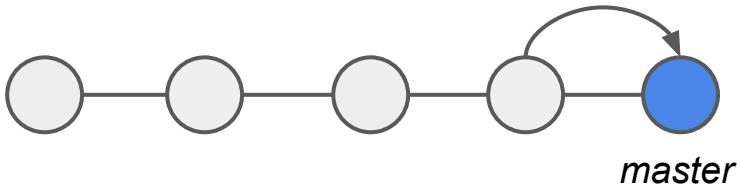
Typically, HEAD points to the last commit

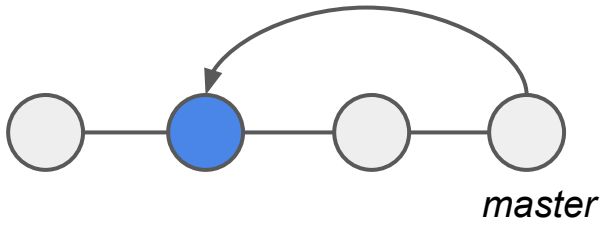
Basic Git workflow

- You can consider your project history as a series of connected commits:

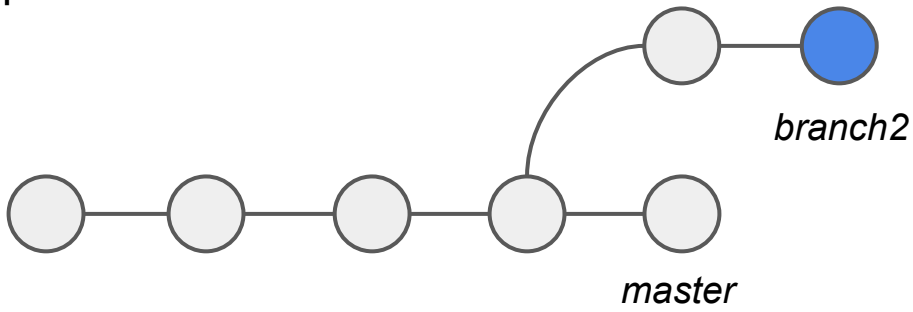


- You can keep adding new commits, check the previous ones, etc.





- However, this process is often **not** linear!

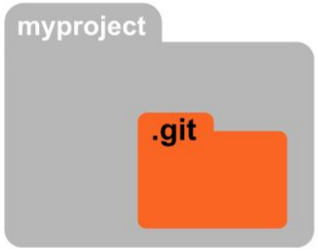


3. Some Git commands

Starting up a repository

- Create a new repository:

```
git init <directory>
```



- Clone an existing repository:

```
git clone <repository> <directory>
```

Starting up a repository

3 types of configuration

System level

apply to every user of the computer

`/etc/gitconfig`

User level

apply to a single user

`~/.gitconfig`

Project level

project to project configurations

`my_project/.git/config`

`git config --system`	system level
`git config --global`	user level
`git config`	project level

Starting up a repository

- Some configuration:

```
git config --global user.name "Your Name"
git config --global user.mail "Your e-mail"
git config --global push.default simple

git config --system core.editor <editor>
```

- Ignore some files and directories:

```
.gitignore
```

Save changes, inspect the repo

- Adding changes in the working directory to the staging area:

```
git add <file>
```

- Commiting the staged snapshot to the repository:

```
git commit -m "message"
```

- List files staged, unstaged and untracked:

```
git status
```

- Show the entire commit history:

```
git log
```

Save changes, inspect the repo

- Delete a file from the working directory and stage the deletion:

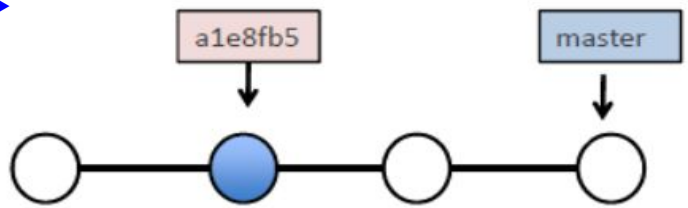
```
git rm <file>
```

- Check an old version of the working directory or a file:

```
git checkout <commit>
git checkout <commit> <file>
```

