Git 101

Matheus Tavares

matheustavares.gitlab.io

Version control

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

- Git Pro Book

Git, GitHub, GitLab, ...

git ≠ GitHub

GitLab

Git, GitHub, GitLab, ...

git

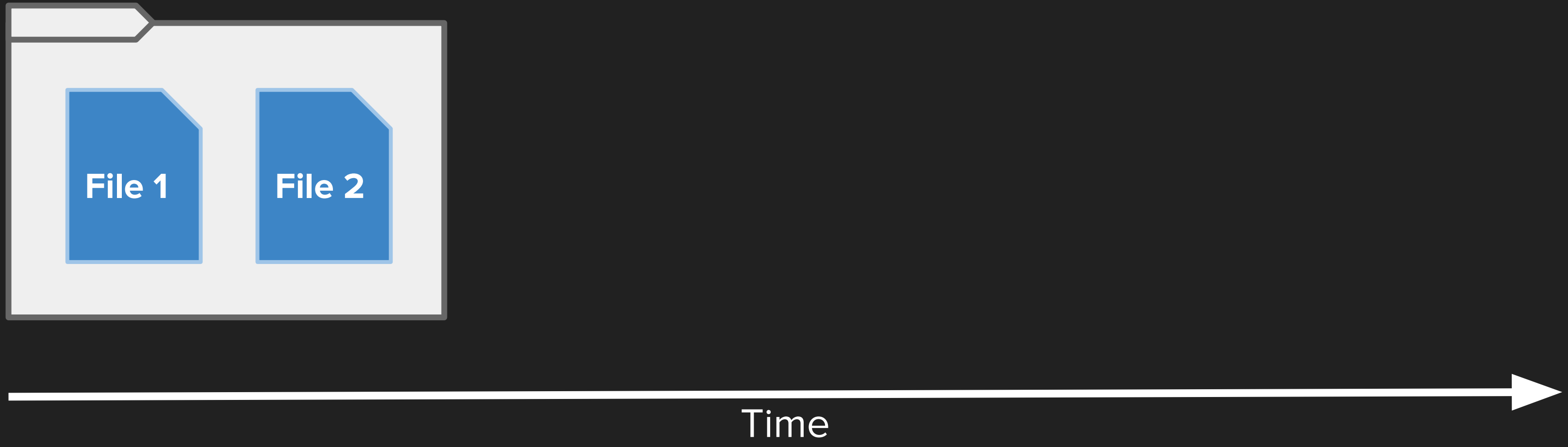
