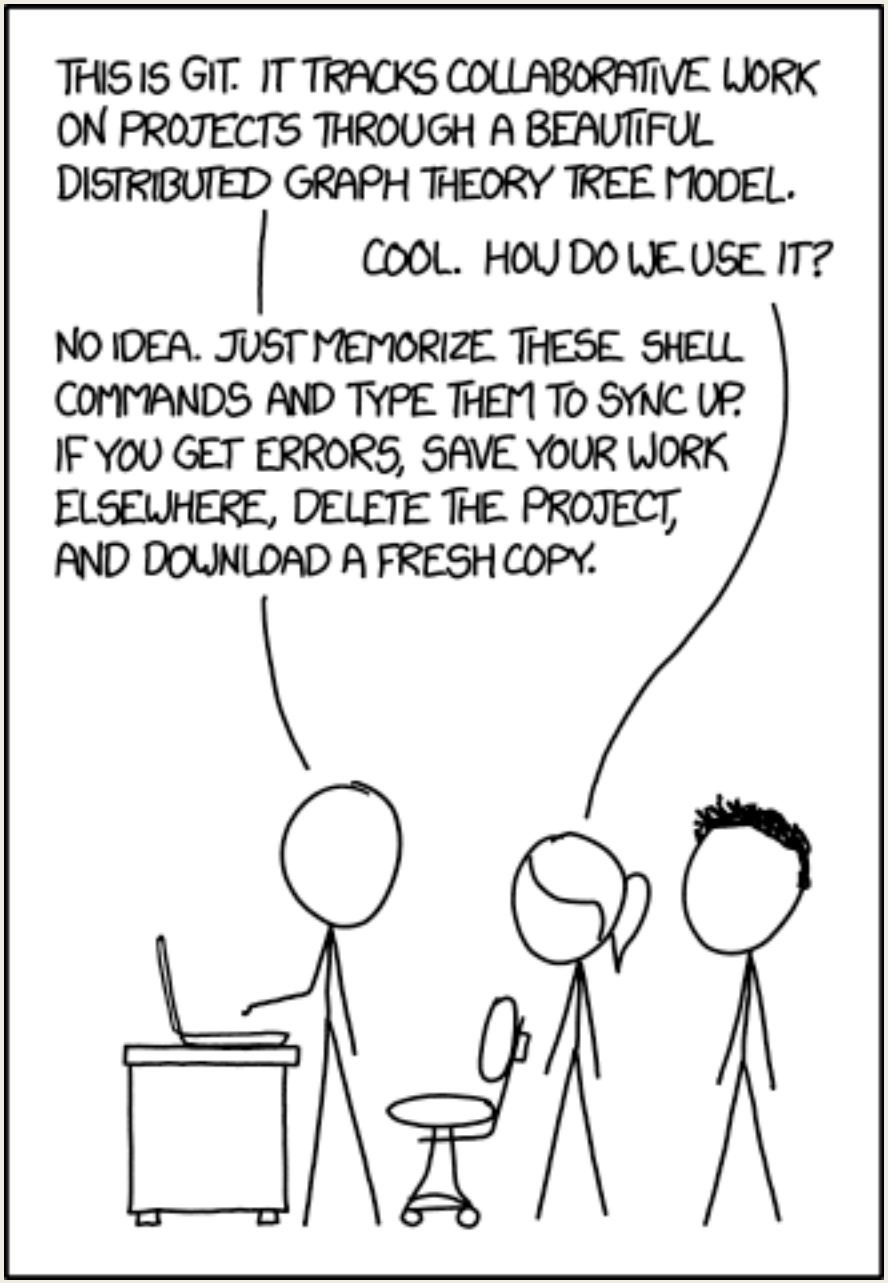
ML Fundamentals

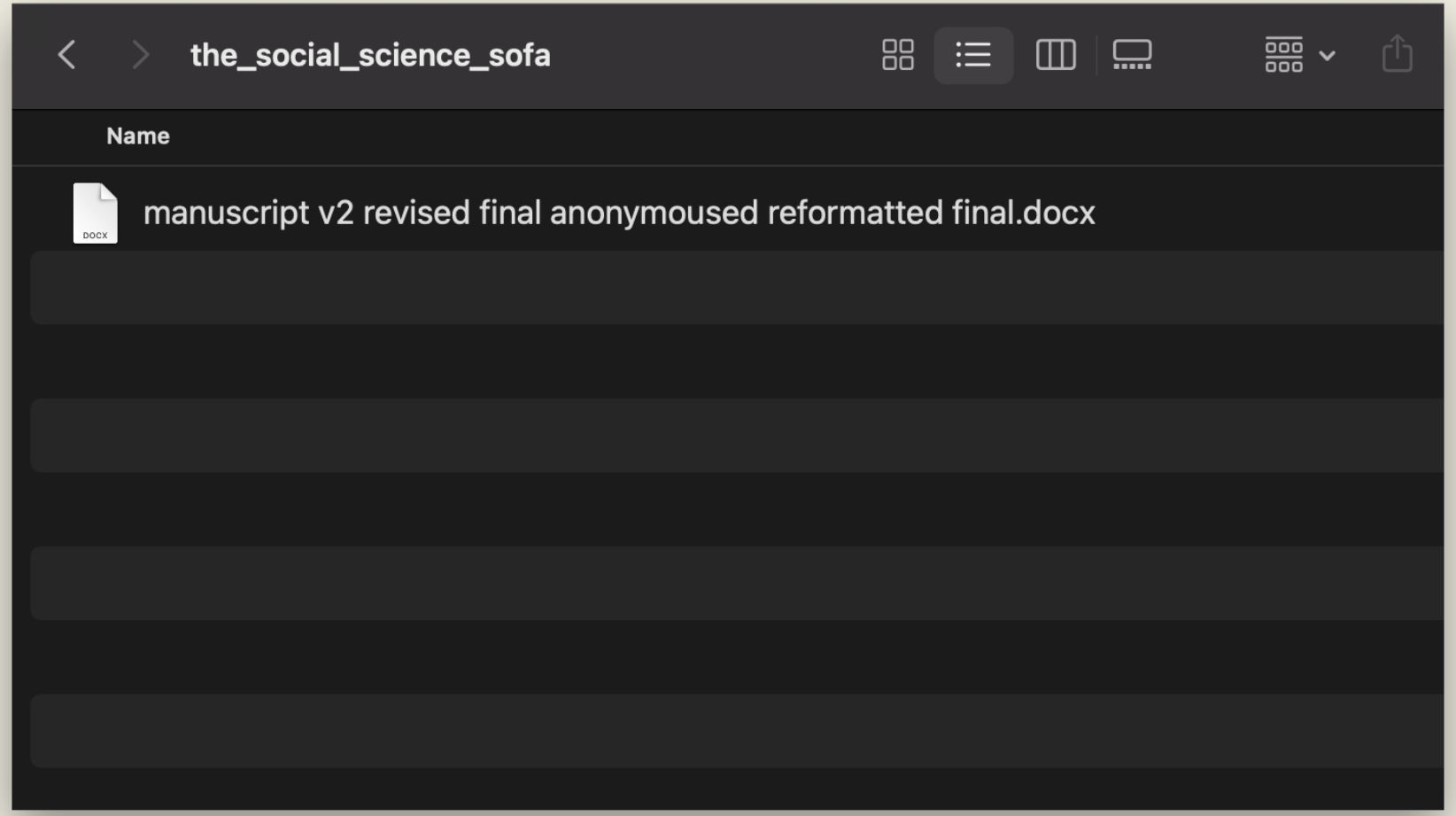
Instituto Balseiro

Instituto Balseiro
16/08/2022

Special session

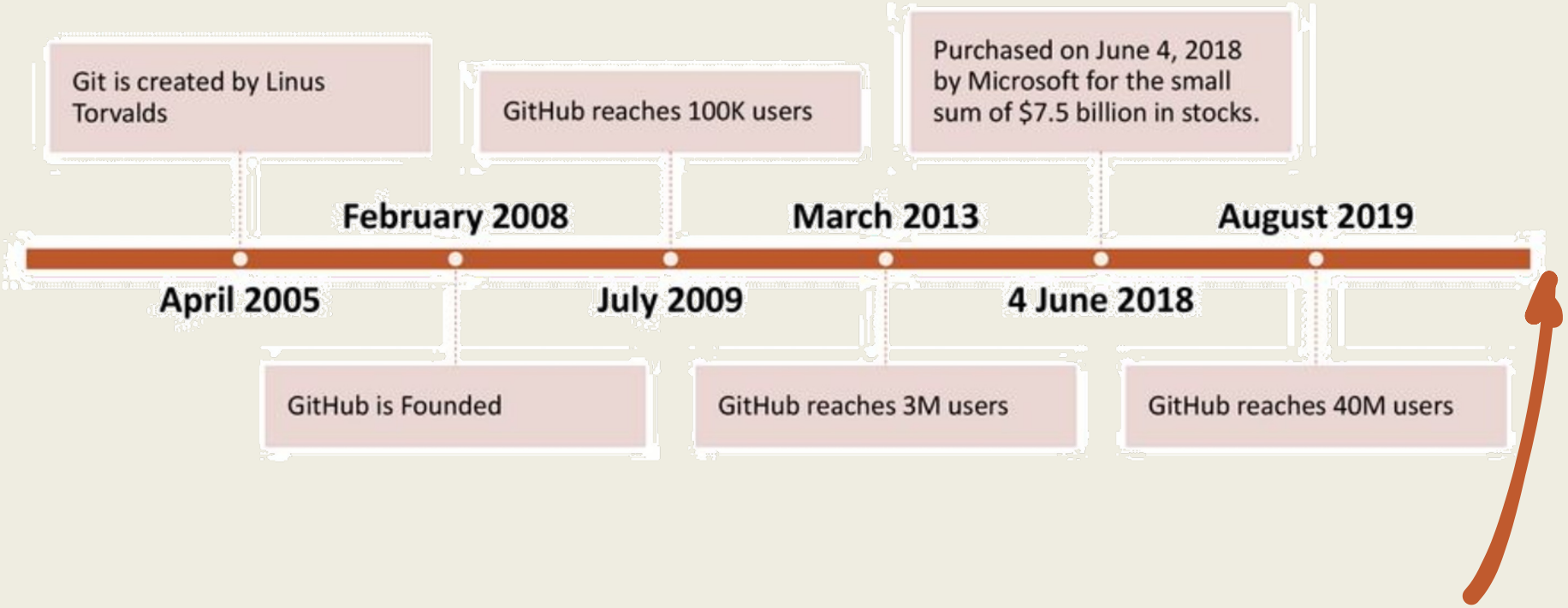

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

The filename nightmare



A penguin named Linus

Git & GitHub timeline



83 million users (as of June 2022)