- Return to master:

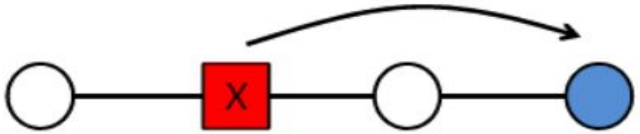
```
git checkout master
git checkout master <file>
```



Undo changes

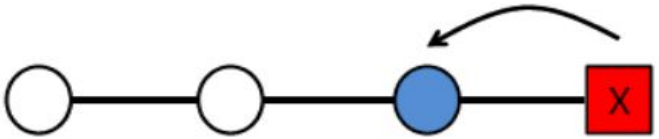
- Undoing a committed snapshot (**safe** way to undo changes):

```
git revert <commit>
```



- Removing committed snapshot (**dangerous** way to undo changes):

```
git reset <commit>
```



- Removing untracked files from working directory:

`git clean` Not undoable (try `git clean -n` first)

- Fix the last commit (change commit message, add new files)

```
git commit --amend
```

4. Branches

Branches

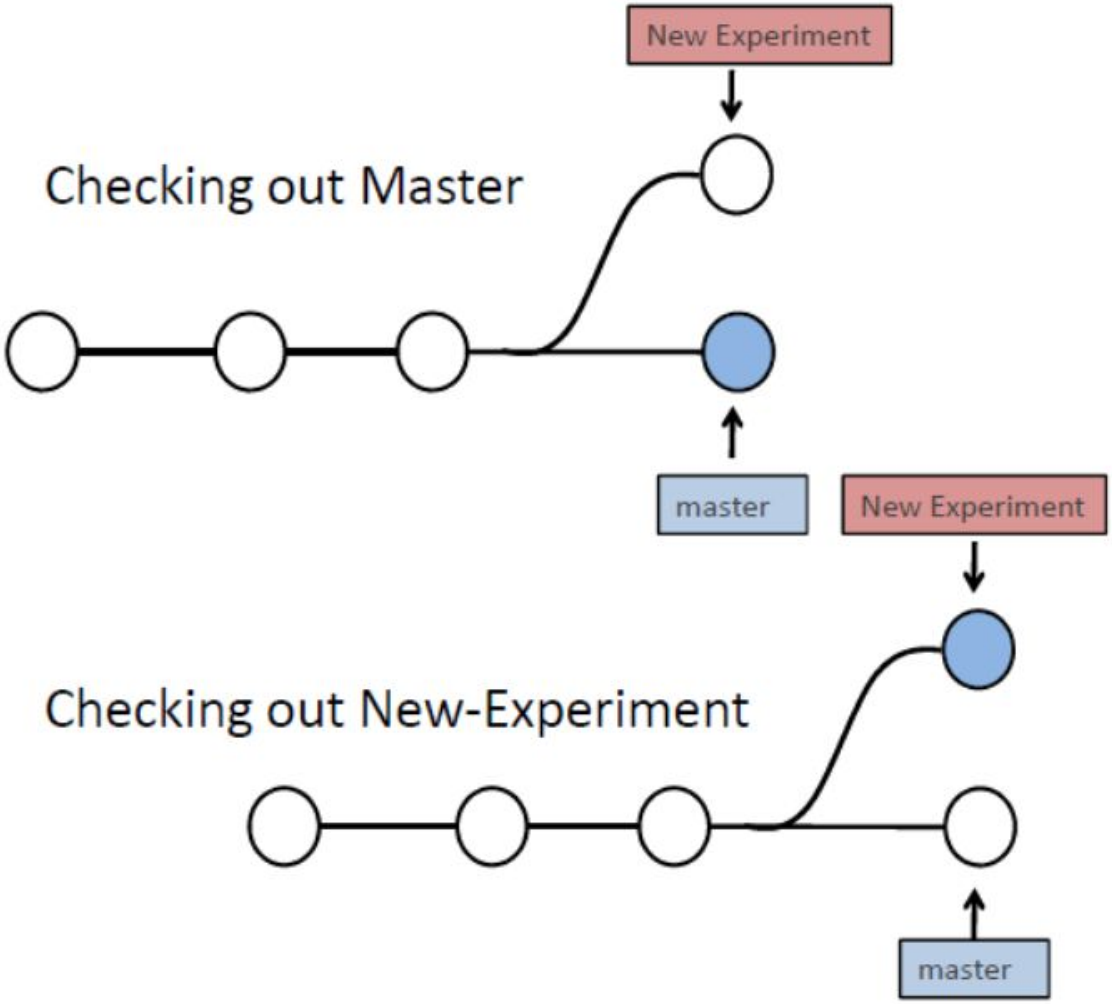
- A **branch** represents an independent line of development (a brand new working directory, staging area and project history).

