

Version control with `git` for scientists



PY Barriat & F. Massonnet

May 05, 2022

some parts inspired on slides from CISM

Discuss

How do you manage different file versions ?

How do you work with collaborators on the same files ?






Notions of code versioning

Track the history and evolution of the project

think of it as a series of snapshots (**commits**) of your code

Benefits

- possibility to go back in time 
 - tracking bugs
 - recovering from mistakes
- Information about the modification 
 - who, when, why

- team work 
 - **Simultaneous** work on a project
 - No need to send email to say "I'm working on that file" (dropbox organization)
 - **Asynchronous** synchronisation
 - Allow work Offline (opposite to overleaf project)
 - Need conflict resolution

Different usage

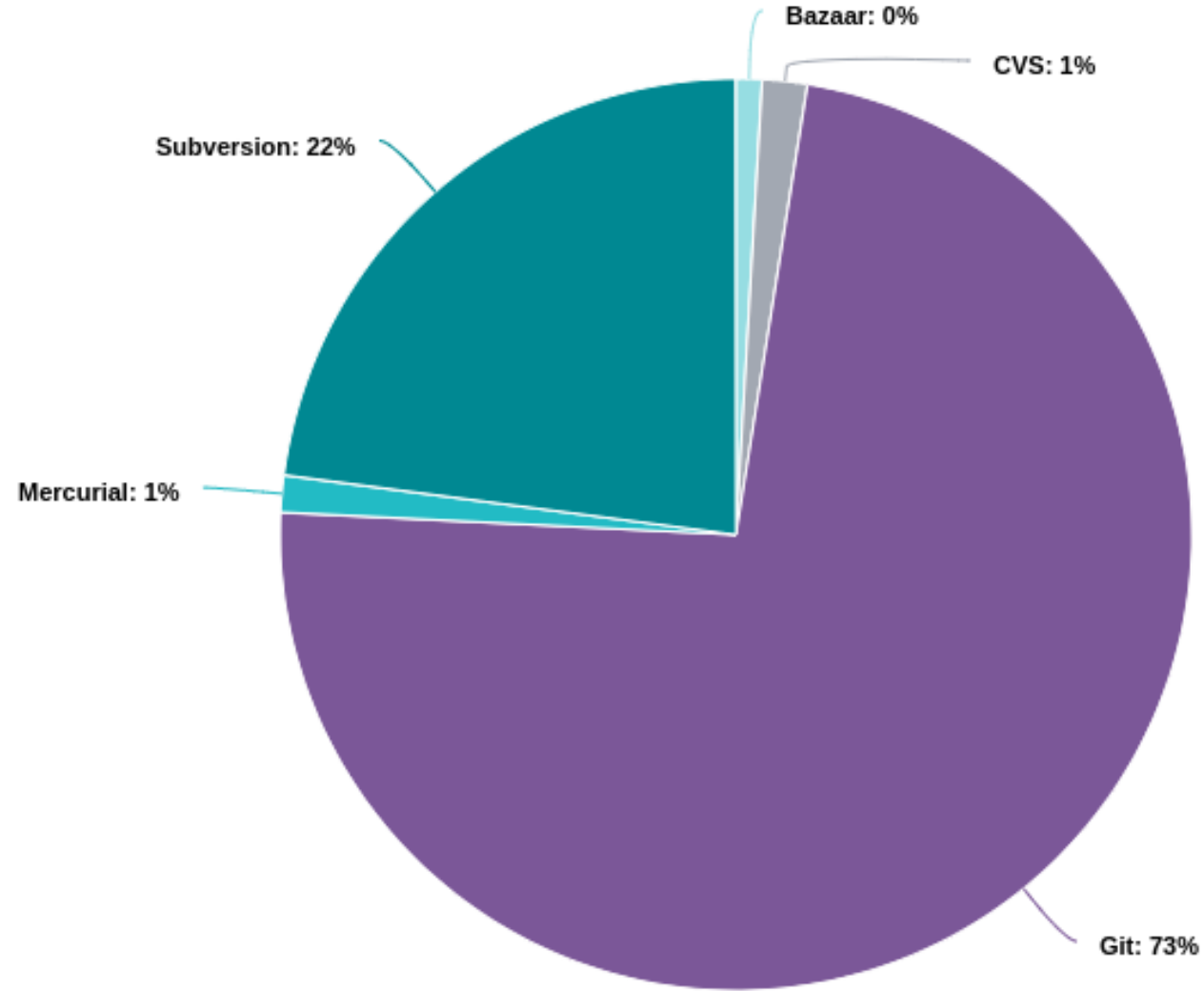
- local
- client-server (Subversion)
- distributed (Git)

Workflow

Testing new idea (and easy way to throw them out) 🚧

Multiple version of the code

- Stable (1.x.y)
- Debug (1.x.y+1)
- Next "feature" release (1.x+1.0)
- Next "huge" release (2.0.0)



Open-Source Code

Compare Repositories

What is `git` ?