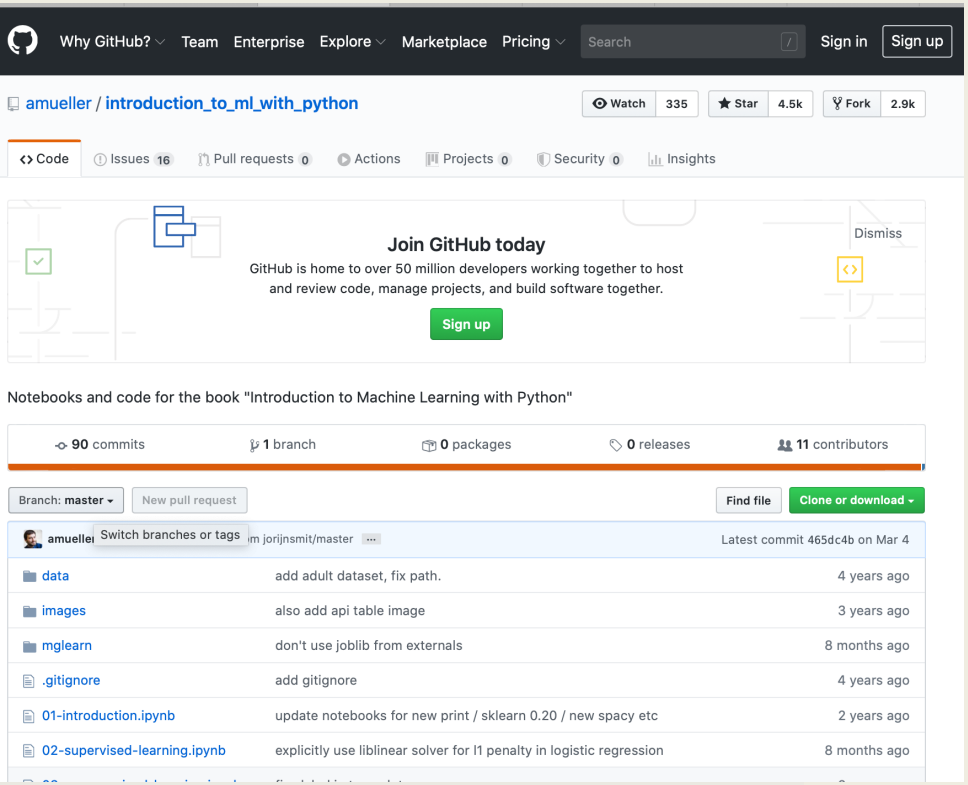
Why git, why GitHub?... why? why?!

What's GIT? It's 'the' distributed version control system

- Performance
- Security
  - Integrity focused, SHA1-based
- Flexibility
  - Supports nonlinear workflow
- Open-source ☺

What's GitHub? It's 'the' git-based hosting service

- Popularity & Community
- Noob-friendly
- Not open source ☹ (but mostly free as in beer)
  - (GitLab is! :D)
- Many (if not most) ML projects are hosted here



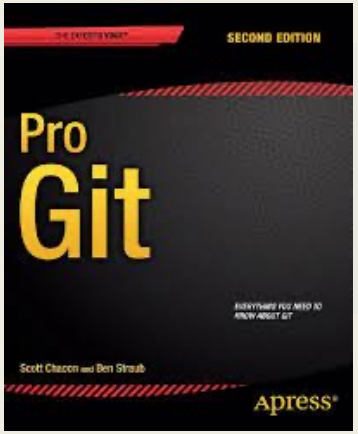
https://github.com/amueller/introduction_to_ml_with_python

https://github.com/ageron/handson-ml2

Git es un sistema de control de version distribuido

Reference Manual

The official and comprehensive **man pages** that are included in the Git package itself.



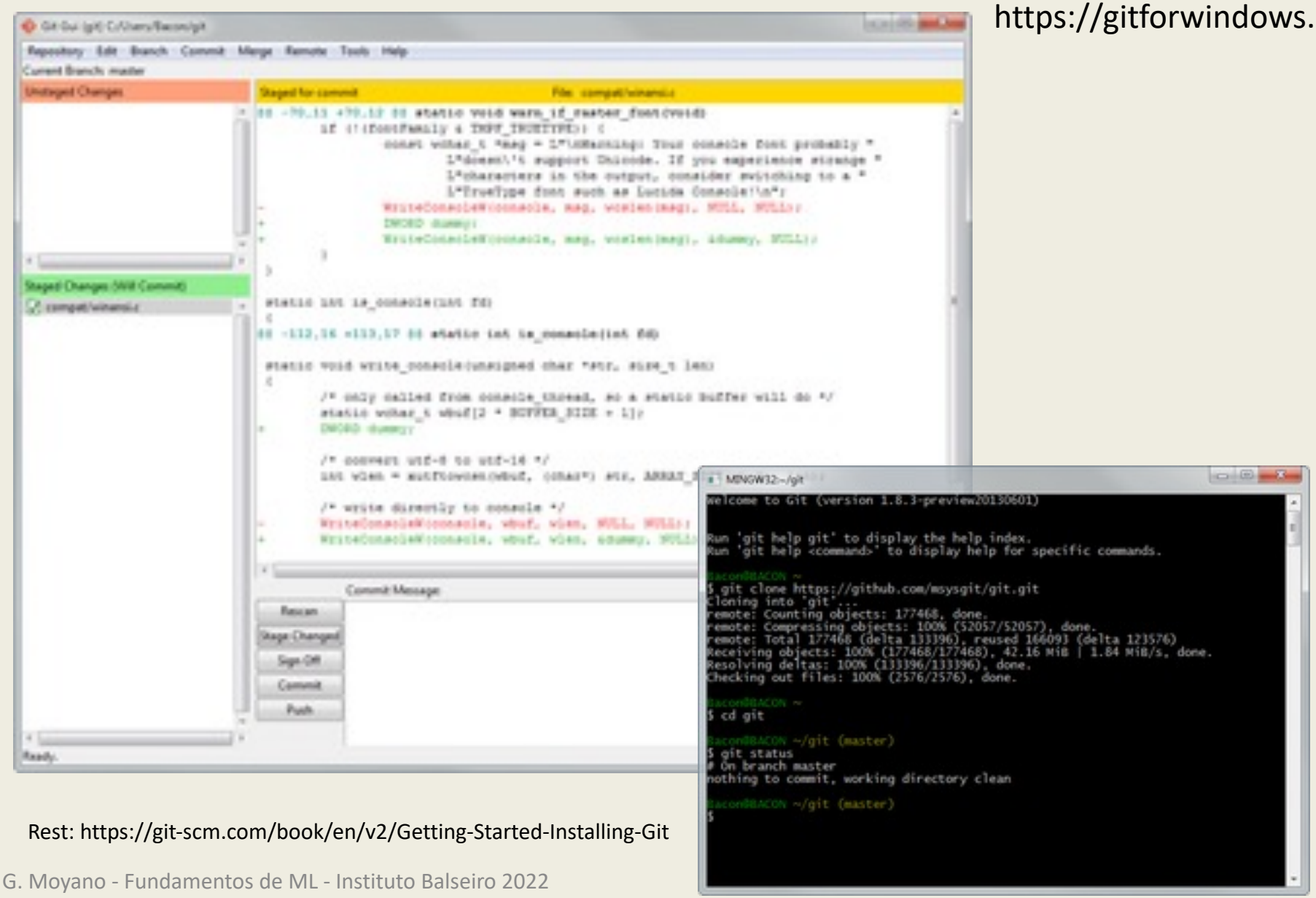
Pro Git

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

In course GitHub repo

Instalación en Windows

https://gitforwindows.org/

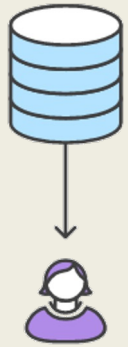


Rest: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

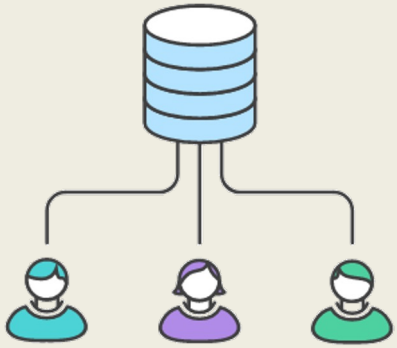
Some git scenarios

- **Local**: Single user, local.
  - backup
  - trial and error
  - flexibility

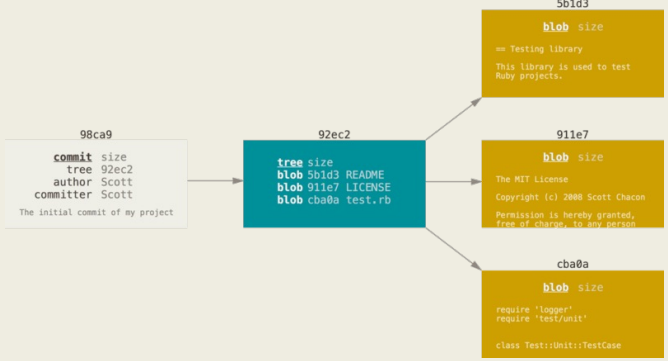
- **Remote**: Single user, local and remote.
  - remote backup
  - work with 3rd party code
  - sharing purposes



- **Distributed:** More than user, each working local and also collaborating on remote.
  - collaboration
  - E.g. writing a paper



Commit!



Commit often! and in logical units!