`git branch`	list
`git branch <branch>`	create
`git branch -m <branch>`	rename
`git branch -d <branch>`	delete (`-D` force delete)

Navigating between branches:

```
git checkout <branch>
```

Branches


New Experiment
Checking out Master
master
New Experiment
Checking out New-Experiment
master

Merging

- Put a forked history back together again

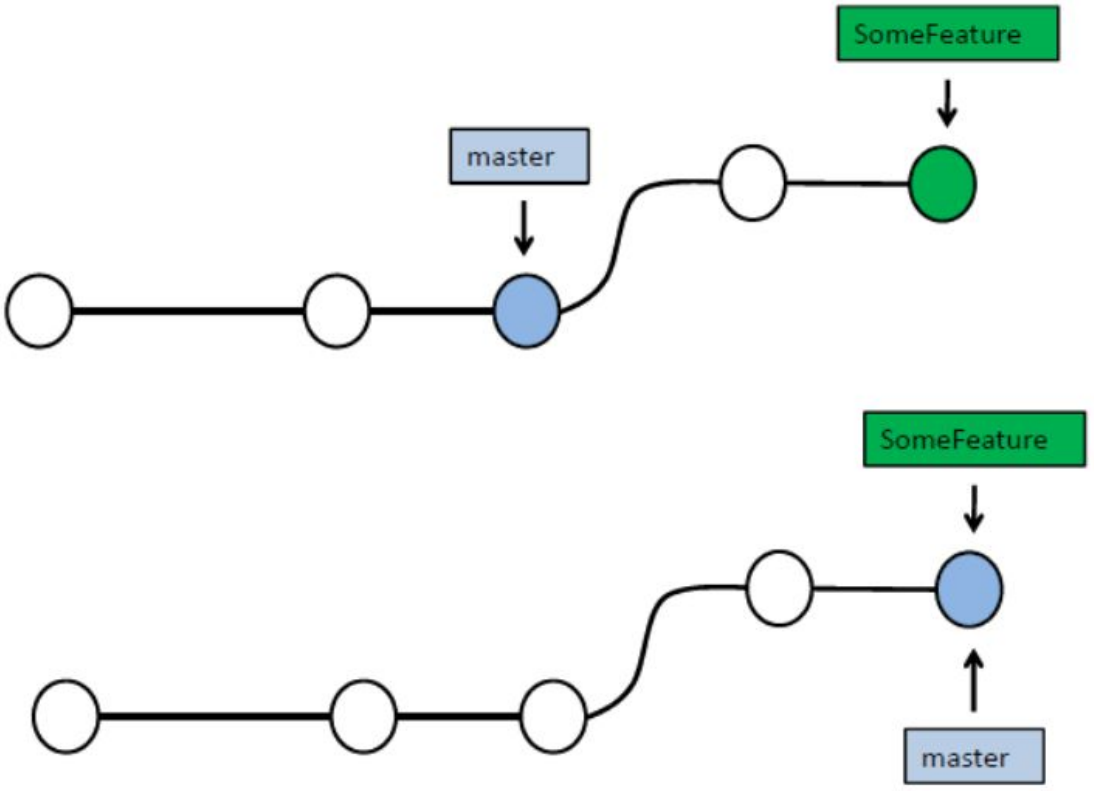
```
git merge <branch>
```

  - A **fast-forward merge** can occur when there is a linear path from the current branch tip to the target branch

