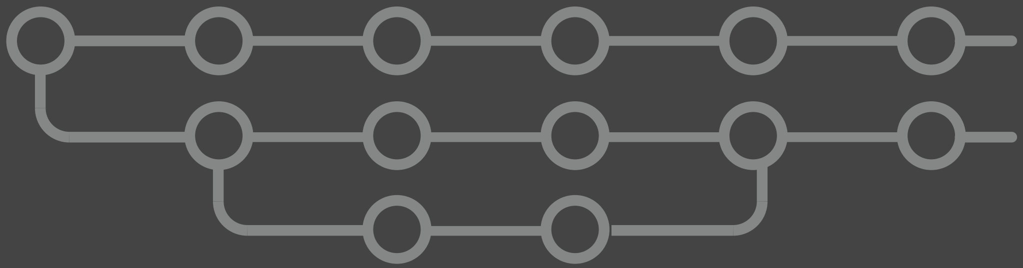
// GIT CONCEPTS

GIT IS NOT A SAVEPOINT



What is git?

Git is a popular distributed version control system that allows developers to manage source code changes, track versions, collaborate with others, and maintain a history of code revisions.

The author

// Basic usage

```
➜  killer-app pwd
/home/gianiaz/apps/killer-app
➜  killer-app ls -lah
totale 8,0K
drwxrwxr-x 2 gianiaz gianiaz 4,0K mar 31 15:31 .
drwxrwxr-x 3 gianiaz gianiaz 4,0K mar 31 15:31 ..

➜  killer-app git init
Inizializzato repository Git vuoto in /home/gianiaz/apps/killer-app/.git/
```

We have now an empty (and *local*) repository!

Let’s do some tweaks:

```
➜ git config --global user.email gianiaz@gmail.com
➜ cat ~/.gitconfig
[user]
    email = gianiaz@gmail.com
➜ git config --global user.name "Giovanni Lenoci"
➜ cat ~/.gitconfig
[user]
    email = gianiaz@gmail.com
    name = Giovanni Lenoci
```

Were git stores is data?

```
➜  killer-app git:(main) ✗ ls -lah
totale 16K
drwxrwxr-x 3 gianiaz gianiaz 4,0K mar 31 15:42 .
drwxrwxr-x 3 gianiaz gianiaz 4,0K mar 31 15:31 ..
drwxrwxr-x 7 gianiaz gianiaz 4,0K mar 31 15:31 .git
```

What's inside .git directory?

```
➜  killer-app git:(main) ✗ cd .git
➜  .git git:(main) ls -la
totale 40
drwxrwxr-x 7 gianiaz gianiaz 4096 mar 31 15:31 .
drwxrwxr-x 3 gianiaz gianiaz 4096 mar 31 15:42 ..
drwxrwxr-x 2 gianiaz gianiaz 4096 mar 31 15:31 branches
-rw-rw-r-- 1 gianiaz gianiaz   92 mar 31 15:31 config
-rw-rw-r-- 1 gianiaz gianiaz   73 mar 31 15:31 description
-rw-rw-r-- 1 gianiaz gianiaz   21 mar 31 15:31 HEAD
drwxrwxr-x 2 gianiaz gianiaz 4096 mar 31 15:31 hooks
drwxrwxr-x 2 gianiaz gianiaz 4096 mar 31 15:31 info
drwxrwxr-x 4 gianiaz gianiaz 4096 mar 31 15:31 objects
drwxrwxr-x 4 gianiaz gianiaz 4096 mar 31 15:31 refs
```

Let’s create a killer content ;-)

```
➜  killer-app git:(main) echo "#This app will make me rich" > README.md
➜  killer-app git:(main) ✗ ls -lah
totale 16K
drwxrwxr-x 3 gianiaz gianiaz 4,0K mar 31 15:48 .
drwxrwxr-x 3 gianiaz gianiaz 4,0K mar 31 15:31 ..
drwxrwxr-x 7 gianiaz gianiaz 4,0K mar 31 15:31 .git
-rw-rw-r-- 1 gianiaz gianiaz   28 mar 31 15:48 README.md
```