$\neq$

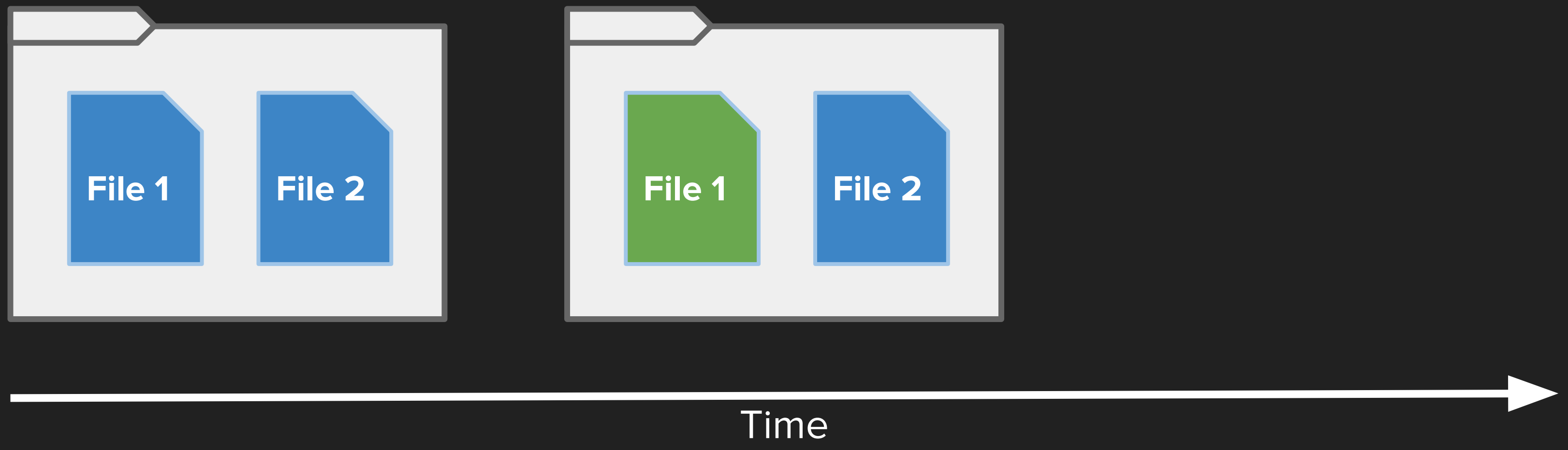
GitHub

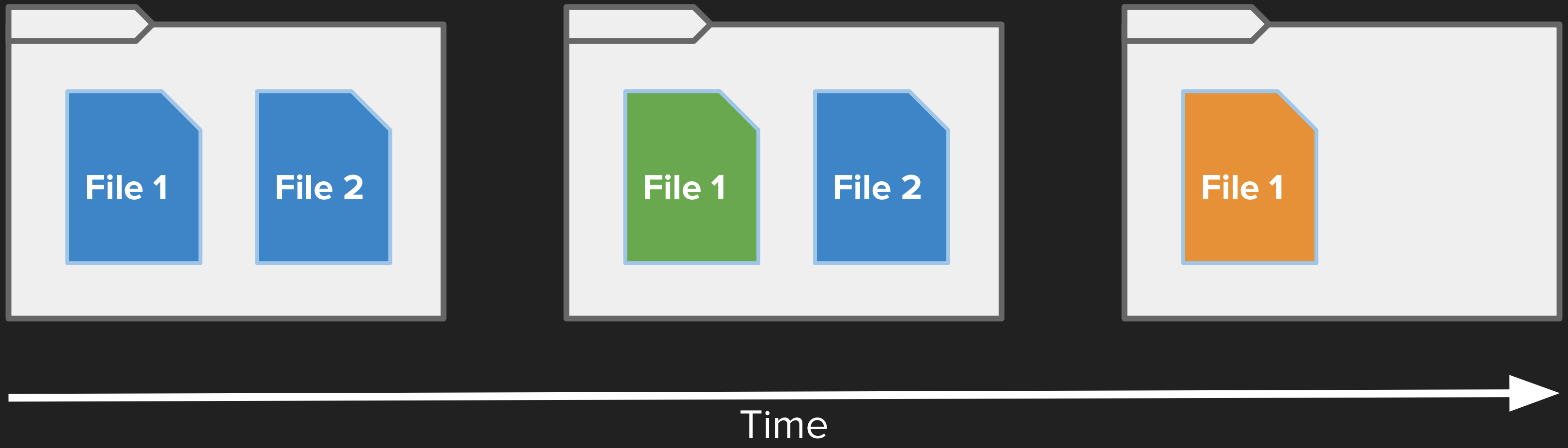
Online git hosting services

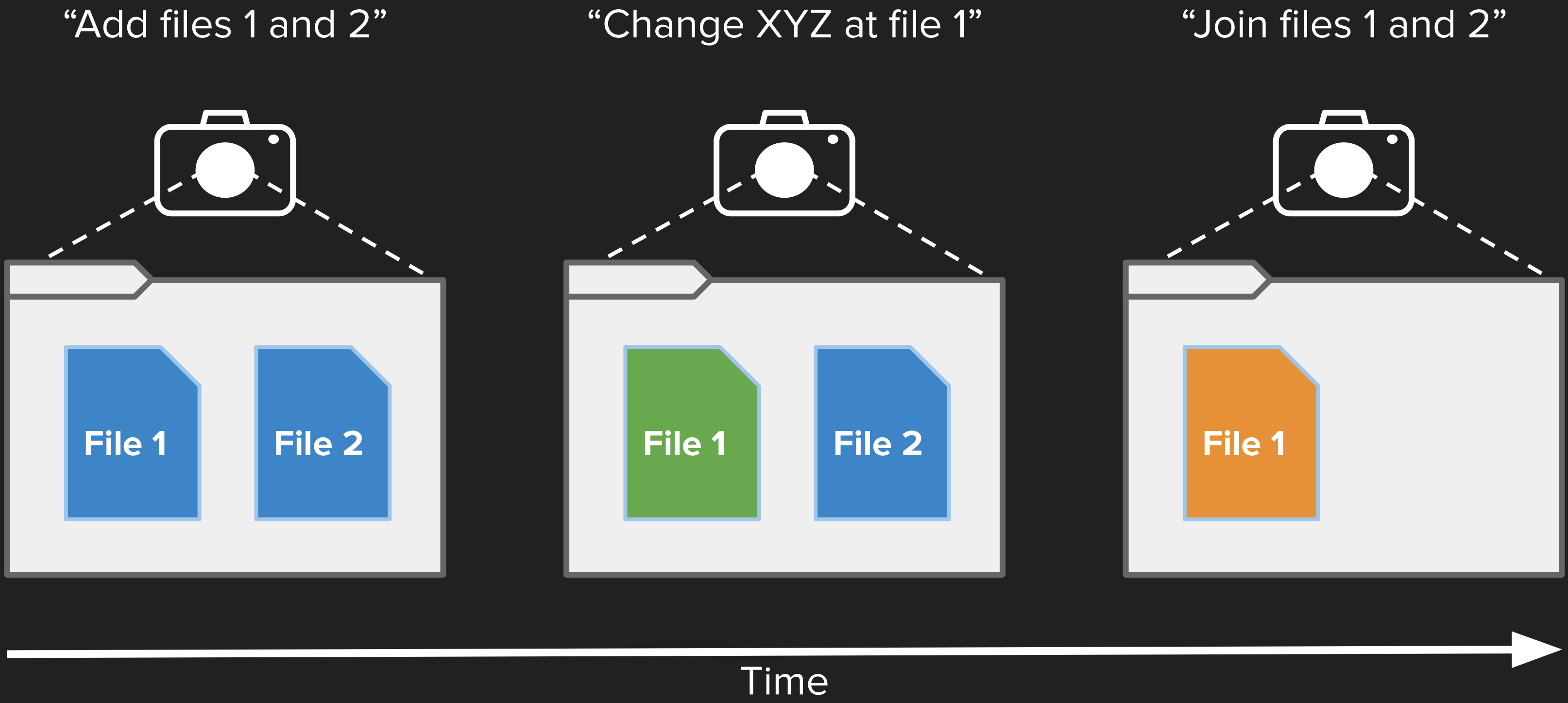
GitLab

Version control system (software you run on your machine)