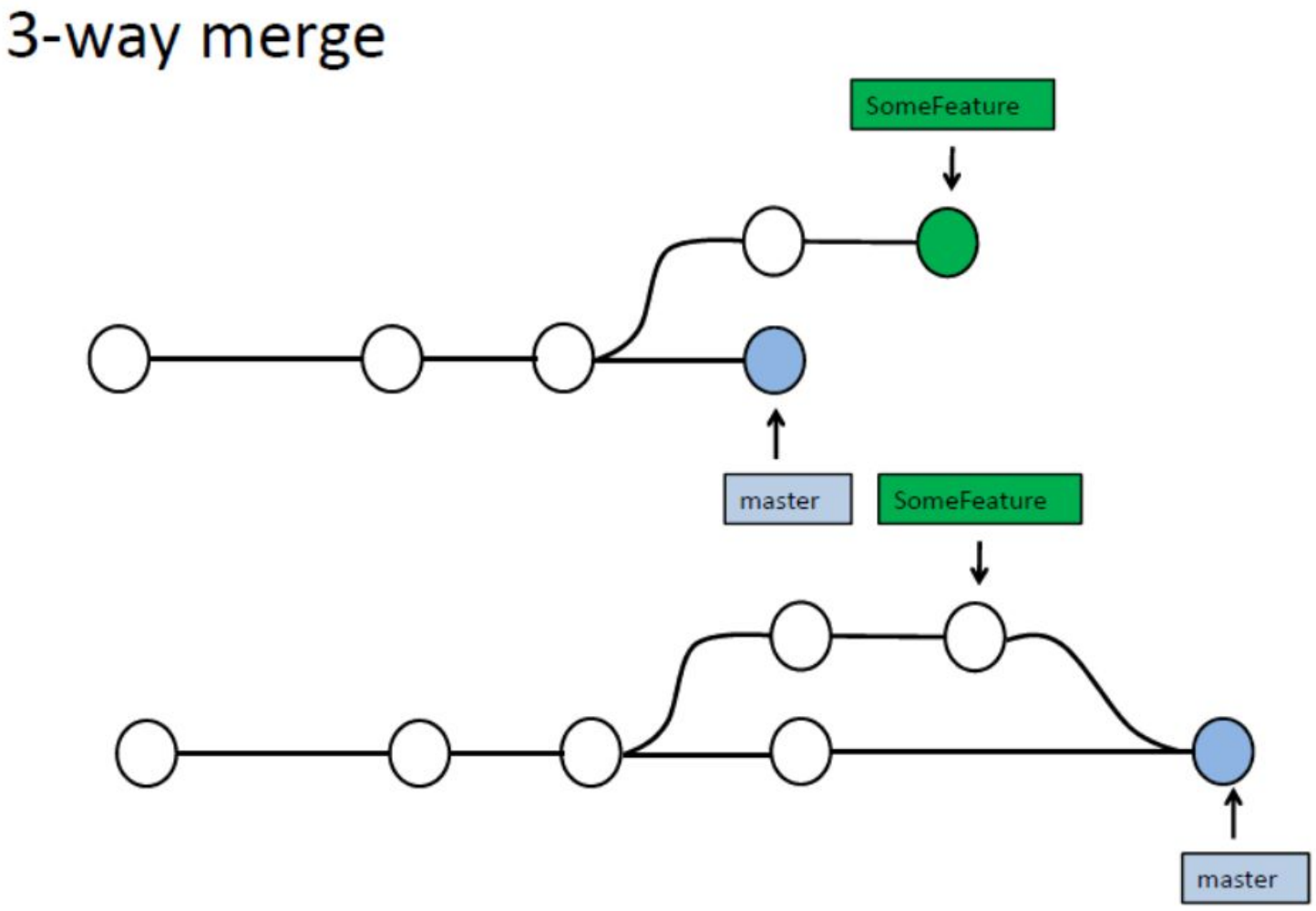
  - A **3-way merge** occurs when there is not a linear path and a dedicated commit is used to tie together the two histories

Merging

Fast forward merge



Merging



Merge conflicts

If two branches change the same part of the same file, Git stops right before the merge commit. Conflicts must be resolved manually.