What’s the state of my git repo?

```
➜  killer-app git:(main) ✗ git status
Sul branch main

Non ci sono ancora commit

File non tracciati:
  (usa "git add <file>..." per includere l'elemento
  fra quelli di cui verrà eseguito il commit)

    README.md

non è stato aggiunto nulla al commit ma sono presenti file
non tracciati (usa "git add" per tracciarli)
```

Glossary

branch

A git branch is a separate version of the codebase that allows developers to work on features or fixes independently without affecting the main codebase until they are ready to merge them.

commit

A git commit is a snapshot of changes made to a repository at a specific point in time with a brief but descriptive message summarizing the changes made.

tracked/untracked files

files that are present/not present in the list of files to be committed

First commit

```
➜  killer-app git:(main) ✗ git commit -m "First commit with Readme file"
[main (commit radice) 72180f0] First commit with Readme file
 1 file changed, 1 insertion(+)
 create mode 100644 README.md

 ➜  killer-app git:(main) git status
Sul branch main
non c'è nulla di cui eseguire il commit, l'albero di lavoro è pulito
```

What shouldn’t be committed

- Credentials
- Personal data
- Configuration that changes between environments
- Libraries (look at ”What should be committed”)
- Generated files (e.g. built bundles)

What should be committed

- Your code :-P
- composer.lock and package.lock
- .gitignore
- documentation (https://www.writethedocs.org/guide/docs-as-code/)

Let’s open our code with phpstorm:

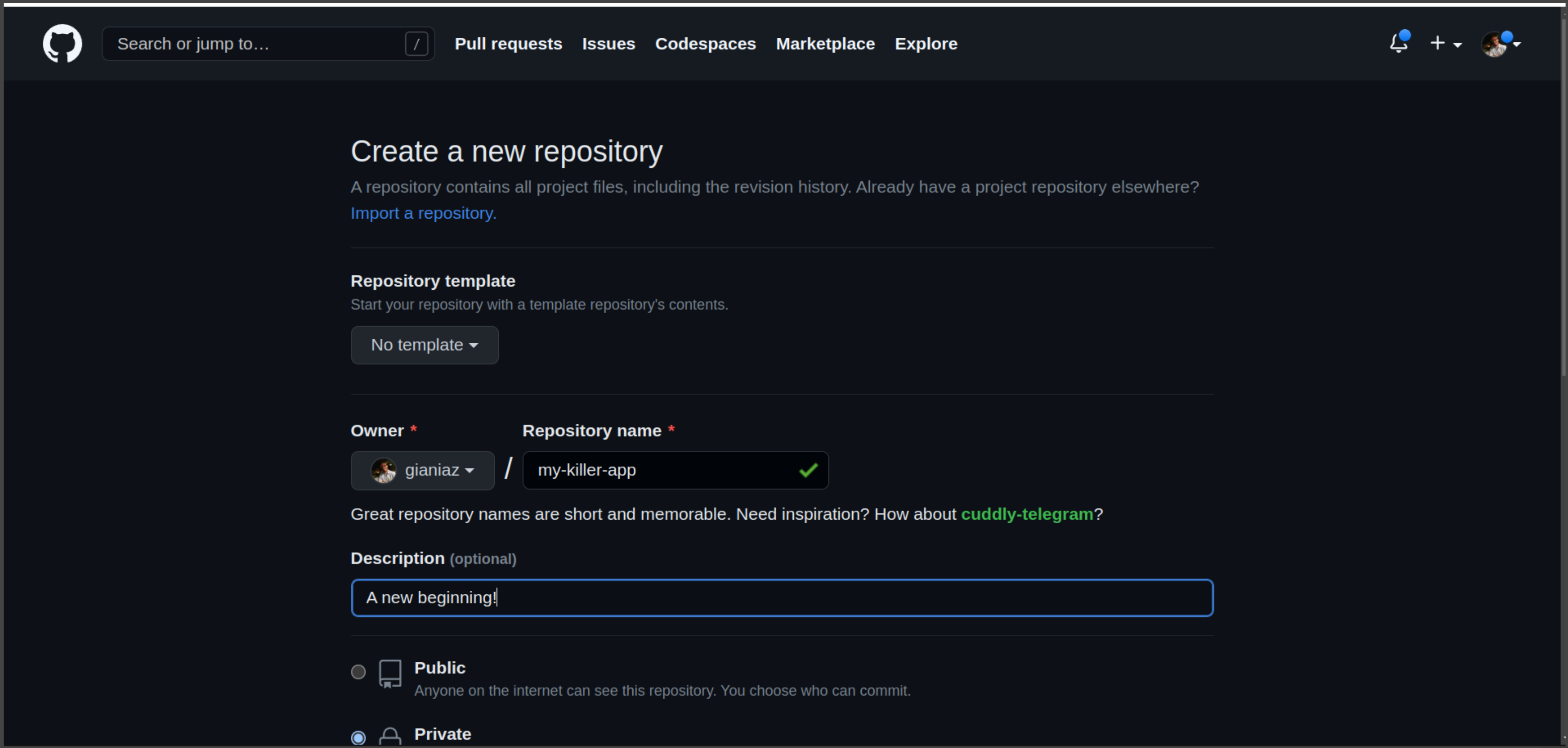
```
➜  killer-app git:(contributors) ✗ git status
Sul branch contributors
File non tracciati:
  (usa "git add <file>..." per includere l'elemento fra quelli di cui verrà eseguito
    .idea/

non è stato aggiunto nulla al commit ma sono presenti file non tracciati (usa "git a
➜  killer-app git:(contributors) ✗ touch .gitignore
```