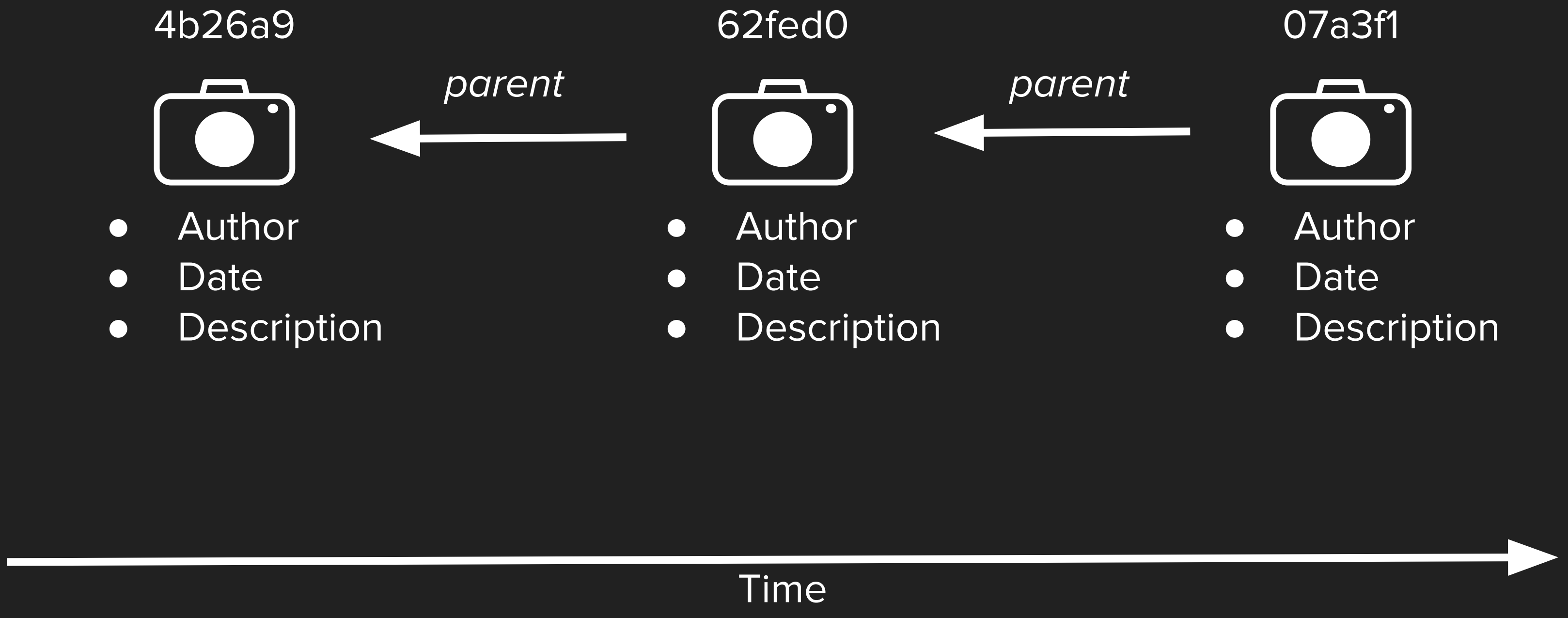

File 1
File 2
Time


File 1
File 2
File 1
File 2
Time


File 1
File 2
File 1
File 2
File 1
Time


“Add files 1 and 2”
“Change XYZ at file 1”
“Join files 1 and 2”
File 1
File 2
File 1
File 2
File 1
Time