Commit tree

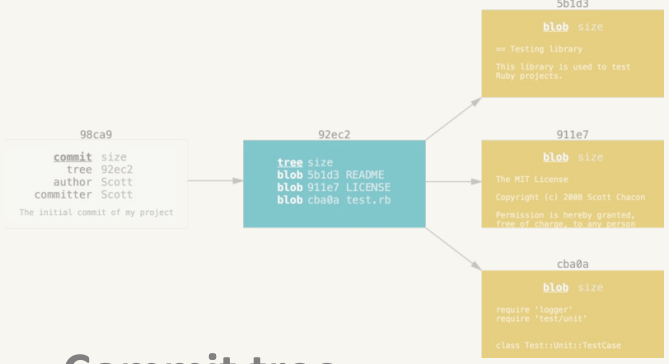
- Snapshop of chosen files
- References
- Plus metadata
  - author, date, time
  - message
  - SHA-1 hashes
    - own
    - parent's
    - blob's
  - etc.

GitHub:

Shell:

```
(base) workmac:guias-profes moyano$ git log --pretty=oneline
80bb9ded1bb5054c619da3644f179126c3f87b90 (HEAD -> main, origin/main) recover guia_01-git_github.tex
ba84e19cf46c8a9d4f4f23d3dfe56bcf6d34f10f finish merge
6572c328ceebadb68c2af3691a183953f4f91b5d commit aux files
6baa57a0a955f755e38b351a1ef688a00a7ab33e nuevos problemas en guia 02
b8054a15ba29a022bf6d0a10eef92a62e7050560 add paper.cls
c287286c84b6e33d1a9c30c9bea0d04a93003751 update schedule
df3ad07f563f5ec01bfe2d0931a59b8b166b8764 update first two practices
28393621d484762d0e46087d91c2b59d3d1e7fcb add guias folder with practice template
```

Commit!

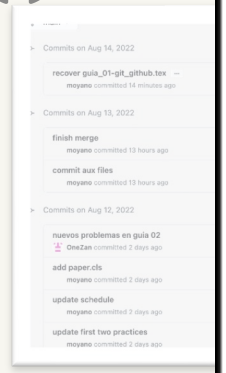


Commit tree

- Snapshop of chosen files

Commit often! and in logical units

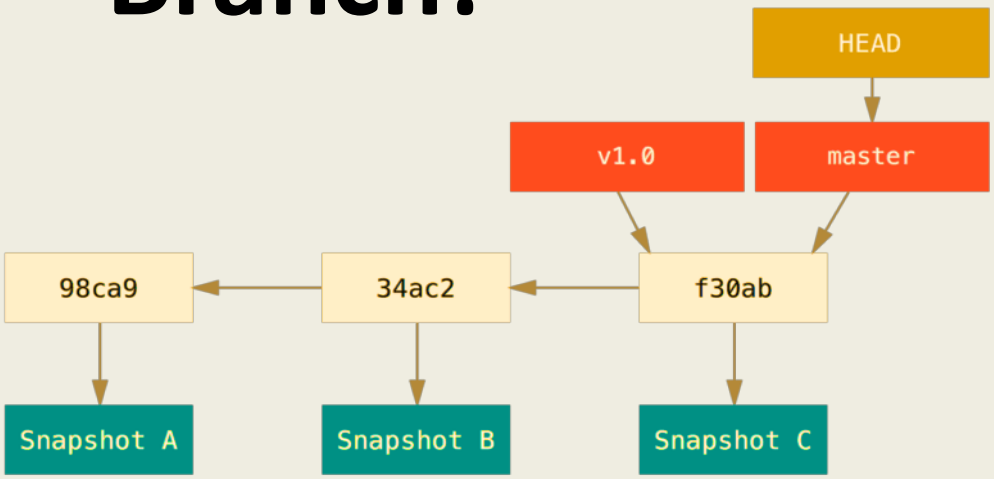
GitHub



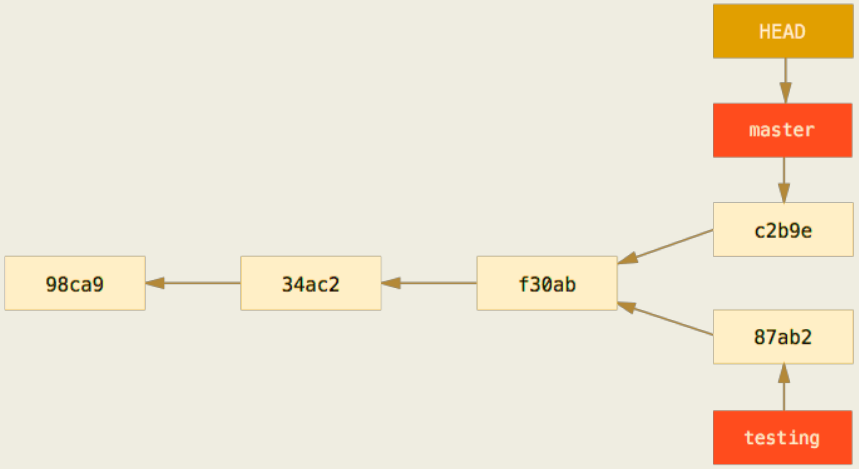
Shell:

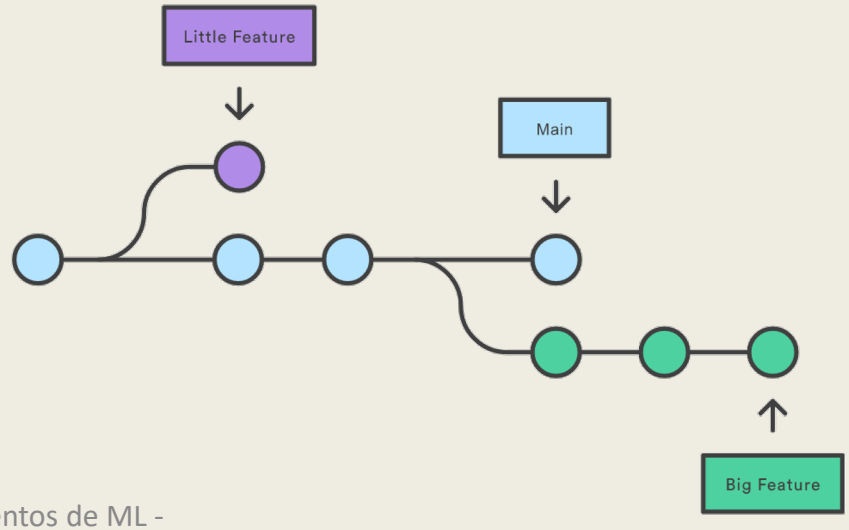
```
(base) work
80bb9ded1bb                                                    ithub.tex
ba84e19cf4
6572c328cee
6baa57a0a95
b8054a15ba29a022bf6d0a10eef92a62e7050560 add paper.cls
c287286c84b6e33d1a9c30c9bea0d04a93003751 update schedule
df3ad07f563f5ec01bfe2d0931a59b8b166b8764 update first two practices
28393621d484762d0e46087d91c2b59d3d1e7fcb add guias folder with practice template
```

Branch!

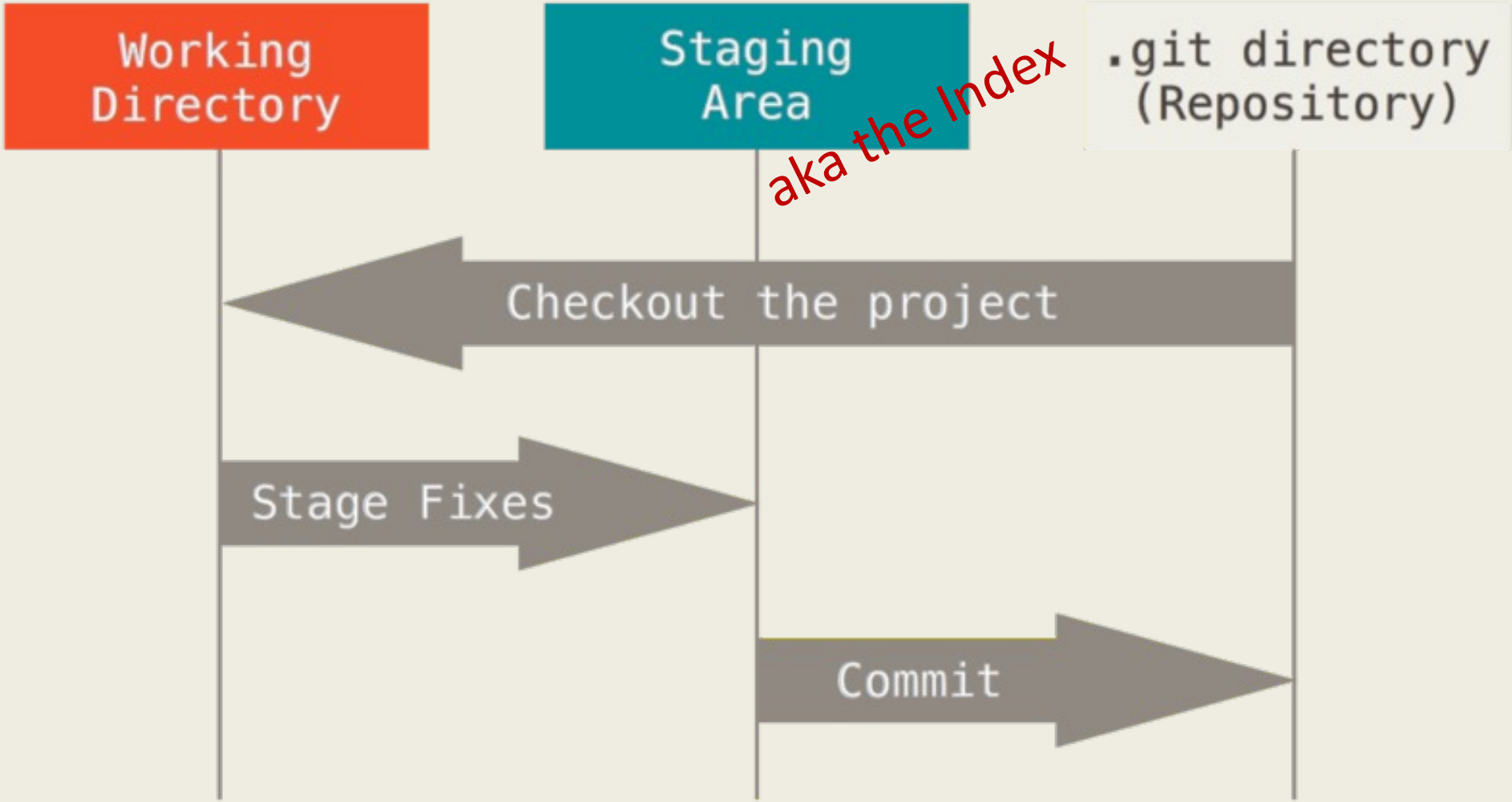


- Just a reference (think of it as a little flag) to a selected commit (& parents)
- Used to "branch" a commit 'time line'
- Gets carried up when commiting
- As many as we want (really cheap!)
- HEAD is a special reference to the checkout commit