Version control system

- Manage different versions of files
- Collaborate with yourself
- Collaborate with other people

Why use `git`

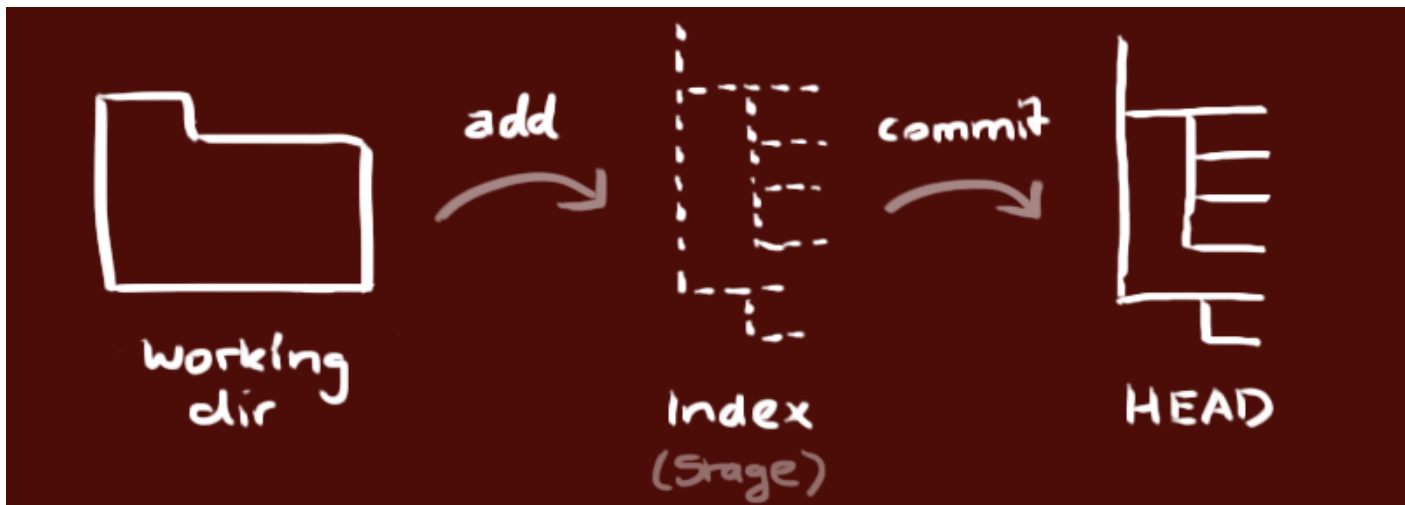
"Always remember your first collaborator is your future self, and your past self doesn't answer emails"

Christie Balhai 😊

git workflow

Your local repository consists of **three areas** maintained by `git`

- the first one is your **Working Directory** which holds the actual files
- the second one is the **INDEX** which acts as a staging area
- and finally the **HEAD** which points to the last commit you've made



Workspace



./WORKDIR

Index



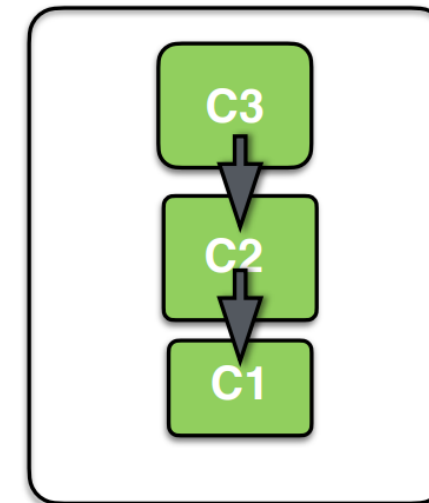
.git/index

Staging area

Repository



.git/



Getting started with `git`

checkout a remote repository

create a local working copy of a remote repository

```
git clone https://gogs.elic.ucl.ac.be/TECLIM/Git_Training.git
```

add & commit

you can propose changes (add it to the **INDEX**)

```
git add <filename>
```

you can commit these changes (to the **HEAD**)

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


commit

`git` versioning is a succession of snapshot of your files at key time of their development

each snapshot is called **commit** which is :

- all the files at a given time
- a unique name (SHA1)
- metadata
 - who created, when, info
- pointer to previous(es) commit(s)

Your changes are now in the **HEAD** of your local working copy.