`git status`	shows which files need to be resolved
edit the file and fix the conflict	
`git add`	add the conflicting file
`git commit`	generate the merge commit

5. Remote repositories

Local and remote repositories

- ***Local*** repositories are located on your computer (and the computers from your collaborators), while ***remote*** repositories are hosted on a server accessible to all of you, from different locations (most likely on the internet or on a local network).
- Technically, a remote repository doesn't differ from a local one. However, a remote repository doesn't have a working directory.
- The actual *work* on your project happens *only* in your local repository: all modifications are made and committed locally. Then, those changes *can* be uploaded to a remote repository in order to share them.
- In Git, there are several commands to interact with a remote repository.

Local and remote repositories

Creating, viewing and deleting connections to other repositories:

`git remote`	list all connections
`git remote add <name> <url>`	create new connection
`git remote rm <name>`	remove connection
`git remote rename <old> <new>`	rename connection

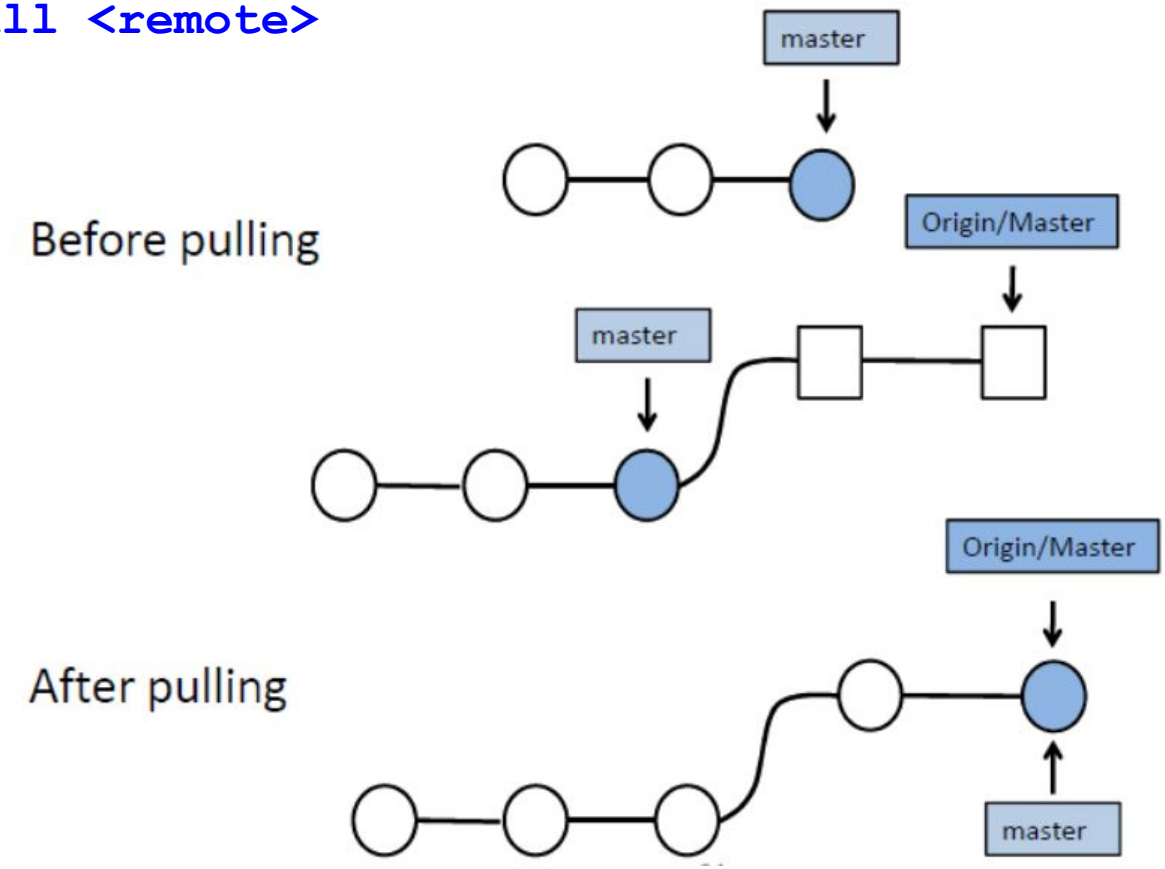
`my_project/.git/config`

When you clone a repository using `git clone`, it automatically creates a connection called **origin** pointing back to the cloned repository.

Import commits from the remote (remote >>> local)

Import commits from the remote repository to a local repository (fetch and merge the remote copy of the branch into the local copy).

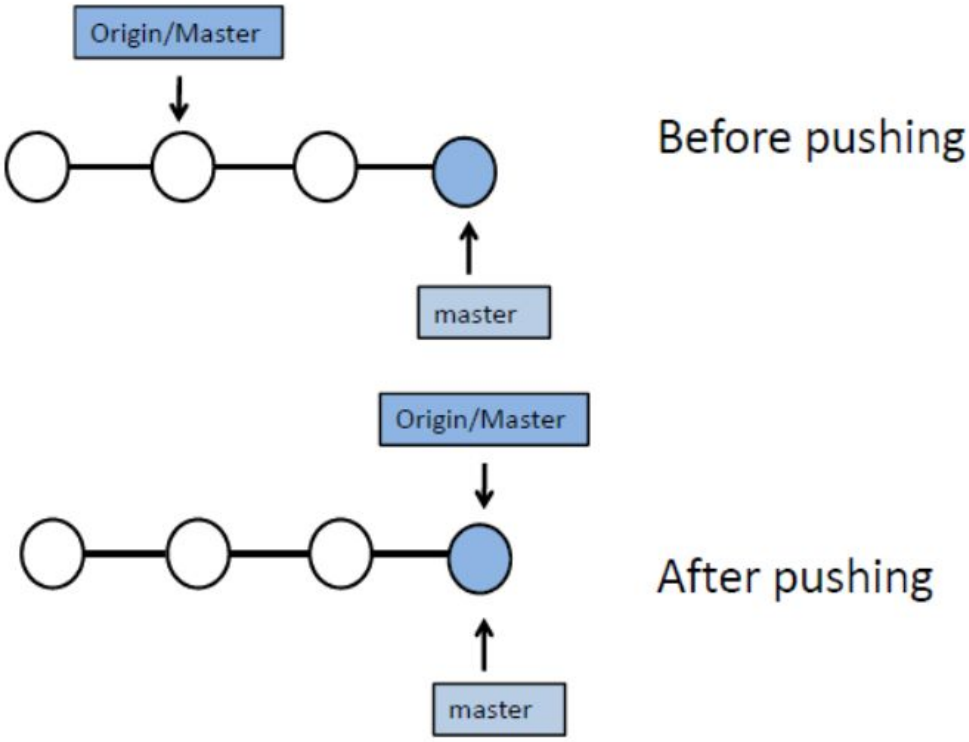
```
git pull <remote>
```



Transfer commits to the remote (local >>> remote)

Transfer commits from a local repository to the remote repository (create a copy of the local branch in the remote repo, update must be a fast-forward merge).