Pictures: ***Commits***



- Author
- Date
- Description

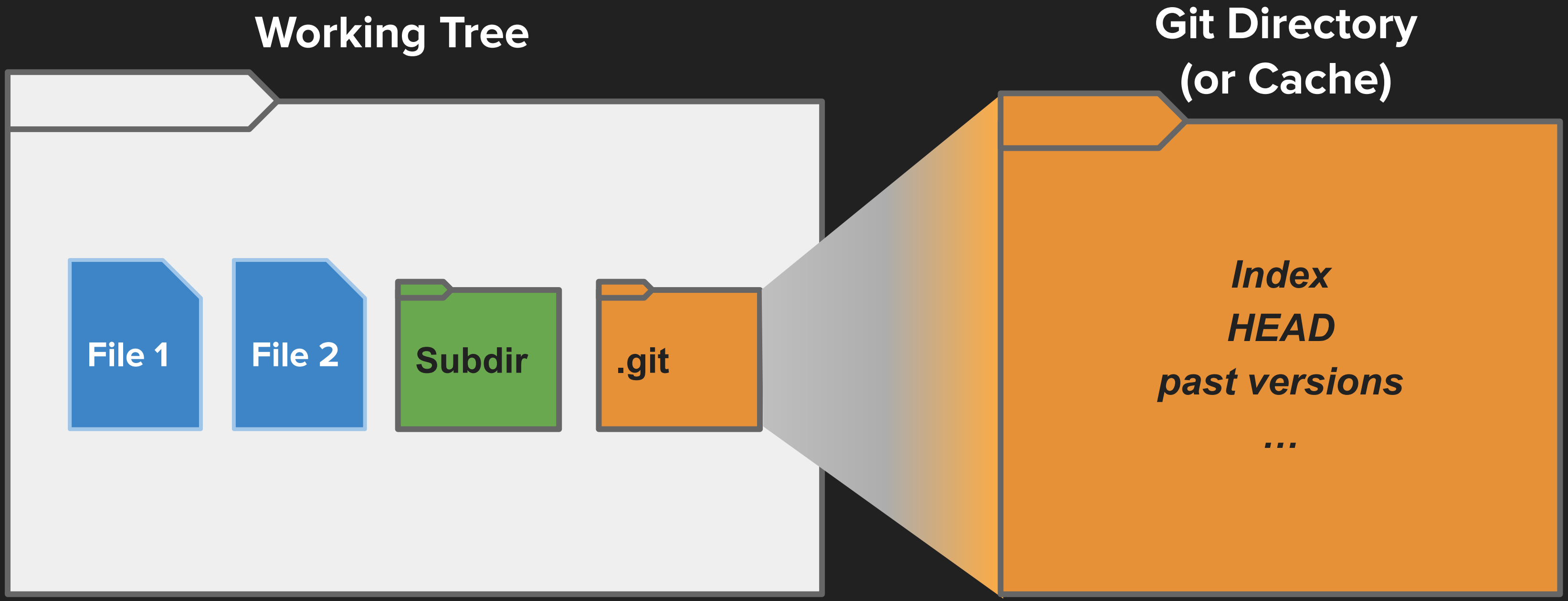
- Author
- Date
- Description

- Author
- Date
- Description

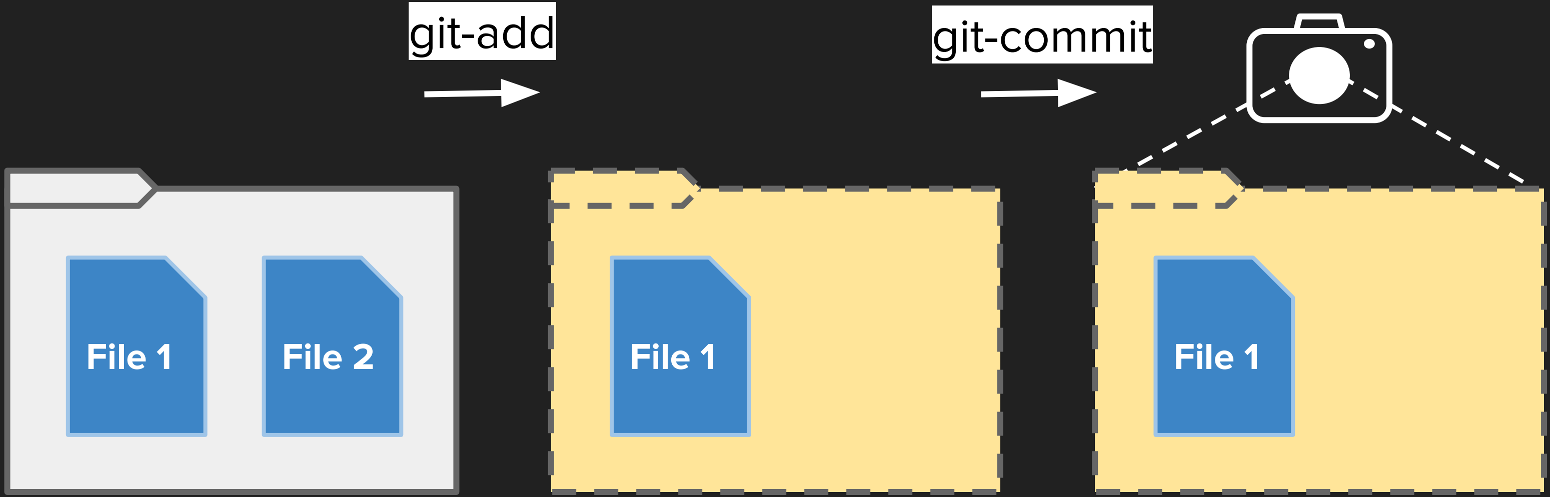
What can I do with that?

- What changed between version X and Y?
- What does files A and B looked liked a month ago?
- Who last changed line L? Why?
- Which change introduced bug #21? Let's revert that.
- etc.

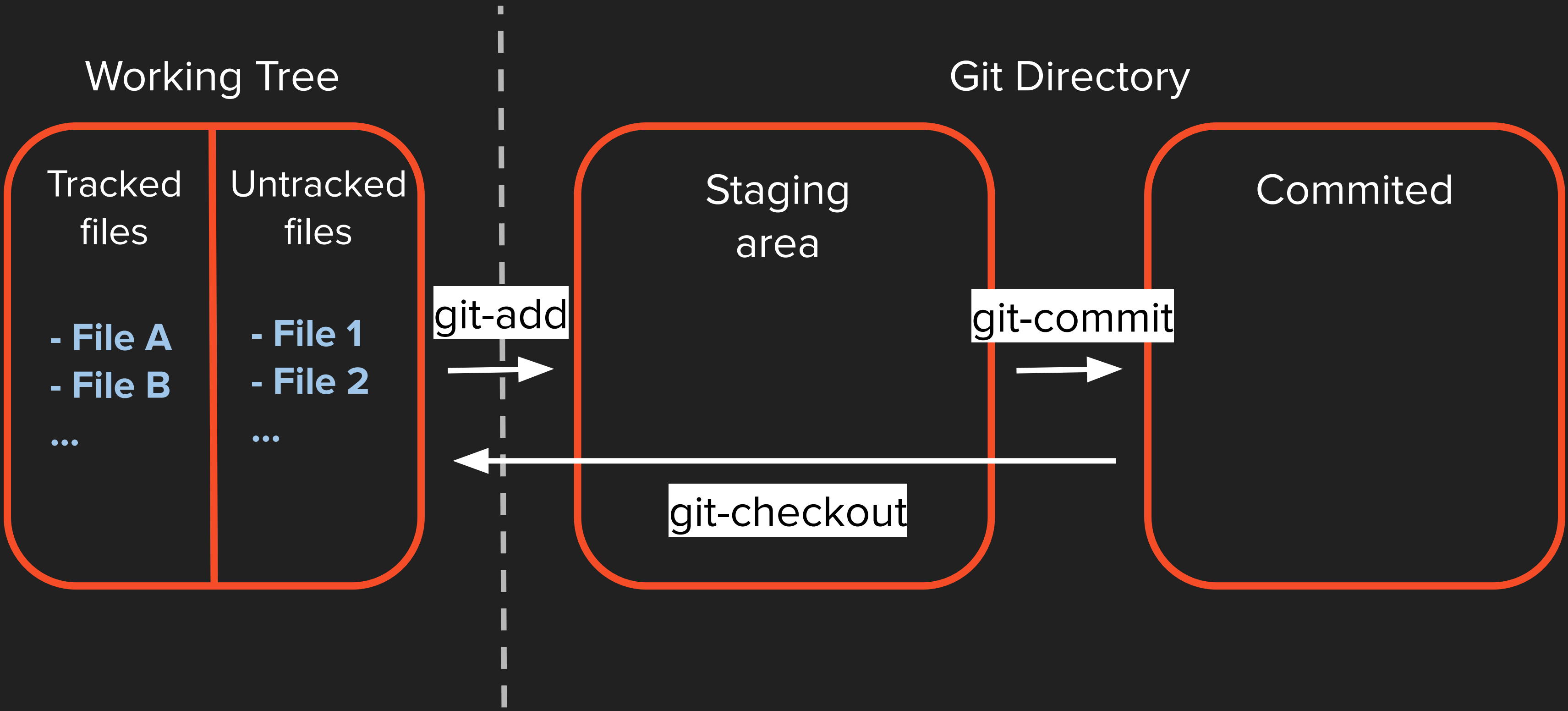
Repository


Working Tree
Git Directory (or Cache)
File 1
File 2
Subdir
.git
Index
HEAD
past versions
...