push

to send those changes to your remote repository

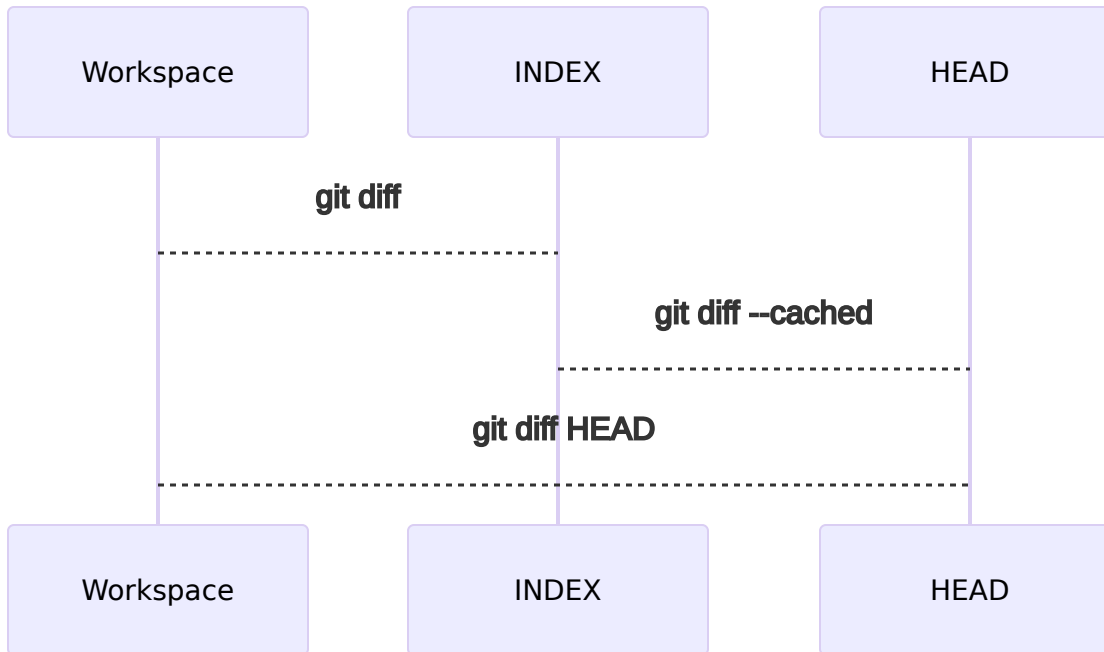
```
git push
```

pull

to update your local working directory to the newest commit, to fetch and merge remote changes