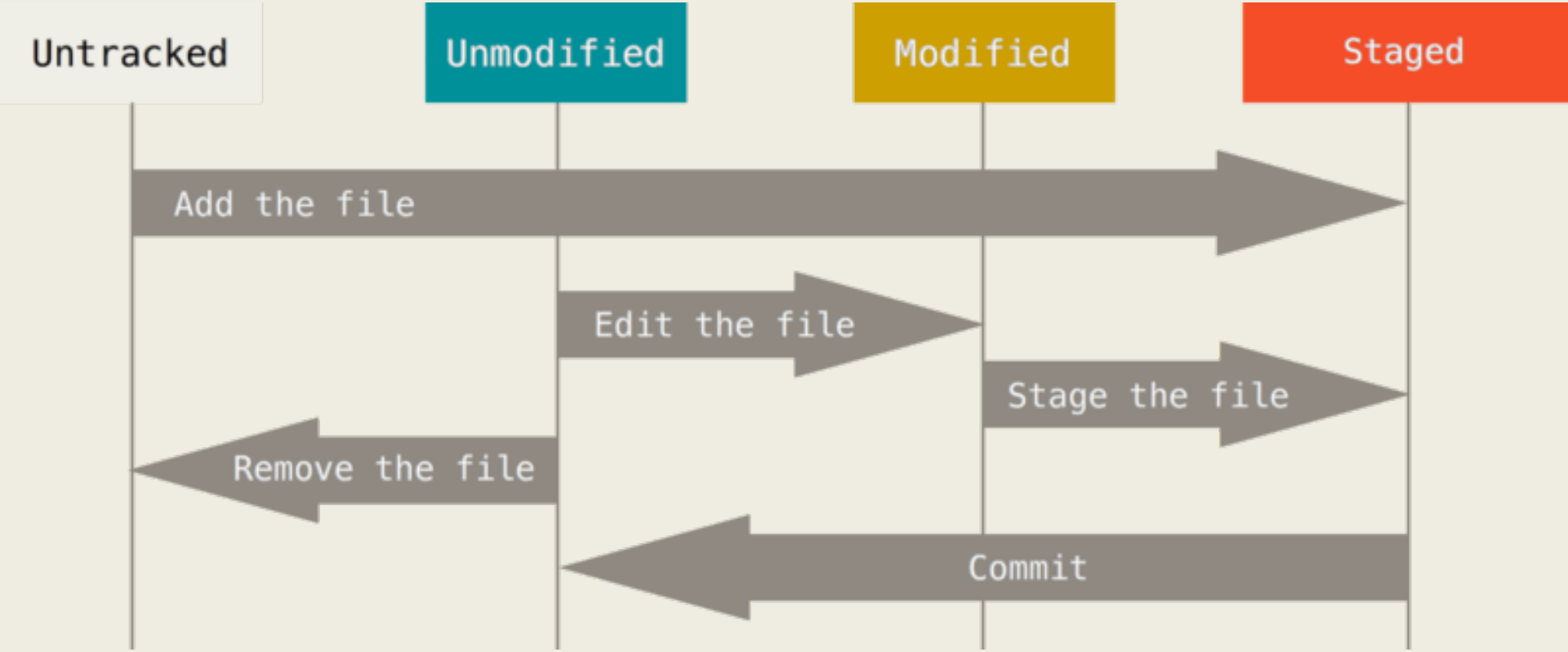




Local git workflow



File life cycle



Config

Configura la información del usuario para todos los respositorios locales

```
$ git config --global user.name "[name]"
```

Establece el nombre que estará asociado a tus commits

```
$ git config --global user.email "[email address]"
```

Establece el e-mail que estará asociado a sus commits

Ignore

- To ensure that certain files (not already tracked by Git) remain untracked.
- There is a practical side, and a security side.
- How? put *globs* in hidden .gitignore file in project folder*
  - globs: **shell pattern matching**, as in .* or *.txt

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

For more see: Ignoring files and in the official git docs and man gitignore

.

* It's possible to have multiple .gitignore files

Create repositories

Inicializa un nuevo repositorio u obtiene uno de una URL existente

```
$ git init [project-name]
```

Crea un nuevo repositorio local con el nombre especificado

```
$ git clone [url]
```

Descarga un proyecto y toda su historial de versiones

In most cases, the cloned remote repo remains with the default name **'origin'**

Make changes

Revisa cambios y crea un commit

```
$ git status
```

Enumera todos los archivos nuevos o modificados de los cuales se van a guardar cambios

```
$ git diff
```

Muestra las diferencias entre archivos que no se han enviado aún al área de espera

```
$ git add [file]
```

Guarda el estado del archivo en preparación para realizar un commit

```
$ git commit -m"[descriptive message]"
```

Registra los cambios del archivo permanentemente en el historial de versiones

diffs

```
Changes from main to working tree
1 file changed, 1 insertion(+), 1 deletion(-)
guias-profes/guia_01-git_github.aux | 2 +-

modified   guias-profes/guia_01-git_github.aux
@@ -5,7 +5,7 @@
 \@nameuse{es@quoting}
 \babel@aux{spanish}{}
 \@writefile{toc}{\contentsline {section}{\numberline
{1}Questions}{1}}
-\@writefile{toc}{\contentsline {section}{\numberline
{2}Problems}{1}}
+\@writefile{toc}{\contentsline {section}{\numberline
{2}Exercises}{1}}
 \@writefile{toc}{\contentsline {subsection}{\numberline
{2.1}Local}{1}}
 \@writefile{toc}{\contentsline {subsection}{\numberline
{2.2}Remote}{2}}
 \@writefile{toc}{\contentsline {subsection}{\numberline
{2.3}Distributed}{2}}
```

Undoing things - locally

- *Change* last commit: --anmend

  Only amend commits that are still local and have not been pushed somewhere.

```
$ git commit -m 'Initial commit'
$ git add forgotten_file
$ git commit --amend
```

- *Unstage* with reset (or restore)

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

    renamed:    README.md -> README
    modified:   CONTRIBUTING.md
```

```
$ git reset HEAD CONTRIBUTING.md
Unstaged changes after reset:
M       CONTRIBUTING.md
```

```
$ git restore --staged CONTRIBUTING.md
```

- *Unmodifying* a modified file with checkout (or restore)

```
$ git checkout -- CONTRIBUTING.md
```

```
$ git restore CONTRIBUTING.md
```

Reverts to last commited form

Branch and merge

Nombra una serie de commits y combina esfuerzos ya completados