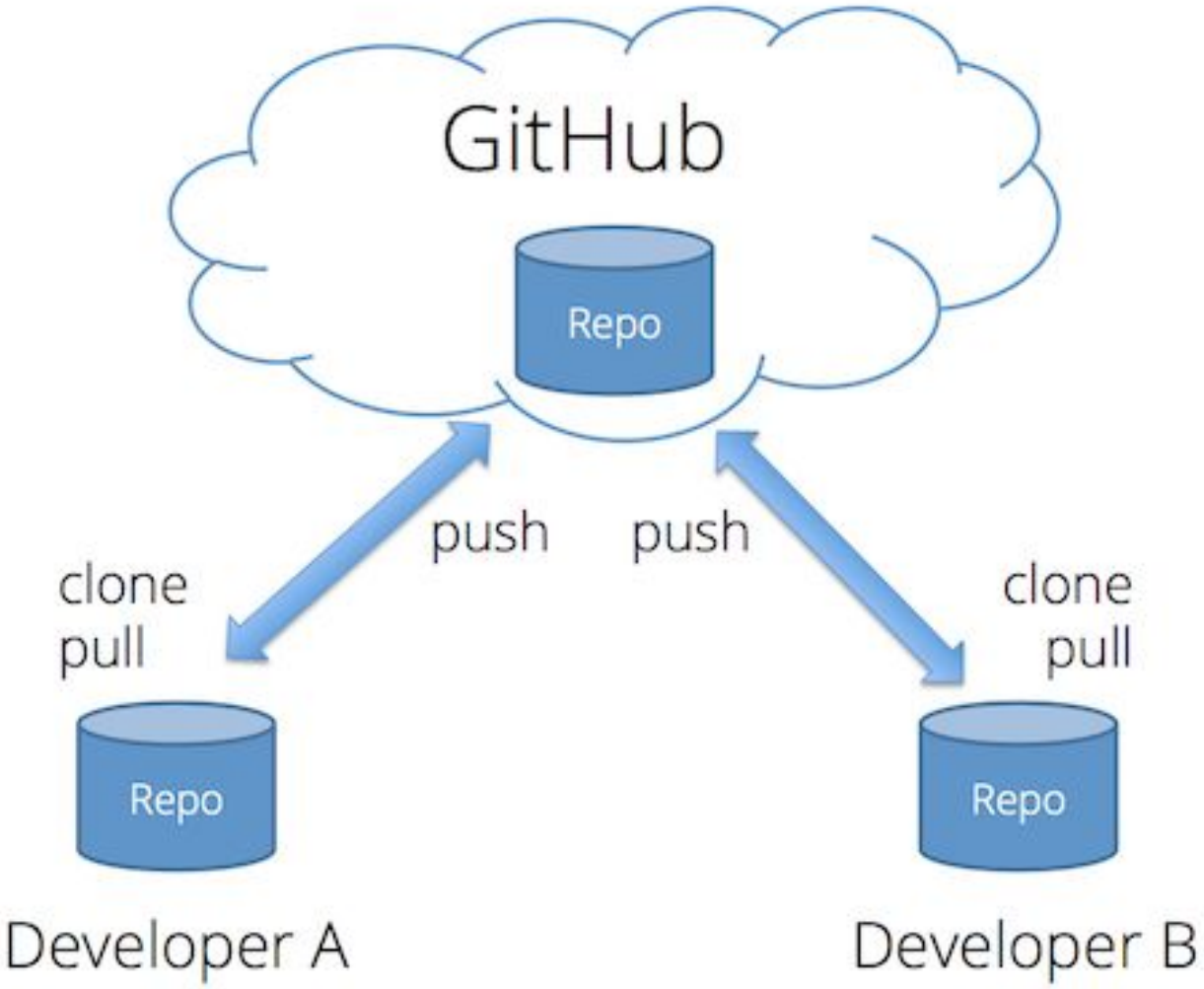
```
git push <remote> <branch>
```



GitHub

- GitHub is most widely-used web-based hosting service for version control using Git, although there are others, such as GitLab or Bitbucket.
- In GitHub, we can host repositories that act as *remotes* for our *local* ones. You can have a look at some repositories at https://github.com/guigolab
- You have been already using it! Actually you got access to these slides from GitHub ;)

Basic Git Workflow (interacting with the *remote*)



Hands On

https://github.com/dgarrimar/teaching/wiki/Git-hands-on

Additional resources

- Git documentation
  - https://git-scm.com/book/en/v2
- Git/GitHub cheatsheet
- https://es.atlassian.com/git/tutorials