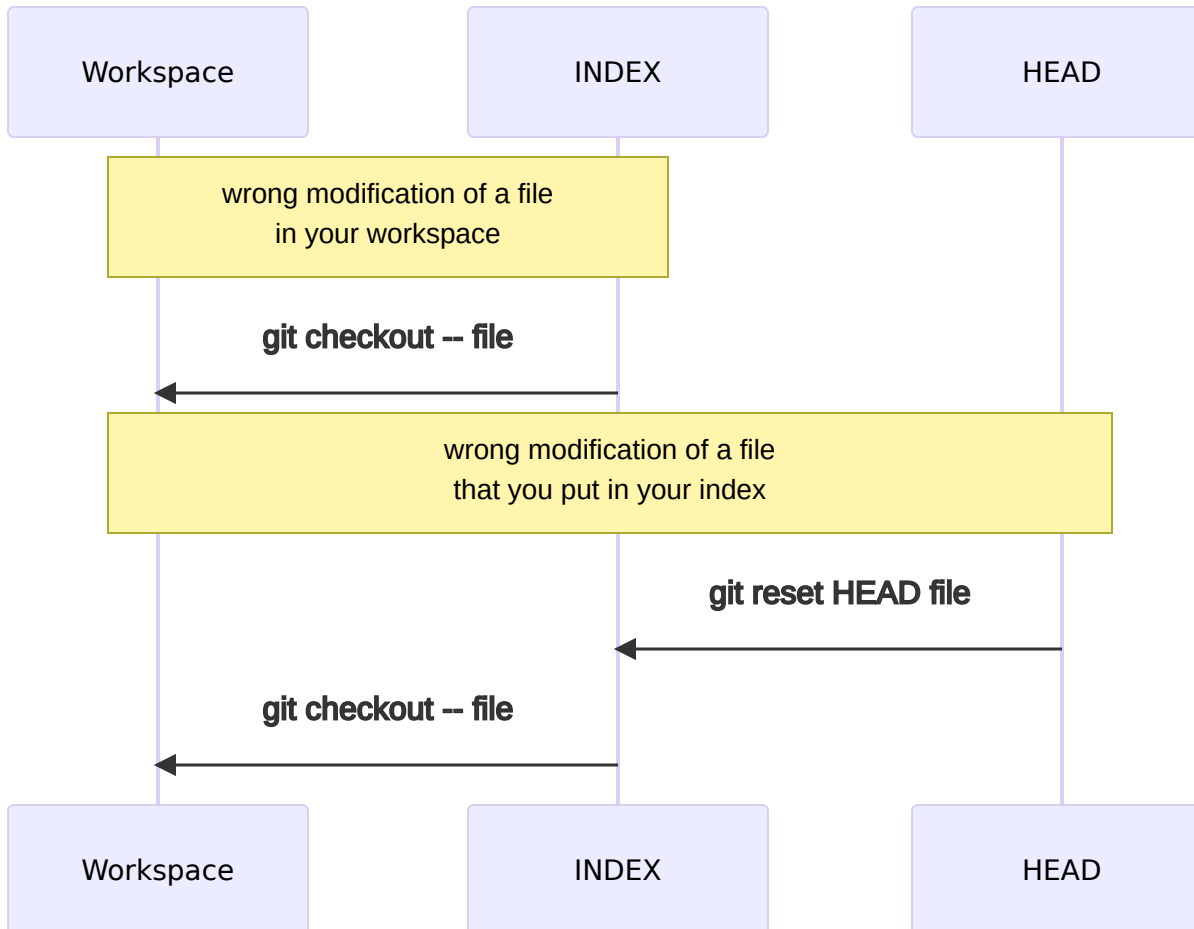
```
git pull
```


git diff



git undo

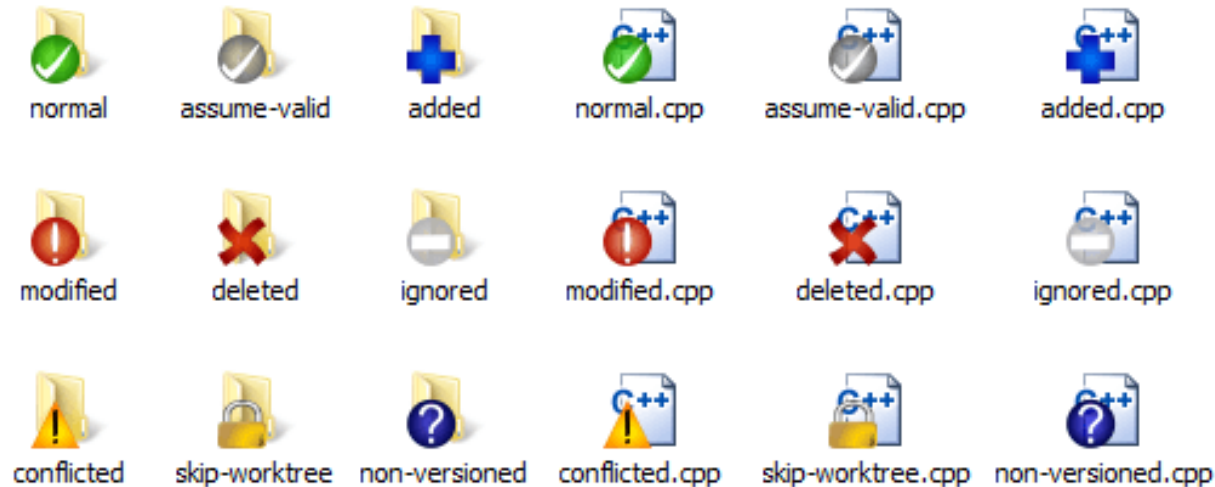
In case you did something wrong (which for sure never happens 😊)



Windows users

How commonly do programmers use Git GUIs instead of the command line ?

Use programs like `SourceTree` or `TortoiseGit`



But, **to be familiar with Git**, try the command line

clone, push/pull, merge, rebase, log, tag, format-patch/am, bisect, blame, etc

Simple Git Exercises

First, configure your environment (just once) 🚧

on your laptop, on your ELIC account, etc

```
git config --global user.name "Your Name"
git config --global user.email "foo@bar.be"
git config --global color.ui auto
git config --global core.editor "vim"

git config --list
```

Now, clone https://gogs.elic.ucl.ac.be/TECLIM/Git_Training.git

Theses are very simple exercices to learn to manipulate git.