```
$ git branch
```

Enumera todas las ramas en el repositorio actual

```
$ git branch [branch-name]
```

Crea una nueva rama

```
$ git checkout [branch-name]
```

Cambia a la rama especificada y actualiza el directorio activo

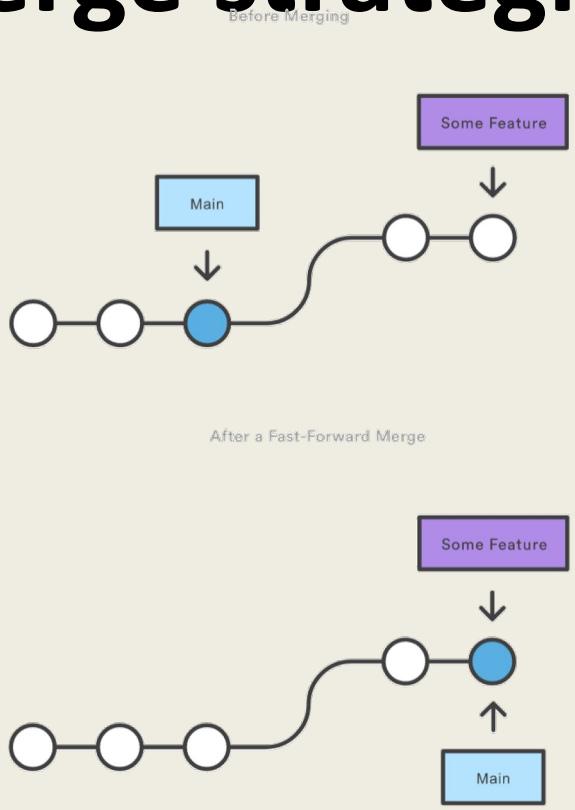
```
$ git merge [branch-name]
```

Combina el historial de la rama especificada con la rama actual

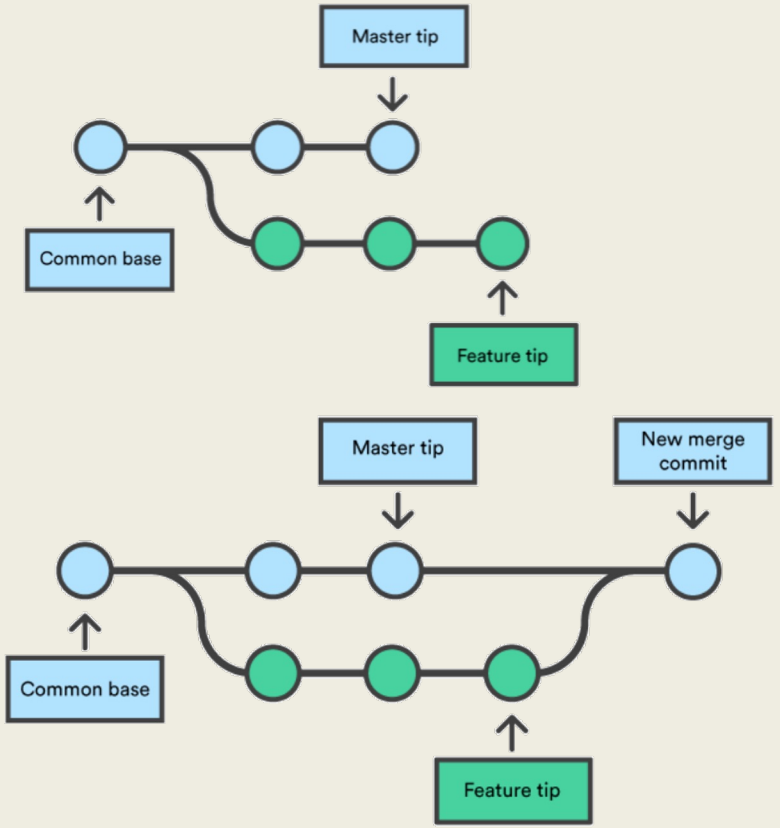
```
$ git branch -d [branch-name]
```

Borra la rama especificada

Merge strategies



Fast-forward merge



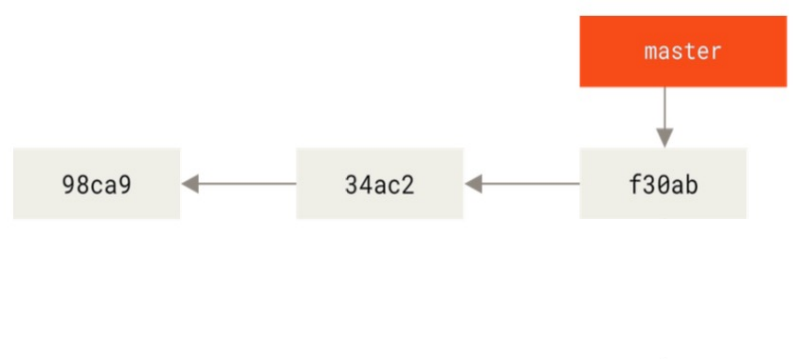
Recursive (3-way merge)

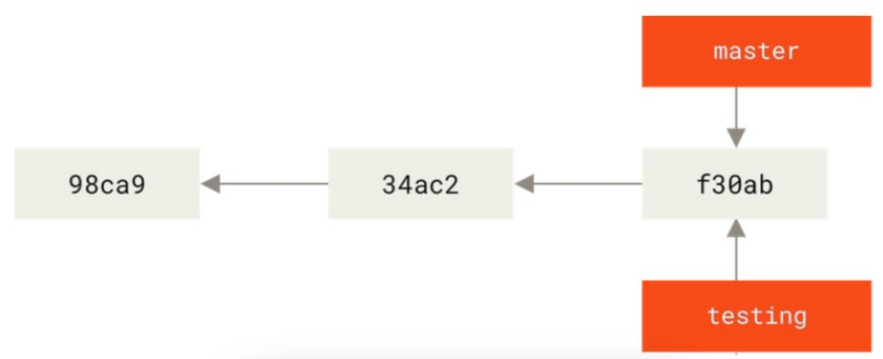
There are more types of merge strategies :

- Ours
- Octopus
- Resolve
- Subtree
- ...

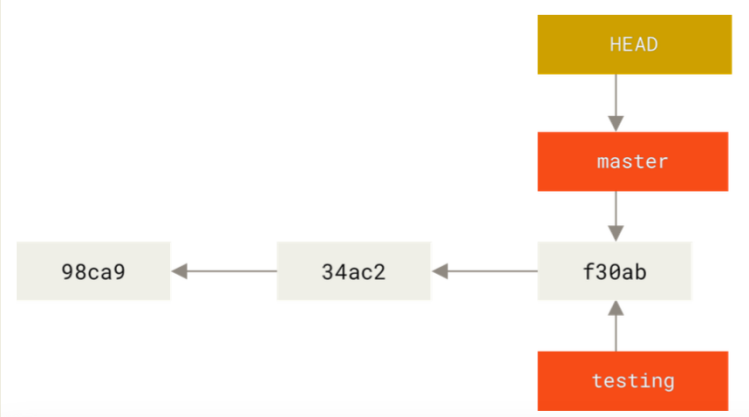
Basic branching: HEAD

HEAD is an alias for the tip of the currently checked out branch





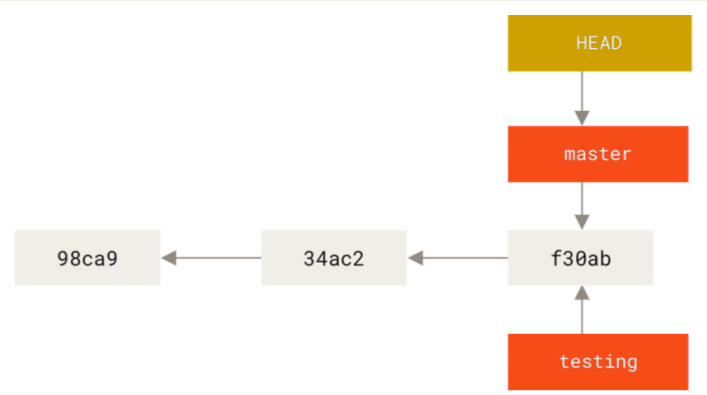

```
$ git branch testing
```

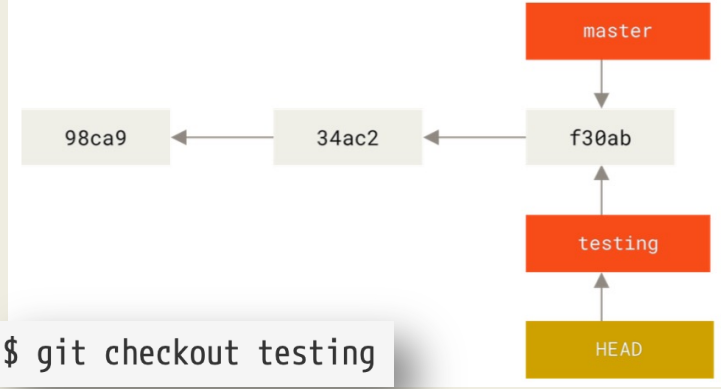



```
$ git log --oneline --decorate
f30ab (HEAD -> master, testing) Add feature #32
```

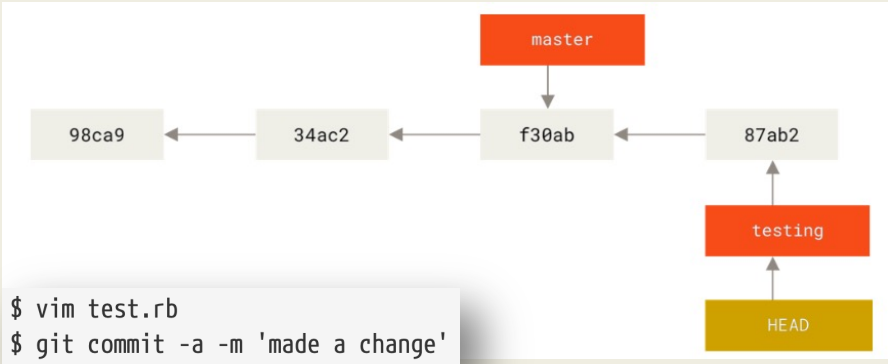

```
$ git checkout testing
```

Basic branching: diverging branches

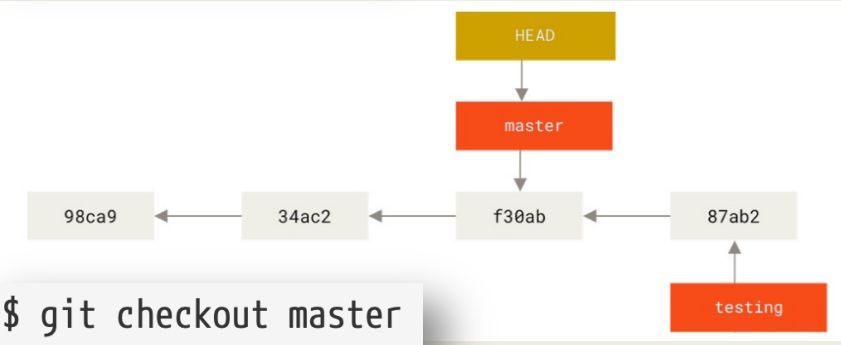




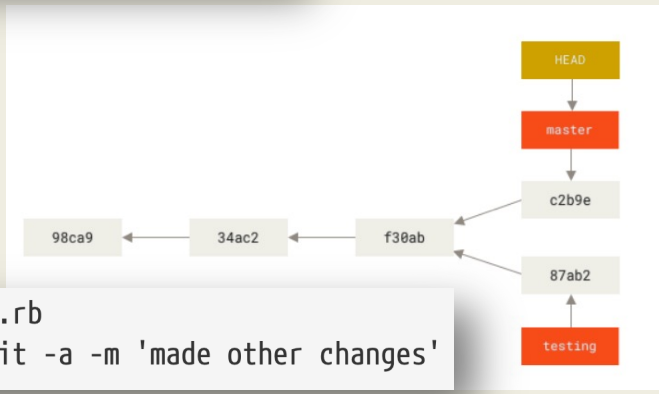