Adding and committing

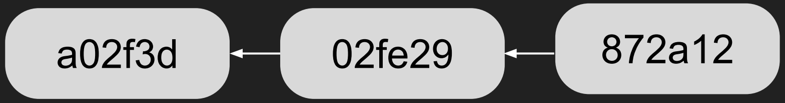


Adding and committing

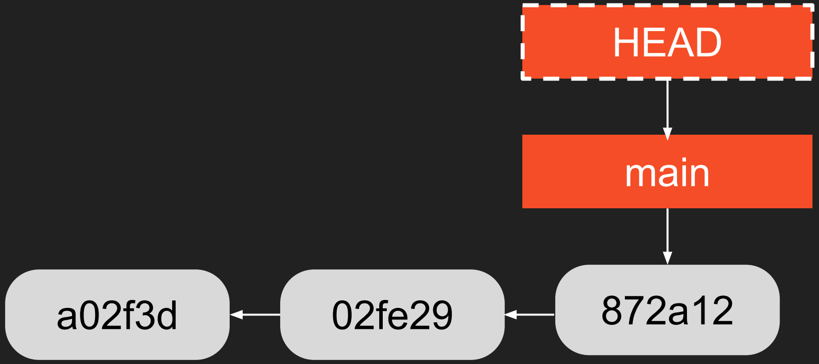


References

References

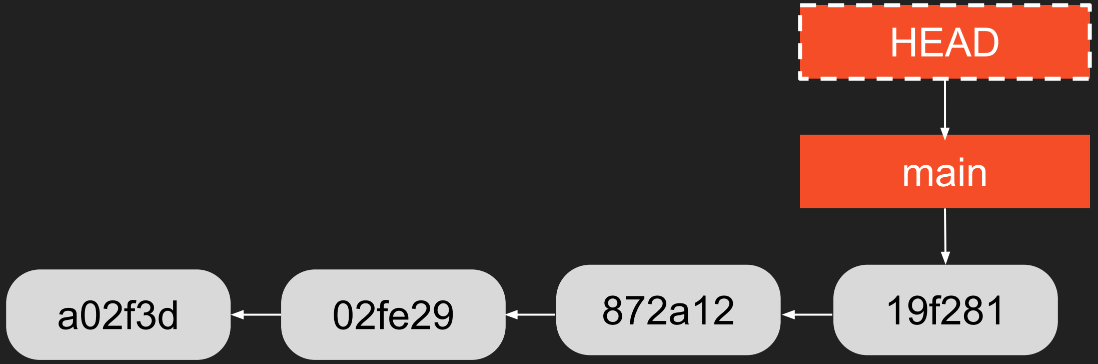


References



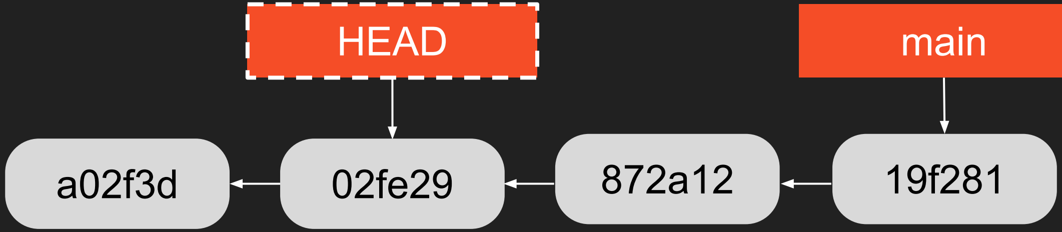
Referências

```
$ git commit
```



References

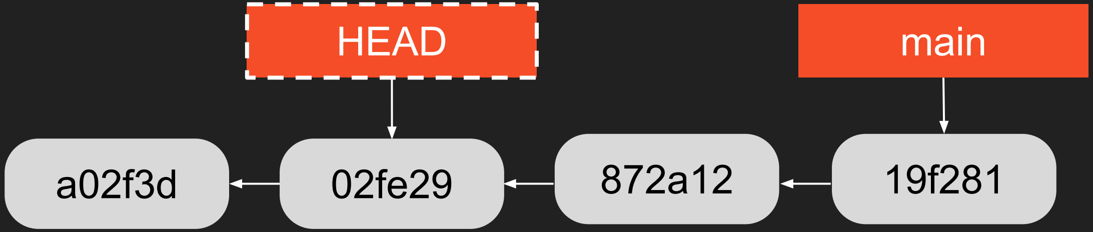
```
$ git checkout 02fe29
```



References

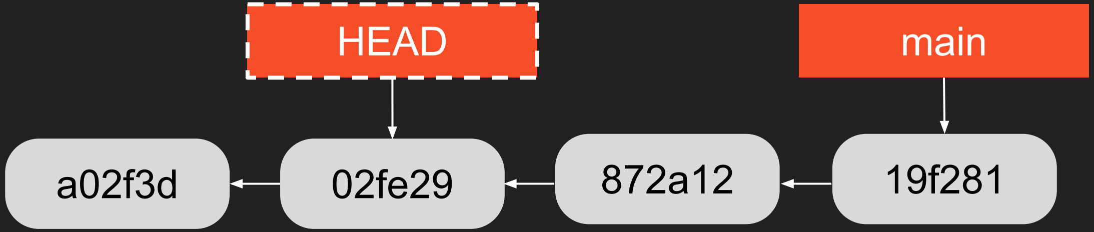
```
$ git checkout 02fe29
```

You are in "detached HEAD" state.