In each folder, simply run `./create.sh` and follow the guide 😎

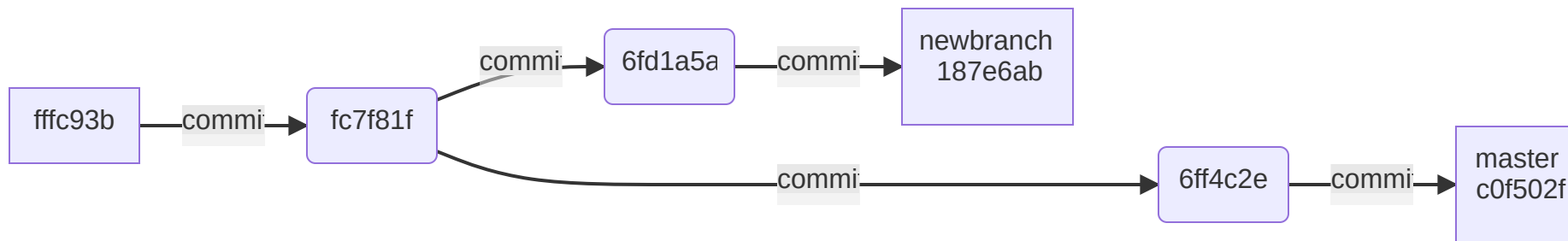
git branches

- a **branch** is pointer to a commit (represent an history)
- a **branch** can point at other commit
| it can move !
- a **branch** is a way to organize your work and working histories
- since commit know which commits they are based on, **branch** represents a commit and what came before it
- a branch is cheap, you can have multiple **branch** in the same repository and switch your working dir from one **branch** state to another

branches demo

```
git commit
git checkout -b newbranch
git checkout newbranch
git commit
git commit
git checkout master
git commit
git commit
```

default branch: **master**



- create a new branch : `git checkout -b newbranch`
- switch to a branch : `git checkout newbranch`
- delete a branch : `git branch -d newbranch`
- list all branches : `git branch -a`
| see both local and remote branches

branch is cheap : do it often 👍

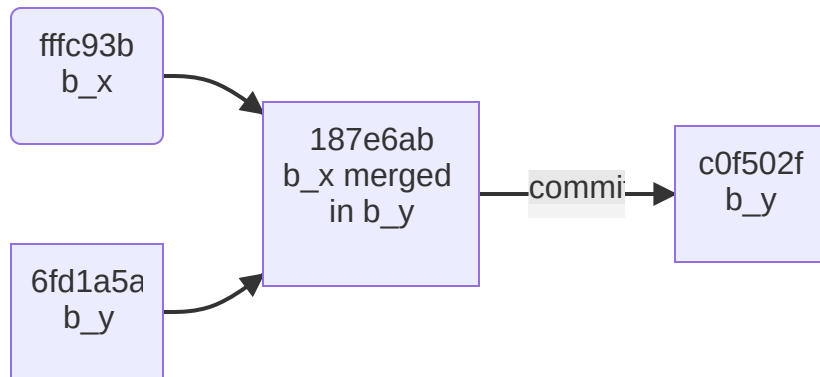
| branch allow to have short/long term parallel development

merging branches

the interest of branch is that you can **merge** them

include in one (branch) file the modification done somewhere else

```
git merge bx  
git branch -d bx  
git commit
```



Difference between `git` & `GitHub` ?

`git` is the version control system **service**

| `git` runs local if you don't use GitHub

`GitHub` is the hosting service : **website**

| on which you can publish (push) your git repositories and collaborate with other people

Github

- It provides a backup of your files
- It gives you a visual interface for navigating your repos
- It gives other people a way to navigate your repos
- It makes repo collaboration easy (e.g., multiple people contributing to the same project)
- It provides a lightweight issue tracking system

... and GitLab vs GitHub vs others

GitLab is an alternative to GitHub

GitLab is free for unlimited private projects. GitHub doesn't provide private projects for free

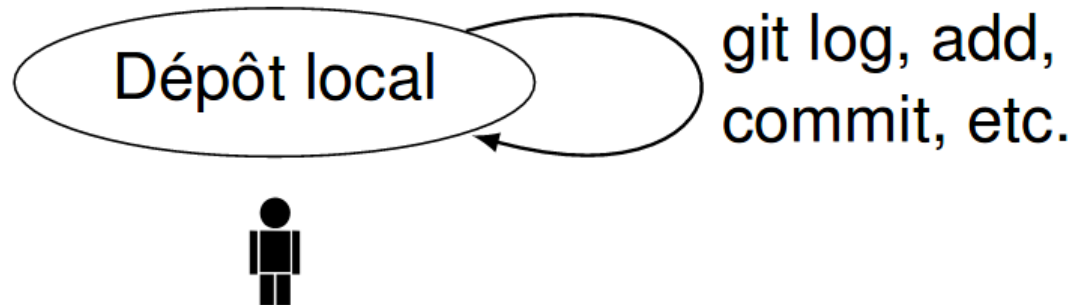
And for **ELIC**, Gogs does the job: <https://gogs.elic.ucl.ac.be/>

- shares the same features
 - dashboard, file browser, issue tracking, groups support, webhooks, etc
- easy to install, cross-platform friendly
- uses little memory, uses little CPU power
- ... and 100% free 😊

What is `git` good for ?