```
$ git checkout testing
```




```
$ vim test.rb
$ git commit -a -m 'made a change'
```

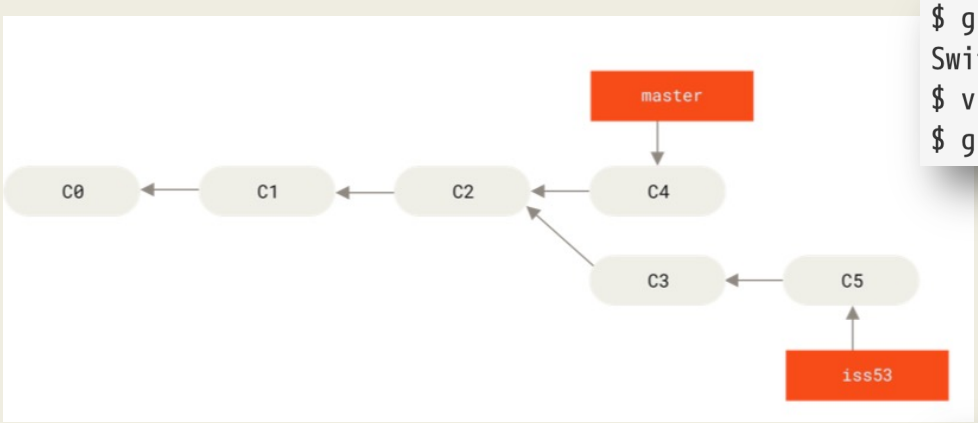



```
$ git checkout master
```

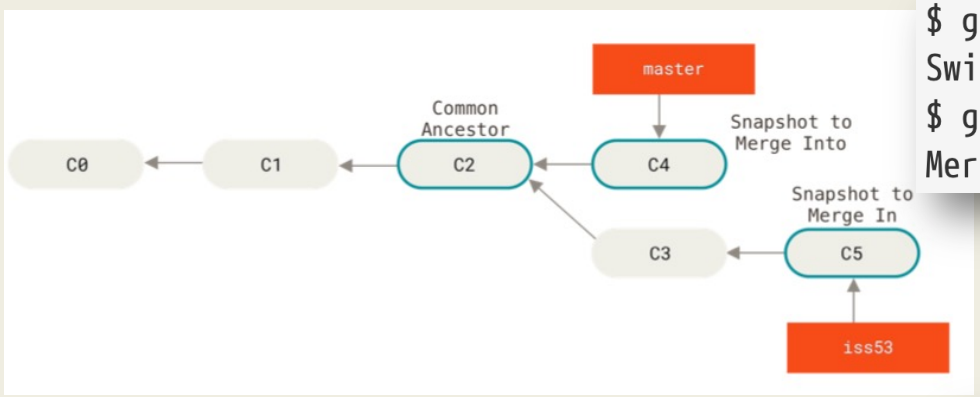



```
$ vim test.rb
$ git commit -a -m 'made other changes'
```

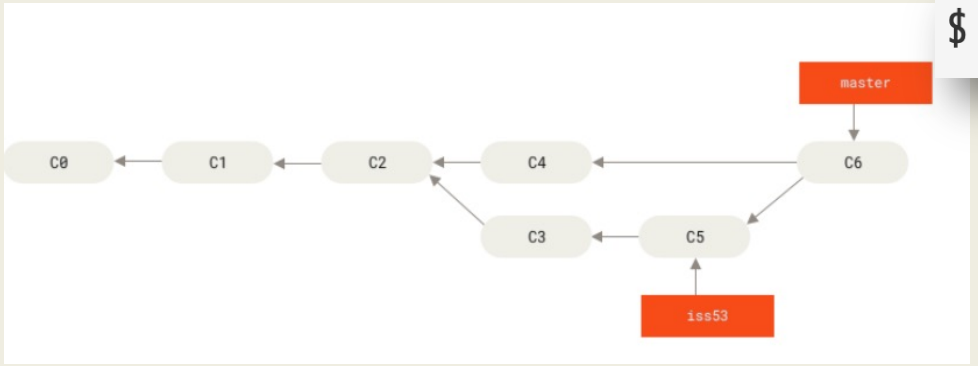
Basic branching: 3-way merge




```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'Finish the new footer [issue 53]'
```

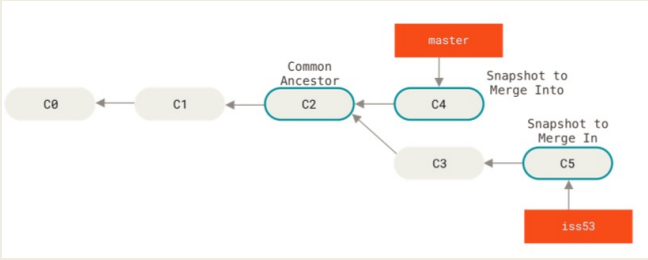



```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
```




```
$ git branch -d iss53
```

Basic branching: conflict




```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

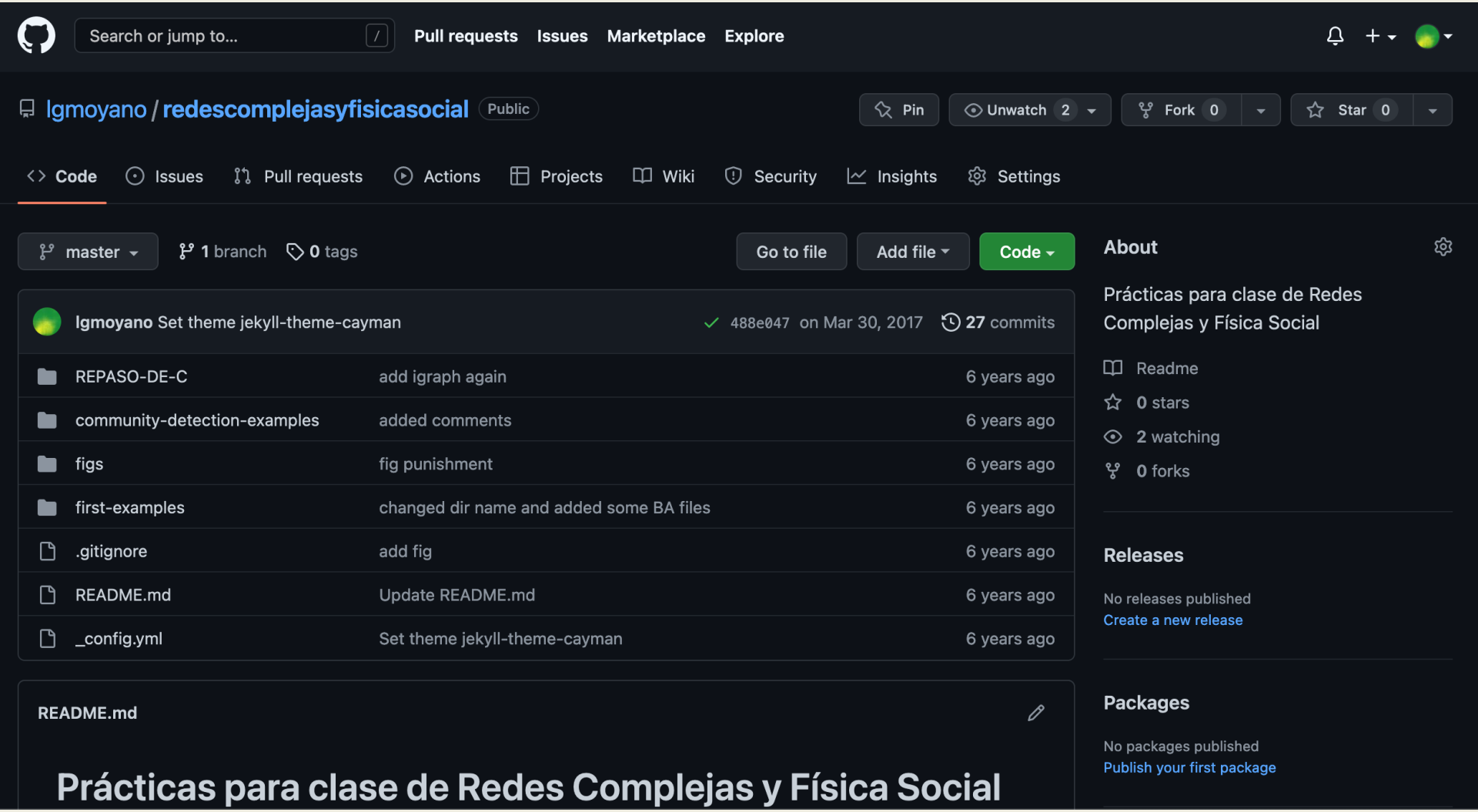
```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

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

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

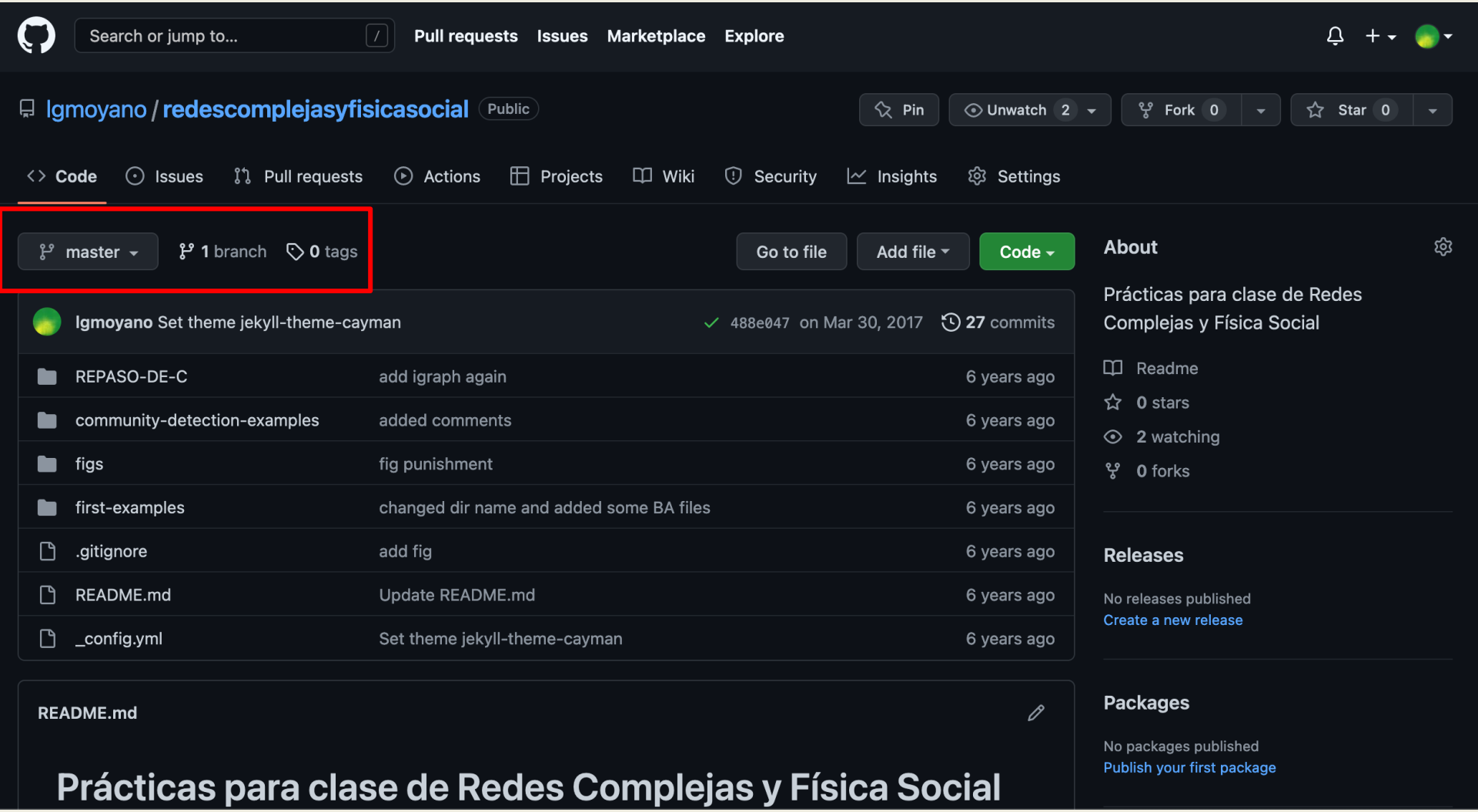
```
<div id="footer">
please contact us at email.support@github.com
</div>
```