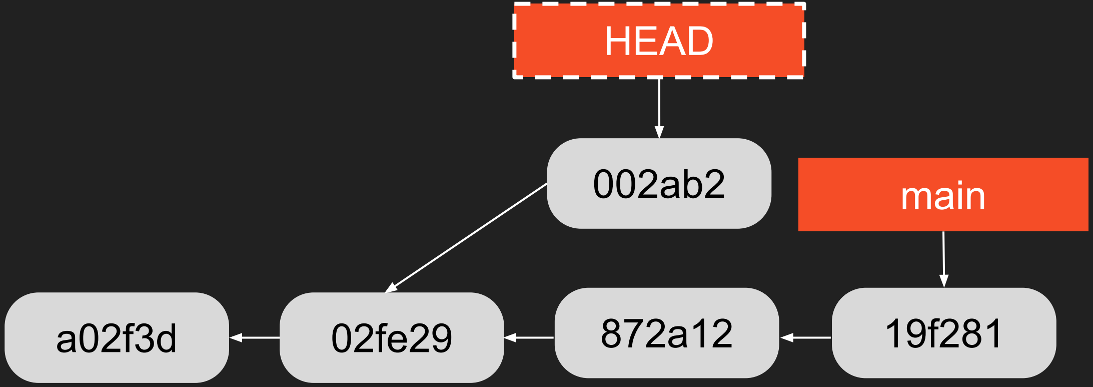
References

```
$ git commit
```



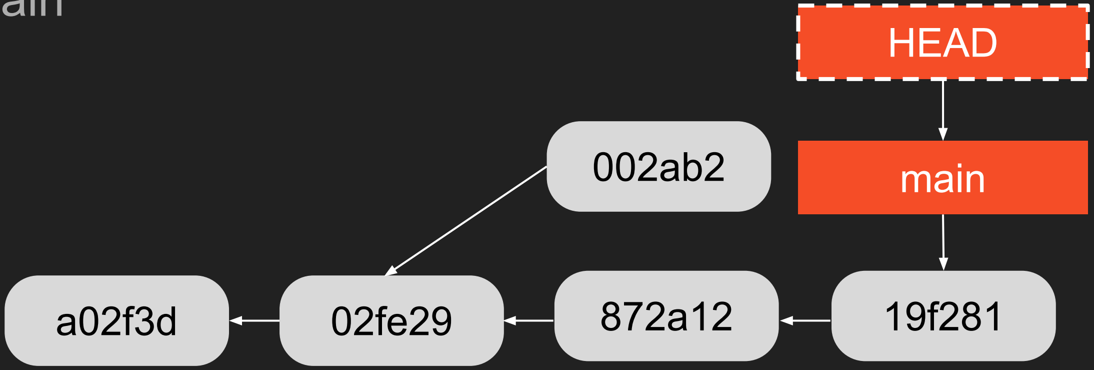
References

```
$ git commit
```



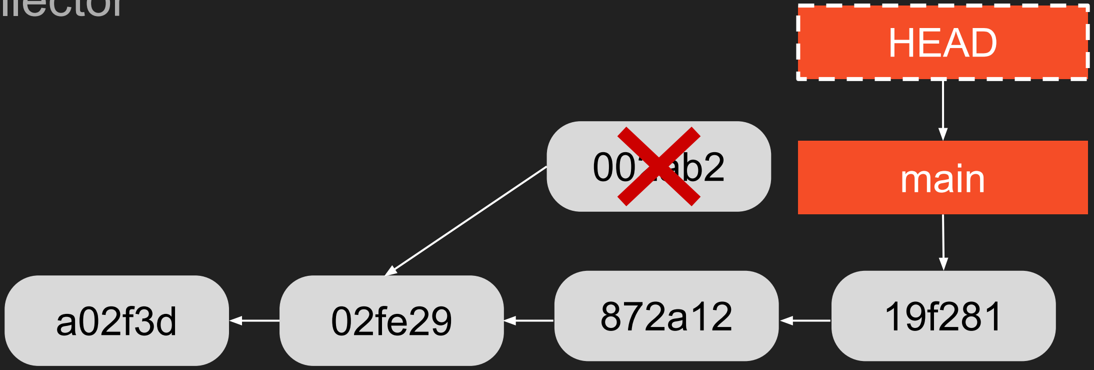
References

```
$ git checkout main
```



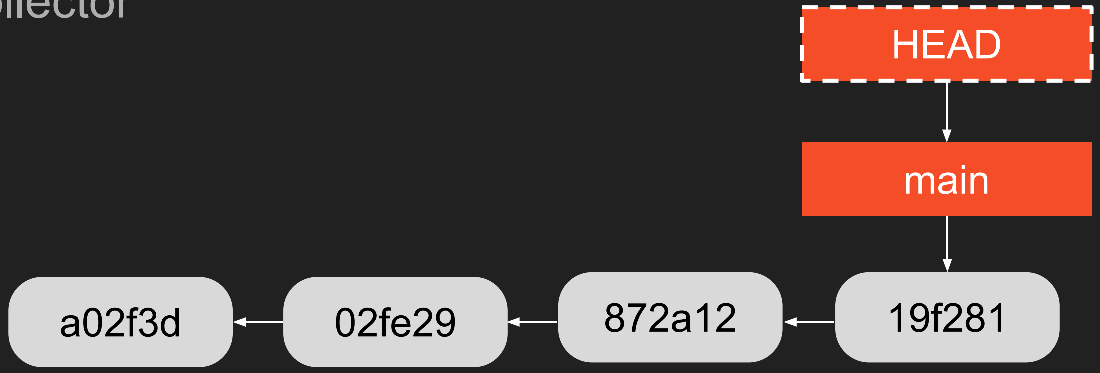
Referências

Git's garbage collector



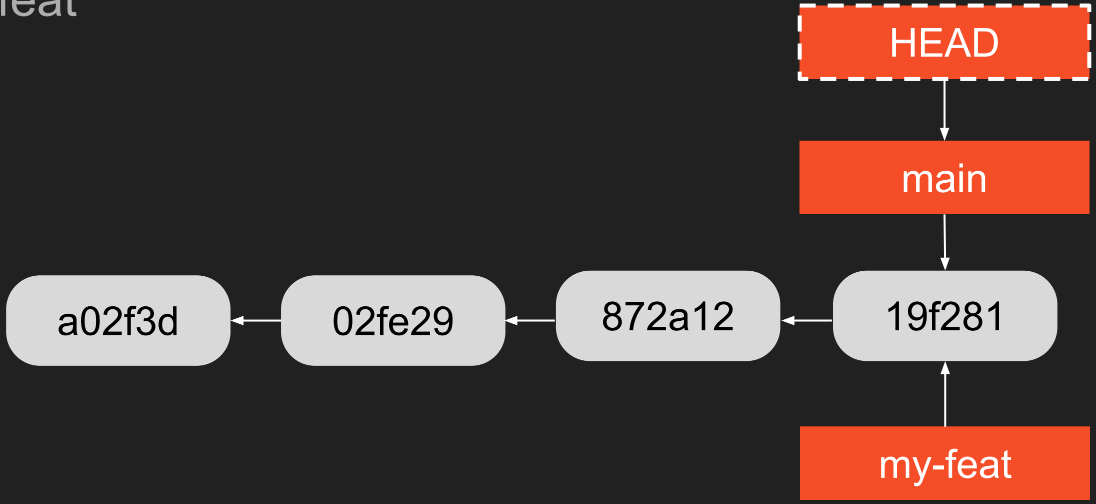
References

Git’s garbage collector



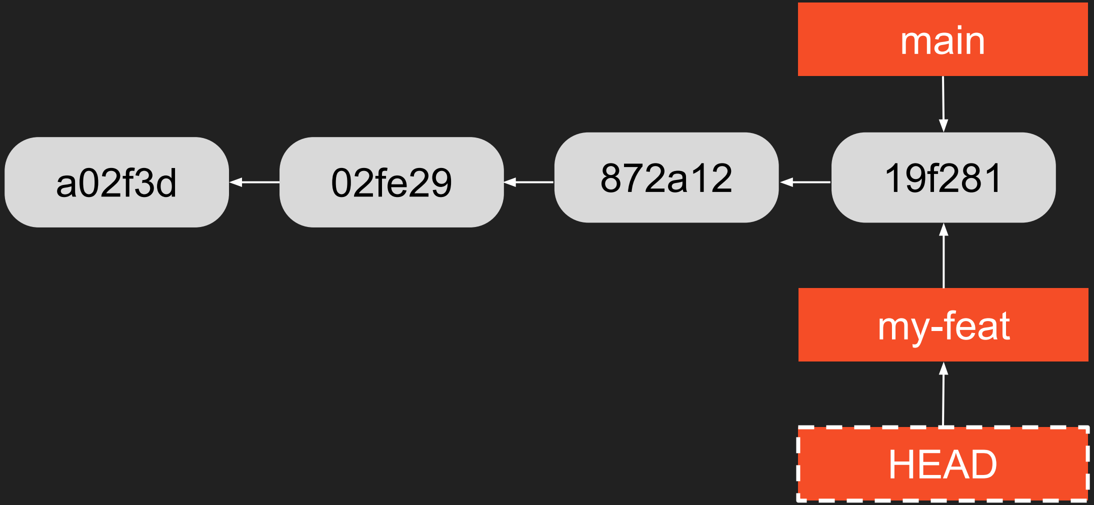
References

```
$ git branch my-feat
```



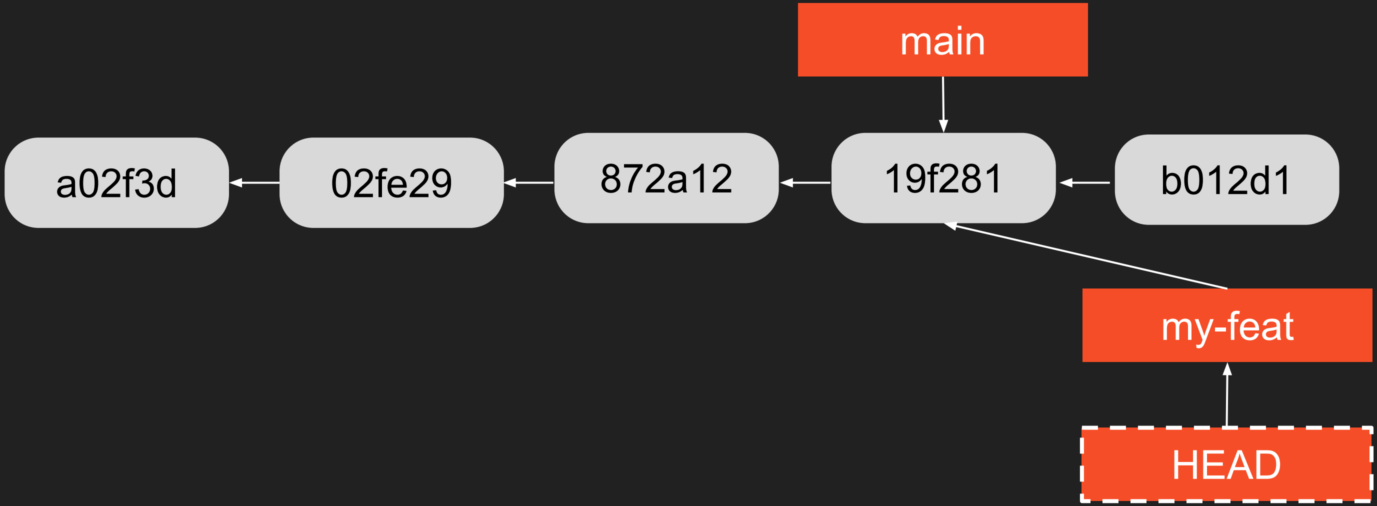
References

```
$ git checkout my-feat
```



References

```
$ git commit
```



Let’s jump to the terminal

Best commit practices

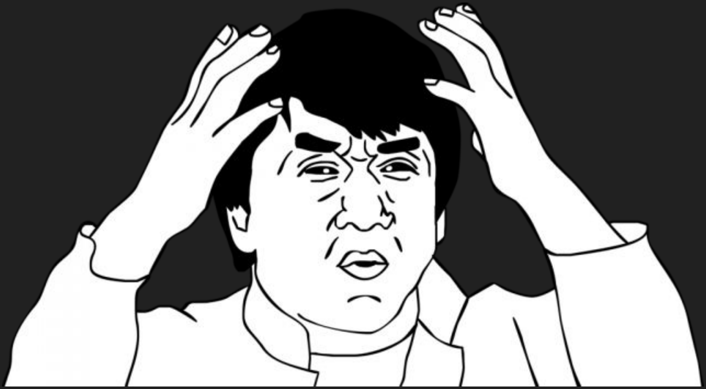
Sounds like a good idea…

```
$ vim file.c

$ git add .

$ git commit -m “bla”
```

… until you need to recap what you did.



https://blog.tiagopariz.com/wp-content/uploads/2018/10/jackie-chan-meme.png

```
commit 5e2faaea31dceaad23a3331cd9ad20afad6c719b
Author: joh123 <>

    Minor changes

commit ae95825114ccac3a996251bd1de4da5e83c95172
Author: joh123 <>

    Fixed some bugs

commit 6e639b014f82521138025091fac660cd2474b7a0
Author: joh123 <>

    Now it compiles!

commit f717b52ddd1e95dbbdd5a1476051b842468cb195
Author: jonh123 <>

    Actually it didn't. NOW IT DOES

commit 90384aec06d44a5d40d9c299f082334503943b32
Author: john123 <>

    Update README
```