Local

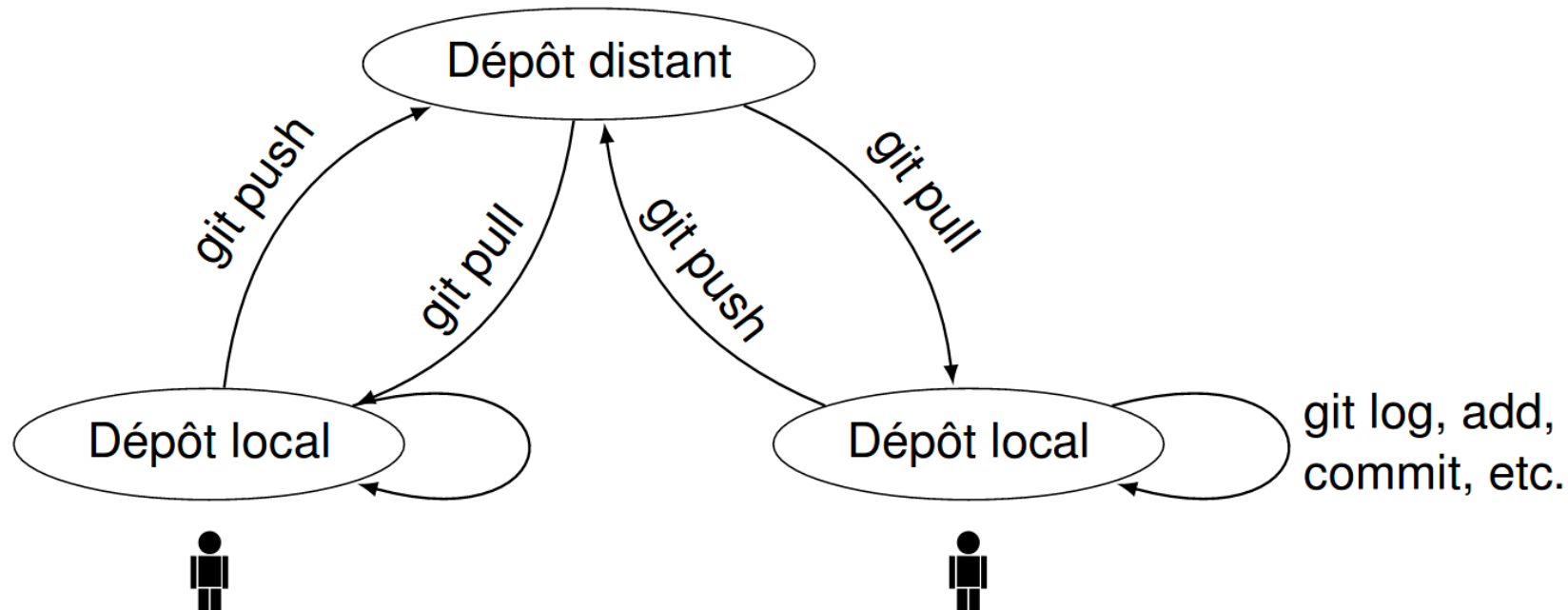
Backup, reproducibility



Utilisation locale

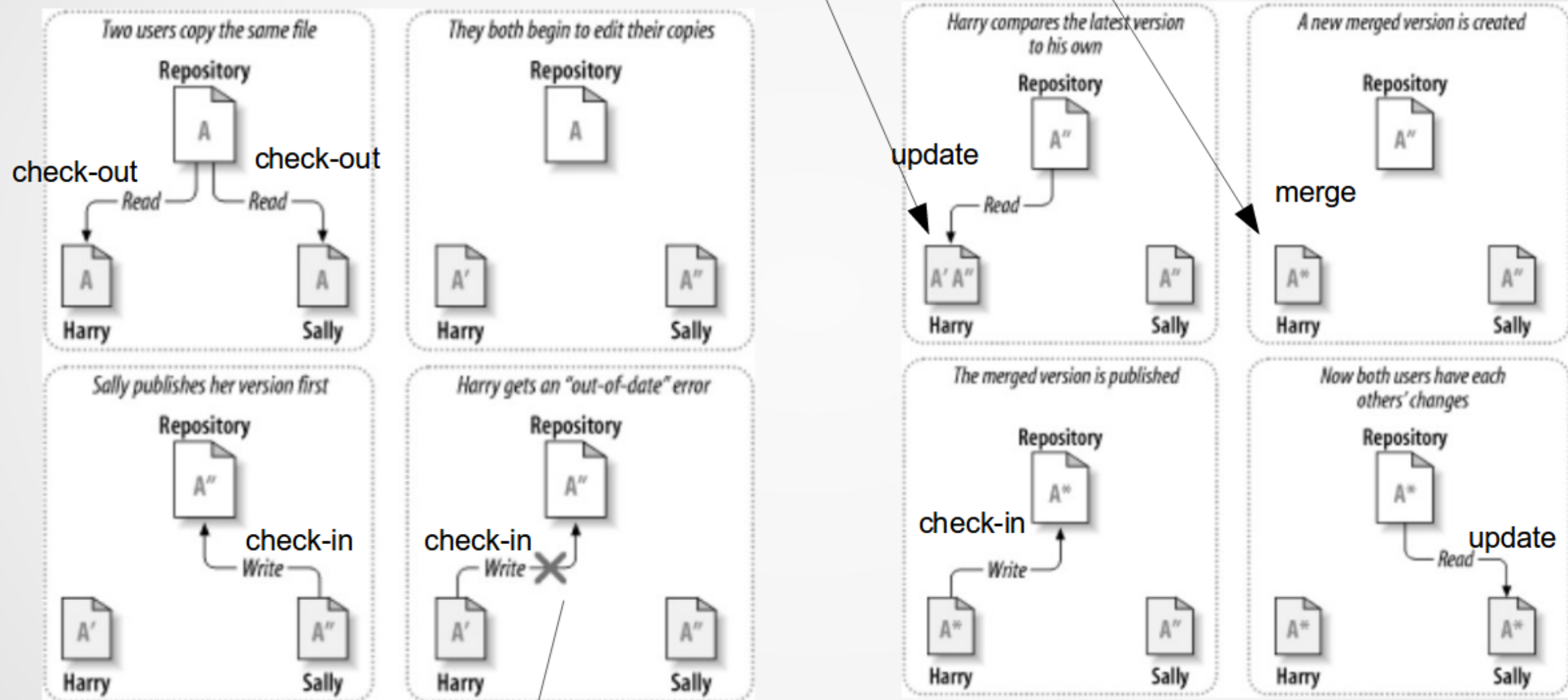
Client-Server

Backup, reproducibility, collaboration



Dépôt commun distant
(Gitolite, Redmine, FusionForge, *GitHub*)

copy-modify-merge solution




cvcs commit: Up-to-date check failed for A

git conflict