.gitignore

```
➜  killer-app git:(contributors) ✗ git add .gitignore
➜  killer-app git:(contributors) ✗ git status
Sul branch contributors
Modifiche di cui verrà eseguito il commit:
  (usa "git restore --staged <file>..." per rimuovere gli elementi dall'area di stag
    nuovo file:             .gitignore

File non tracciati:
  (usa "git add <file>..." per includere l'elemento fra quelli di cui verrà eseguito
    .idea/

~/projects/personal/my-killer-app on my-new-branch ✗ > echo .idea/ >> .gitignore

~/projects/personal/my-killer-app on my-new-branch ✗ > git status
Sul branch my-new-branch
File non tracciati:
  (usa "git add <file>..." per includere l'elemento fra quelli di cui verrà eseguito
    .gitignore
```

// Let’s open to the outside world

A new github repository (origin)



Let’s add our new origin:

```
git remote add origin git@github.com:gianiaz/my-killer-app.git
git branch -M main
git push -u origin main
```

And now we can push our code on github:

```
➜  killer-app git:(main) git push
fatal: The current branch main has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin main

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

➜  killer-app git:(main) git push --set-upstream origin main

Enumerazione degli oggetti in corso: 3, fatto.
Conteggio degli oggetti in corso: 100% (3/3), fatto.
Scrittura degli oggetti in corso: 100% (3/3), 258 byte | 258.00 KiB/s, fatto.
3 oggetti totali (0 delta), 0 riutilizzati (0 delta), 0 riutilizzati nel file pack
To github.com:gianiaz/my-killer-app.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

How is it possible?

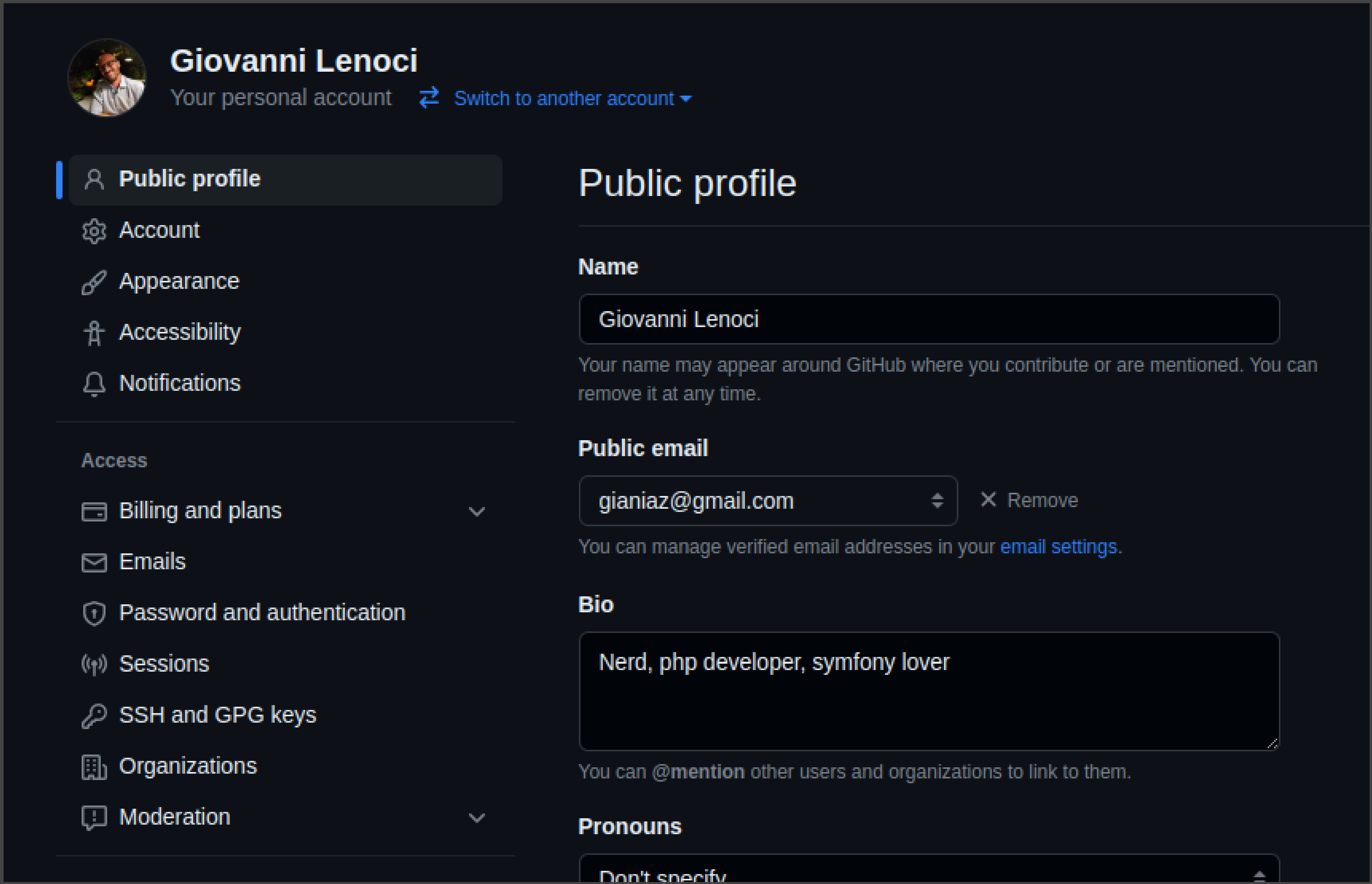
Git knows me.

But how?

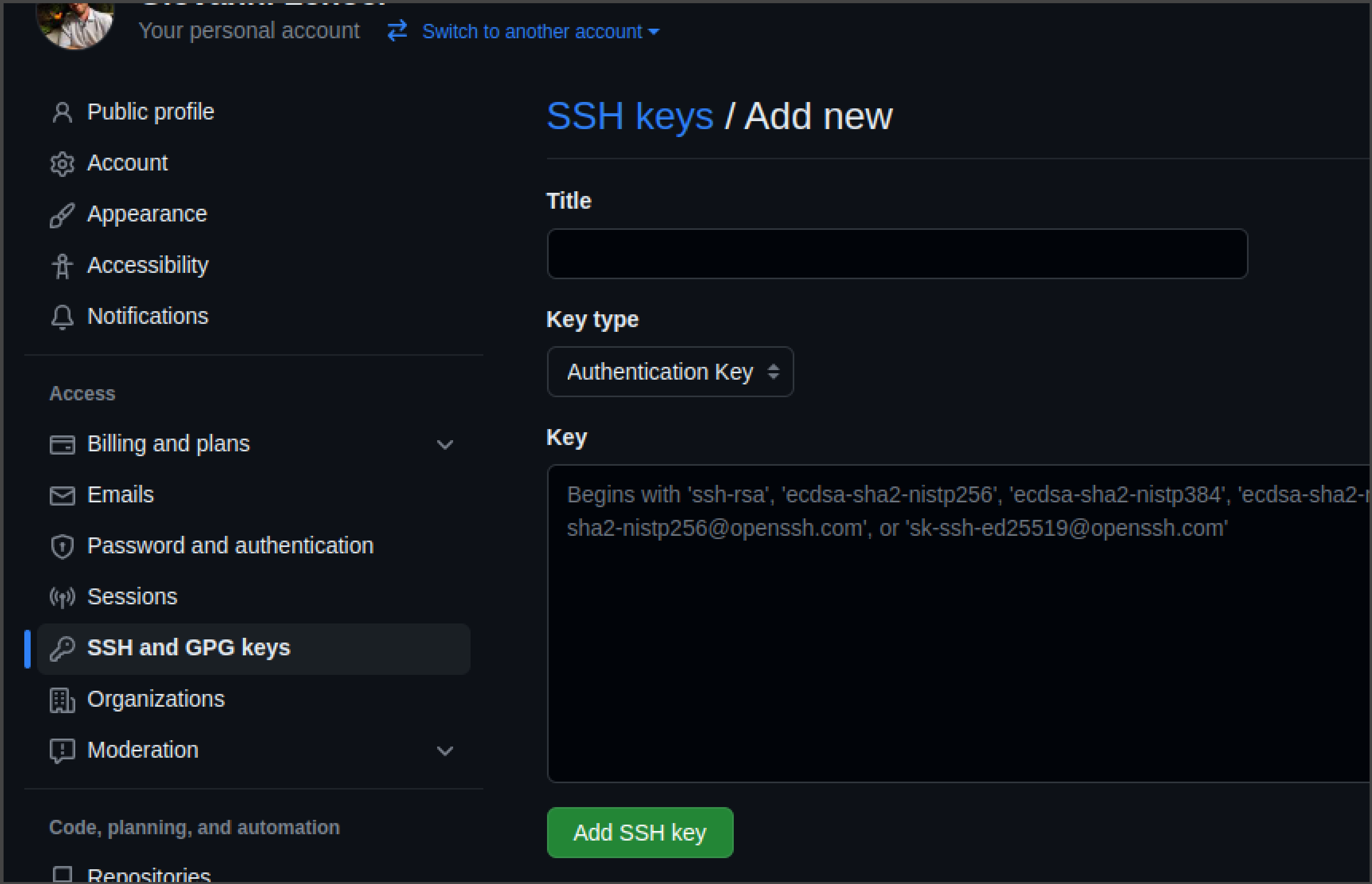
1. Generate an ssh key with:

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

2. Go to your profile



3. Add your key



This also work on a private repository for the same reason