```
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

    modified:   index.html
```

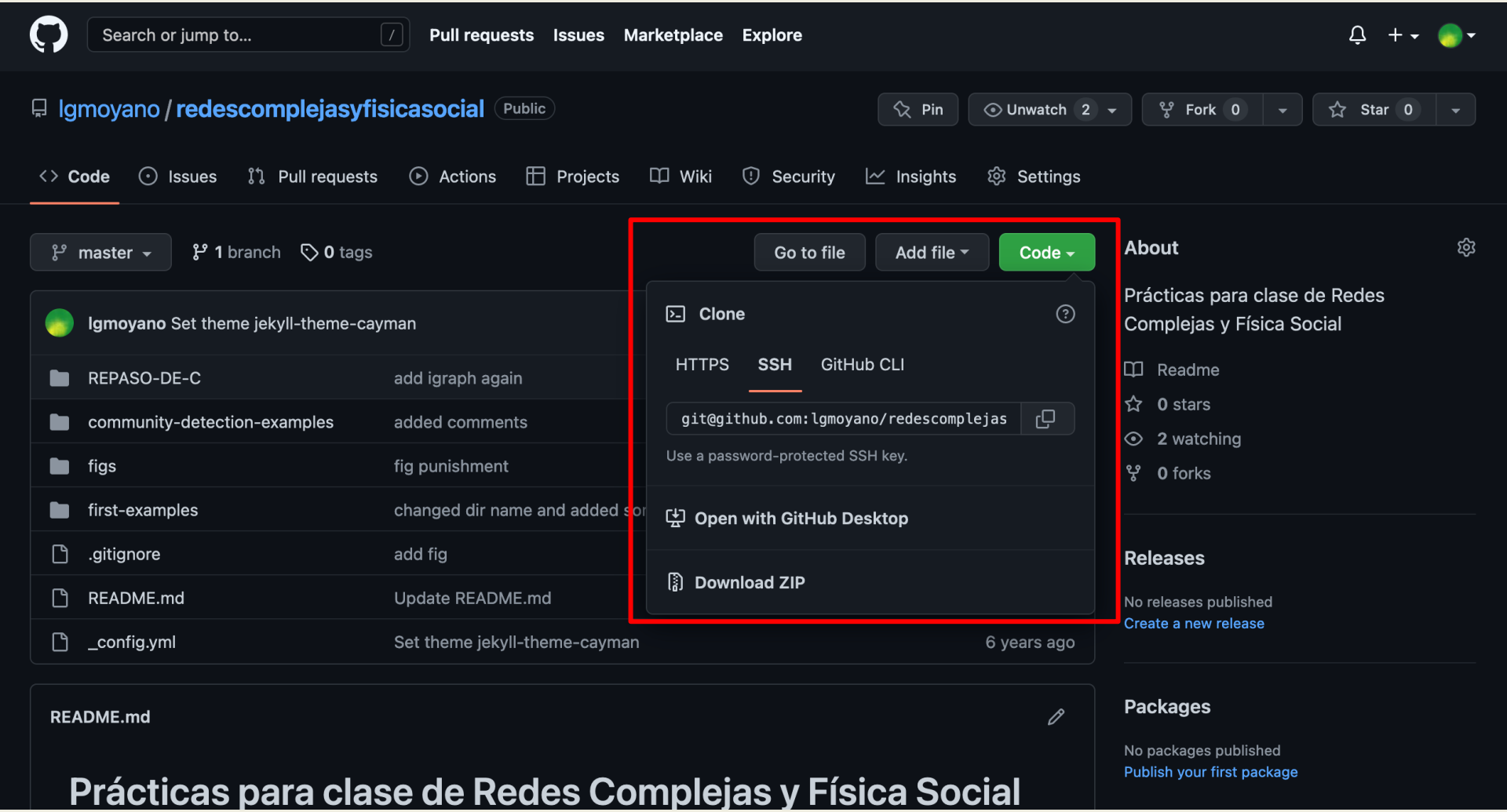
GitHub – main page



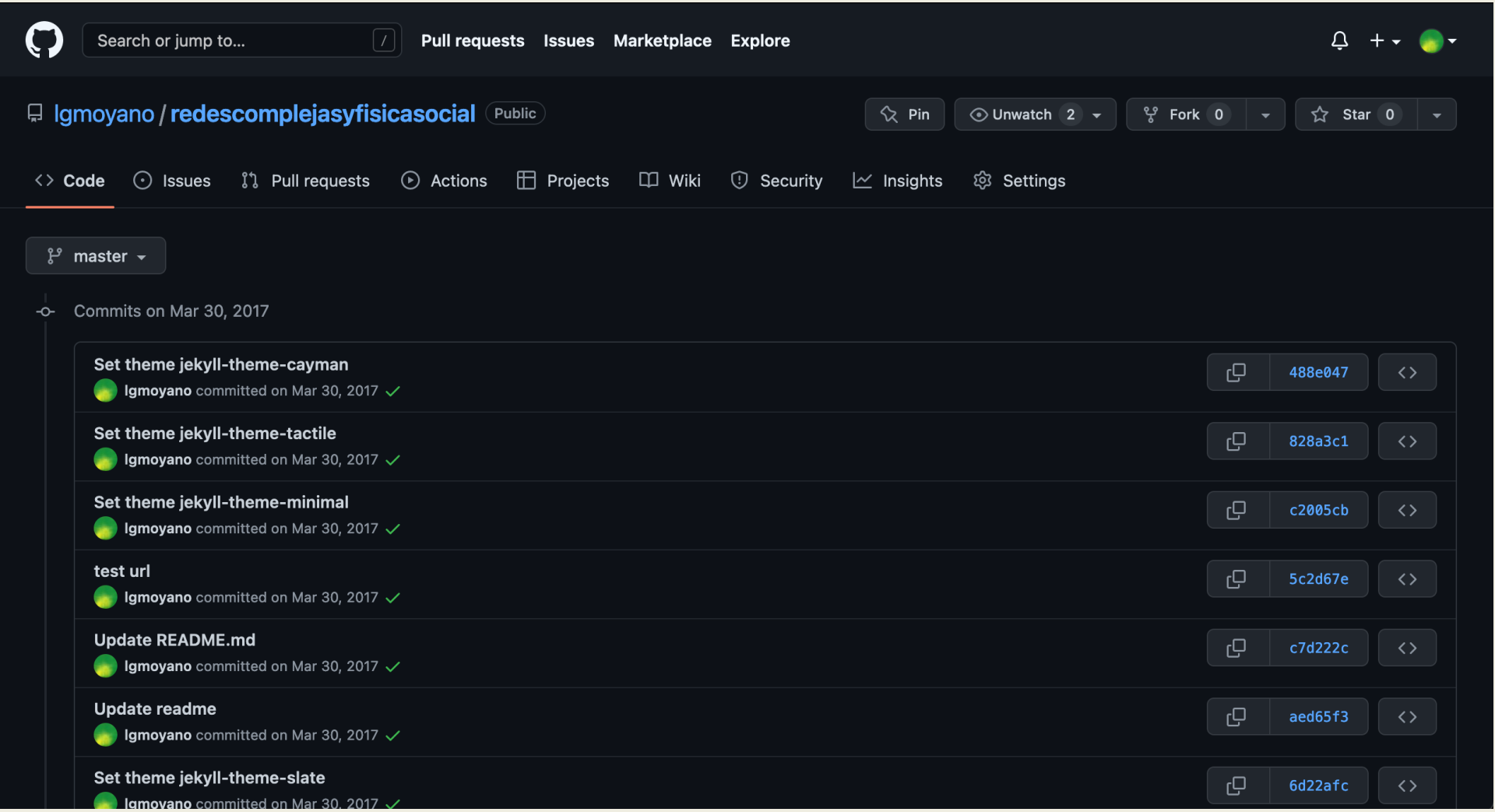
GitHub – main page

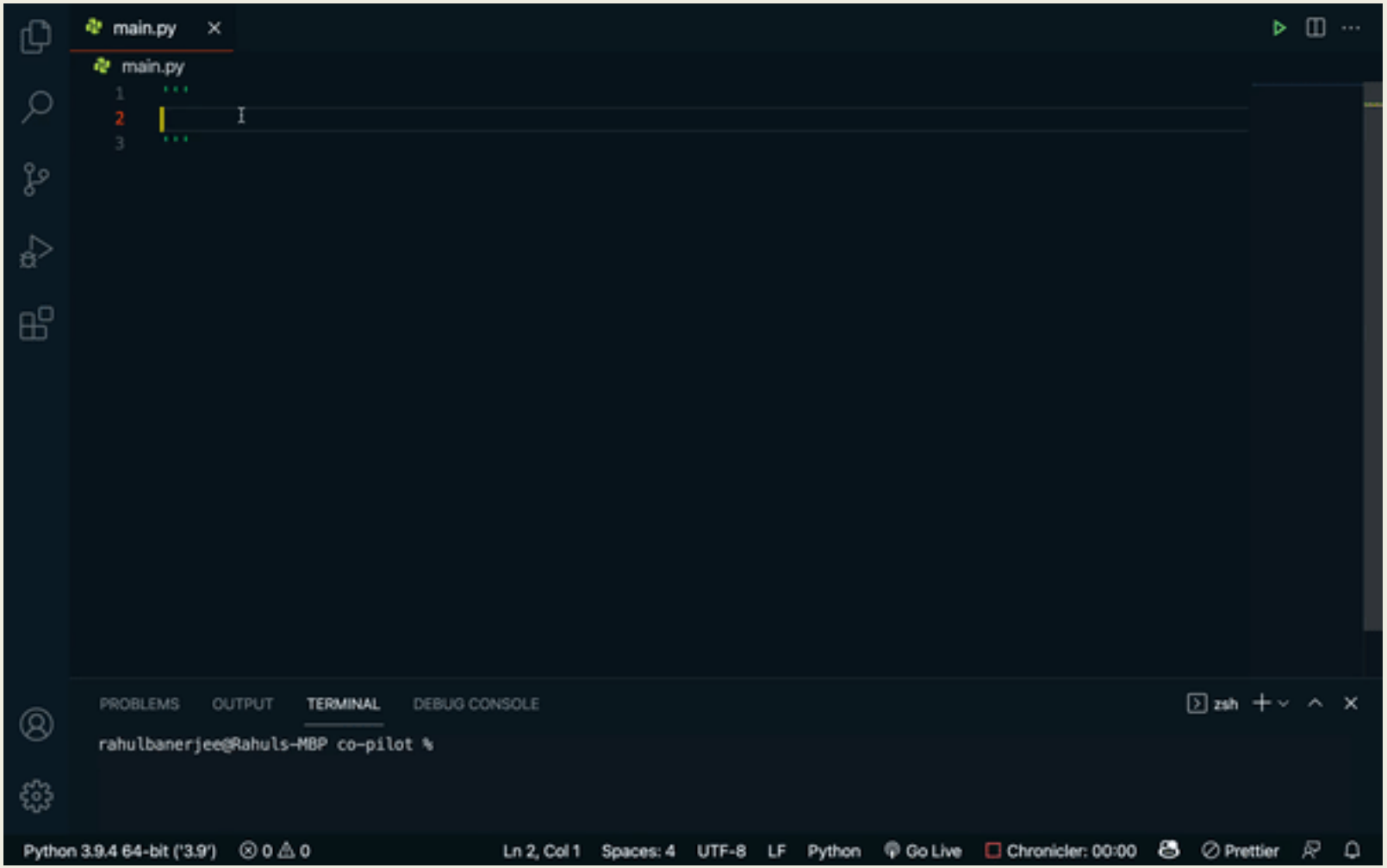


GitHub – main page: Code



GitHub - Commits




GitHub
Copilot
main.py
main.py
'''
'''
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
zsh
rahulbanerjee@Rahuls-MBP co-pilot %
Python 3.9.4 64-bit ('3.9')
Ln 2, Col 1
Spaces: 4
UTF-8
LF
Python
Go Live
Chronicler: 00:00
Prettier

Config SSH keys to use GitHub

- GitHub improved security by dropping older, insecure key types on March 15, 2022.
- To access via SSH, one needs to authenticate with private SSH keys (access credentials for accesing the SSH protocol).
- **Steps:**
  - Generate a new private SSH key. Follow prompts.

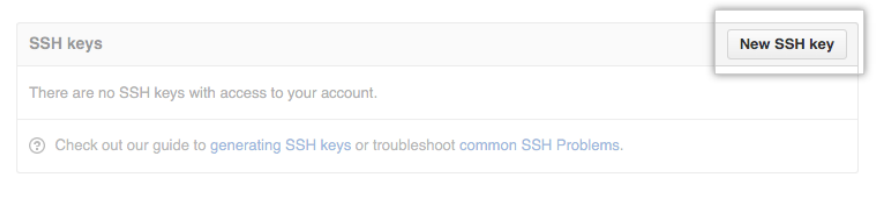
```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