multiple version of files are great

- not always easy to know how to merge them
- conflict will happen (same line modify by both user)

conflict need to be resolved manually !

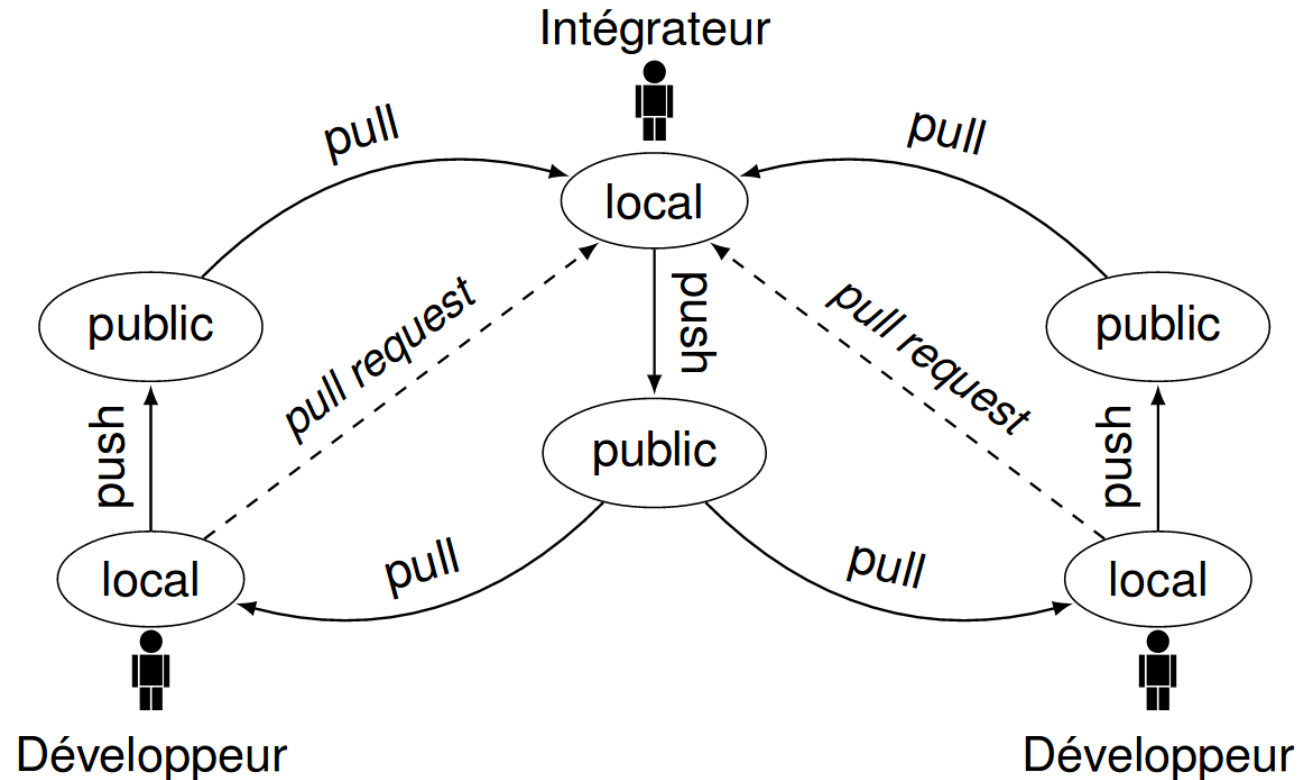
- boring task
- need to understand why a conflict is present !
- **do not be afraid of conflict !** 

Do not try to avoid them at all cost !