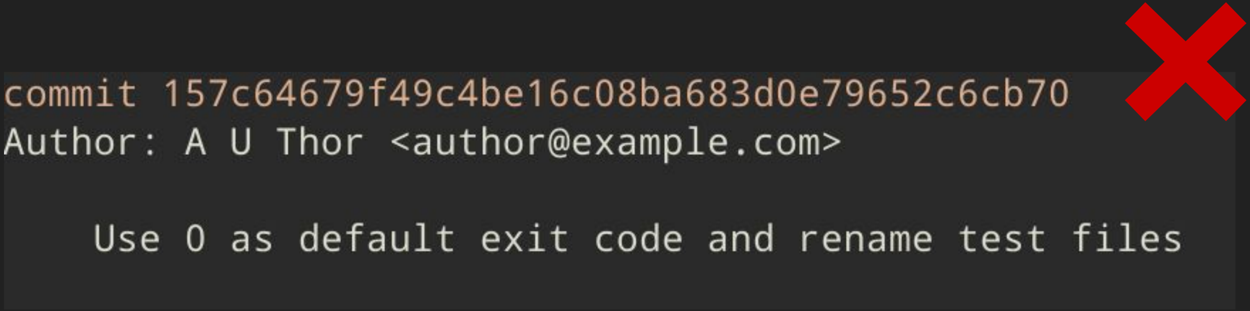
Dissecting a good commit

1. **Unrelated** changes belong to **separate** commits.
2. **Avoid committing** incomplete work.
3. Invest on writing **informative** commit messages.

1. Unrelated changes belong to separate commits.

- Easier to review (better reviews → better codebase)
- Easier to **revert**
- Easier to **merge** with other changes and branches

1. Unrelated changes belong to separate commits.



```
commit 157c64679f49c4be16c08ba683d0e79652c6cb70
Author: A U Thor <author@example.com>

    Use 0 as default exit code and rename test files
```

2. Avoid committing incomplete work

- Commits should be justified by themselves (although there can be dependencies among them).
- It’s nice to have each commit compilable and passing the tests (easier to find bugs and behavior changes)

3. Invest on writing informative commit messages

"a well-crafted Git commit message is the best way to communicate context about a change to fellow developers (and indeed to [your future self]). A diff will tell you **what changed**, but only the commit message can **properly tell you why**."

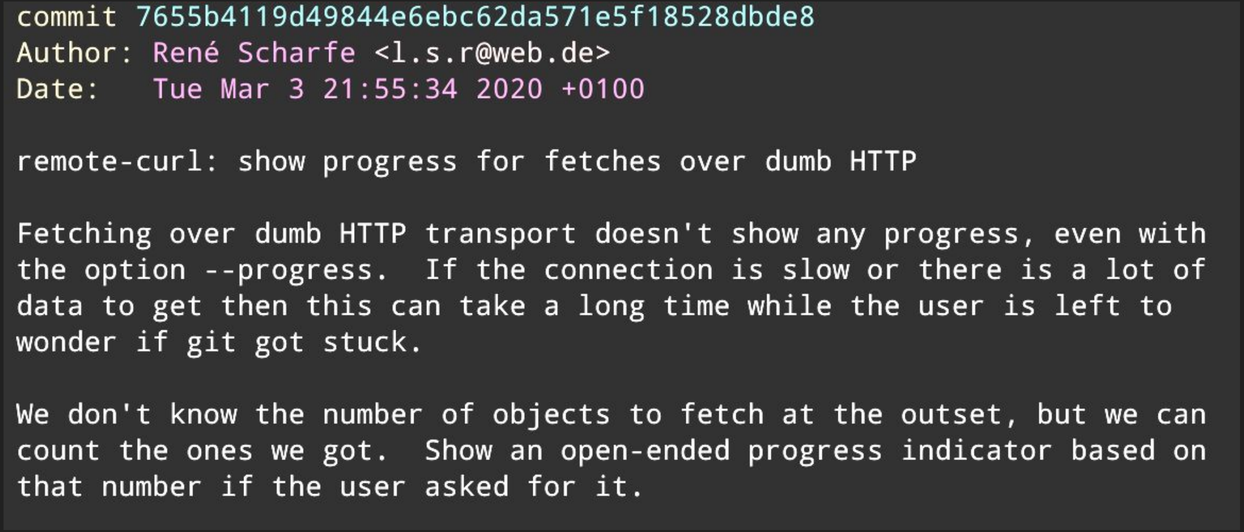
- Chris Beams (https://chris.beams.io/posts/git-commit/)

3. Invest on writing informative commit messages

- **Describe the problem**
  What is not good in the codebase?
- **Justify how your changes solve the problem**
  Why the codebase state is better after this patch?
- **Discarded alternatives [optional]**
  Are there other ways to implement this? Why this was chosen?

Tip:
git commit -v ou
git config --global commit.verbose true

Example



```
commit 7655b4119d49844e6ebc62da571e5f18528dbde8
Author: René Scharfe <l.s.r@web.de>
Date:   Tue Mar 3 21:55:34 2020 +0100

remote-curl: show progress for fetches over dumb HTTP

Fetching over dumb HTTP transport doesn't show any progress, even with
the option --progress.  If the connection is slow or there is a lot of
data to get then this can take a long time while the user is left to
wonder if git got stuck.

We don't know the number of objects to fetch at the outset, but we can
count the ones we got.  Show an open-ended progress indicator based on
that number if the user asked for it.
```

Remote

- A *remote* repository relative to the same project.
    - Backups in the cloud
    - Share code
    - Collaborative development
    - etc.

Remote

- Clone from remote:
    - **$ git clone** https://github.com/me/repo.git
- You may have multiple remotes inside the same repo
    - **$ git remote add** john https://github.com/john/repo.git
- Update remote branch:
    - **$ git push <remote> <branch>**
- Update local branch:
    - **$ git pull <remote> <branch>**

Extra tips

- Exercising git: https://github.com/Gazler/githug
- Visualizing git: https://git-school.github.io/visualizing-git/
- $ git help glossary
- $ git help revisions
- $ git blame / git log -S
- “man git ...” is your friend :)

References

1. **Pro Git**, Scott Chacon and Ben Straub: https://git-scm.com/book/en/v2
2. **Git Docs:** https://git-scm.com/docs/
3. **How to Write a Git Commit Message**, Criss Beans: https://chris.beams.io/posts/git-commit/
4. **Developer Tip: Keep Your Commits "Atomic"**, Sean Patterson: https://www.freshconsulting.com/atomic-commits/