  - Add it to the SSH agent.

```
$ eval "$(ssh-agent -s)"
> Agent pid 59566
```

```
ssh-add -K /Users/you/.ssh/id_rsa
```

  - Add the public SSH key to your account through GitHub's settings



https://www.atlassian.com/git/tutorials/git-ssh

https://docs.github.com/en/authentication/connecting-to-github-with-ssh

Synchronize changes from 'origin'

Registrar un marcador para un repositorio e intercambiar historial de versiones

```
$ git fetch [bookmark]
```

Descarga todo el historial del marcador del repositorio

```
$ git merge [bookmark]/[branch]
```

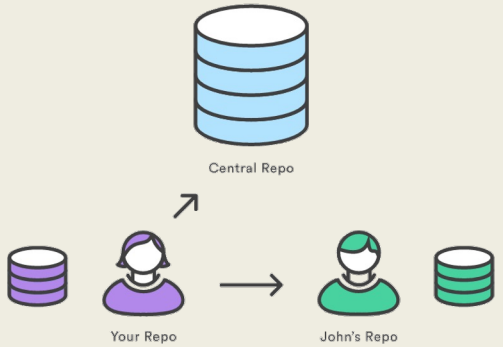
Combina la rama del marcador con la rama local actual

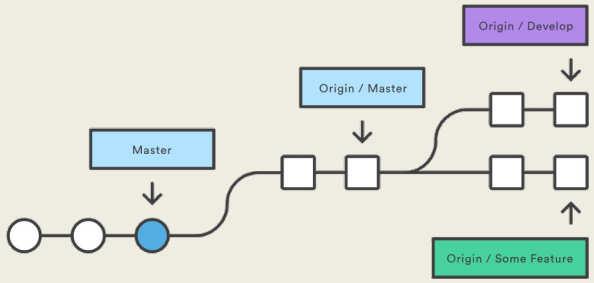
```
$ git push [alias] [branch]
```

Sube todos los commits de la rama local a GitHub

```
$ git pull
```

Descarga el historial del marcador e incorpora cambios





Basic remote workflow

Assuming one's got everithing runing smoothly

```
git add <file1> <file2>
git commit –m 'my message'
git status
    (to check relative position wrt 'origin')
git pull
    (to absorb changes if needed)
git push
    (to 'upload' local changes)
```

Workflow Overview

The single thing NOT to do:

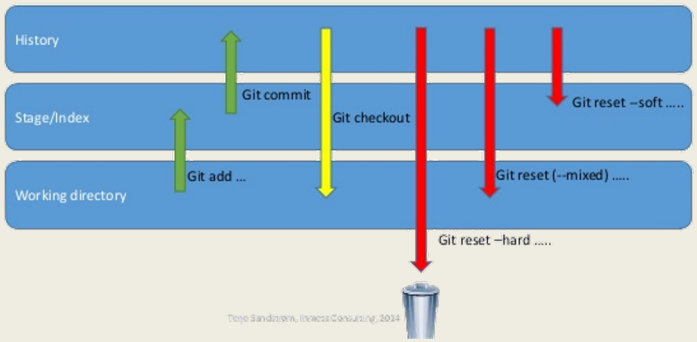
DO NOT rewrite remote history

- Sometimes, you may need to change history. You may need to undo a commit.
- But don't rewrite ***remote*** history if working with collaborators. You could create big problems for them.
  - Don't do `git reset --hard` if changes already went to repo. Use `git revert` instead, the safest way to undo things.
  - Beware of `push --force`, you may delete other people's work
  - Don't *ammend* a commit in the remote casually

Many more useful commands to explore

- *"At some point in your Git journey, you may accidentally lose a commit."*
  - git reflog
  - git fsck --lost-found (10.7 Git Internals - Maintenance and Data Recovery)
- reset –hard, --mixed, --soft
- stash
- tags and logs
- rebase, squash,
- cherrypick

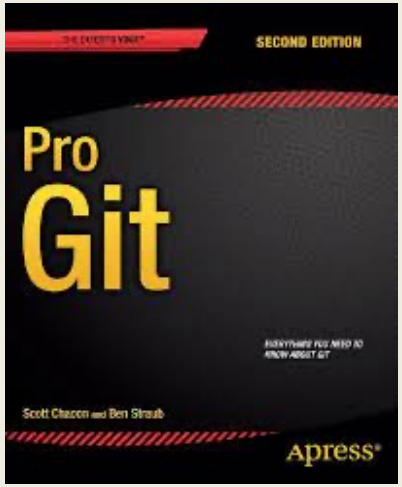
Git tree movements visualized



https://davidzych.com/difference-between-git-reset-soft-mixed-and-hard/

Git guide + Help

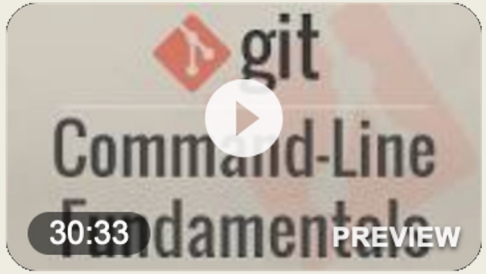
- Get hands dirty!
- Some questions
- Some problems, organized by scenarios
  - Third part in pairs
- Use Git Pro @biblio, as well as SO & the web



stackoverflow

git --everything-is-local

https://git-scm.com/



Summary: see [git - the simple guide]

https://rogerdudler.github.io/git-guide/