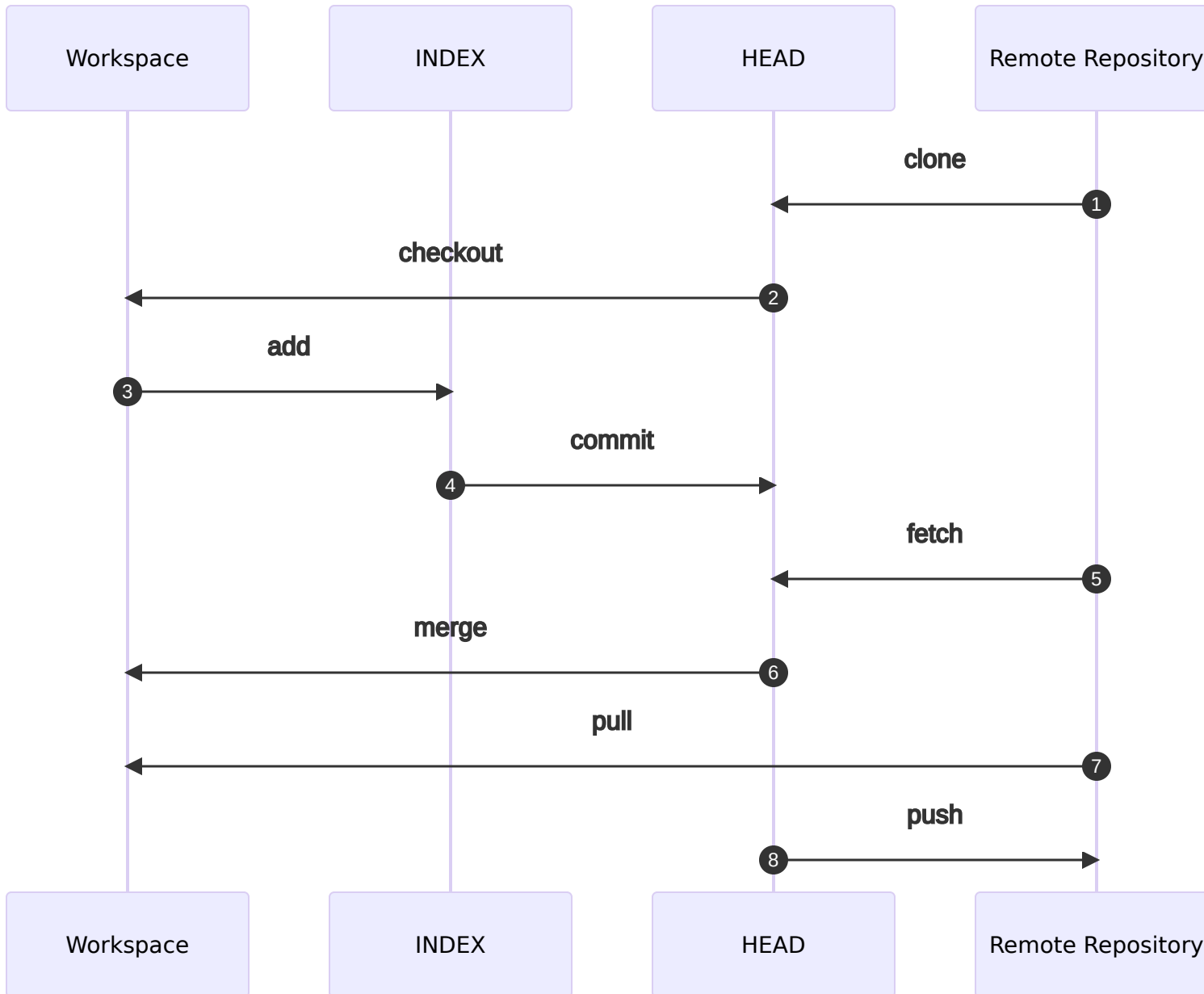
- stay in sync as most as possible and keep line short

Distributed

Backup, reproducibility, collaboration, transparency



Utilisation distribuée (*GitHub*, *Linux*)



Conclusion

- **versioning** is crucial both for small/large project !
- avoid dropbox for paper / project 😞
- do meaningful commit
- do meaningful message
- `git` more complicated but the standard 😊

Version control with Git for scientists