```
~/projects/personal > git clone git@github.com:gianiaz/my-killer-app.git
Clone in 'my-killer-app' in corso...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
Ricezione degli oggetti: 100% (9/9), fatto.
remote: Total 9 (delta 0), reused 3 (delta 0), pack-reused 0
```

Fetch from origin

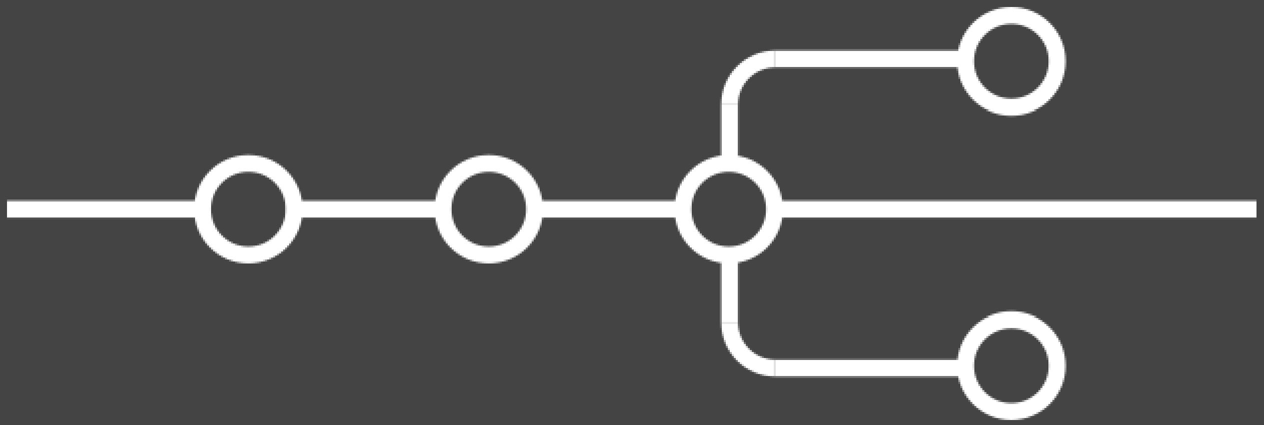
```
➜  killer-app git:(contributors) ✗ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Decompressione degli oggetti in corso: 100% (3/3), 689 byte | 689.00 KiB/s, fatto.
Da github.com:gianiaz/my-killer-app
 * [nuovo branch]    fixes-for-readme -> origin/fixes-for-readme
➜  killer-app git:(contributors) ✗
```

// Branches and stashes

Branches

Let’s create a new branch:

```
~/projects/personal/my-killer-app on main ✓ > git checkout -b "add-contributors"
Si è passati a un nuovo branch 'add-contributors'

~/projects/personal/my-killer-app on add-contributors ✓ >
```



Stash

We start by creating our file:

```
➜  killer-app git:(add-contributors) echo "* Giovanni Lenoci <gianiaz@gmail.com>" >>
➜  killer-app git:(add-contributors) ✗ git status
Sul branch contributors
File non tracciati:
  (usa "git add <file>..." per includere l'elemento fra quelli di cui verrà eseguito
    CONTRIBUTORS.md

non è stato aggiunto nulla al commit ma sono presenti file non tracciati (usa "git a
➜  killer-app git:(add-contributors) ✗

➜  killer-app git:(add-contributors) ✗ git add CONTRIBUTORS.md
➜  killer-app git:(add-contributors) ✗ git stash save "They interrupted me"
Directory di lavoro e stato indice salvati: On contributors: They interrupted me
➜  killer-app git:(add-contributors) git status
Sul branch contributors
non c'è nulla di cui eseguire il commit, l'albero di lavoro è pulito

➜  killer-app git:(add-contributors) ✗ git stash list
stash@{0}: On contributors: They interrupted me
(END)
```

```
~/projects/personal/my-killer-app on main ✓ > git checkout -b "urgent-bugfix"
Si è passati a un nuovo branch 'urgent-bugfix'

~/projects/personal/my-killer-app on urgent-bugfix ✓ >
```


Git
rebase
Git
Merge

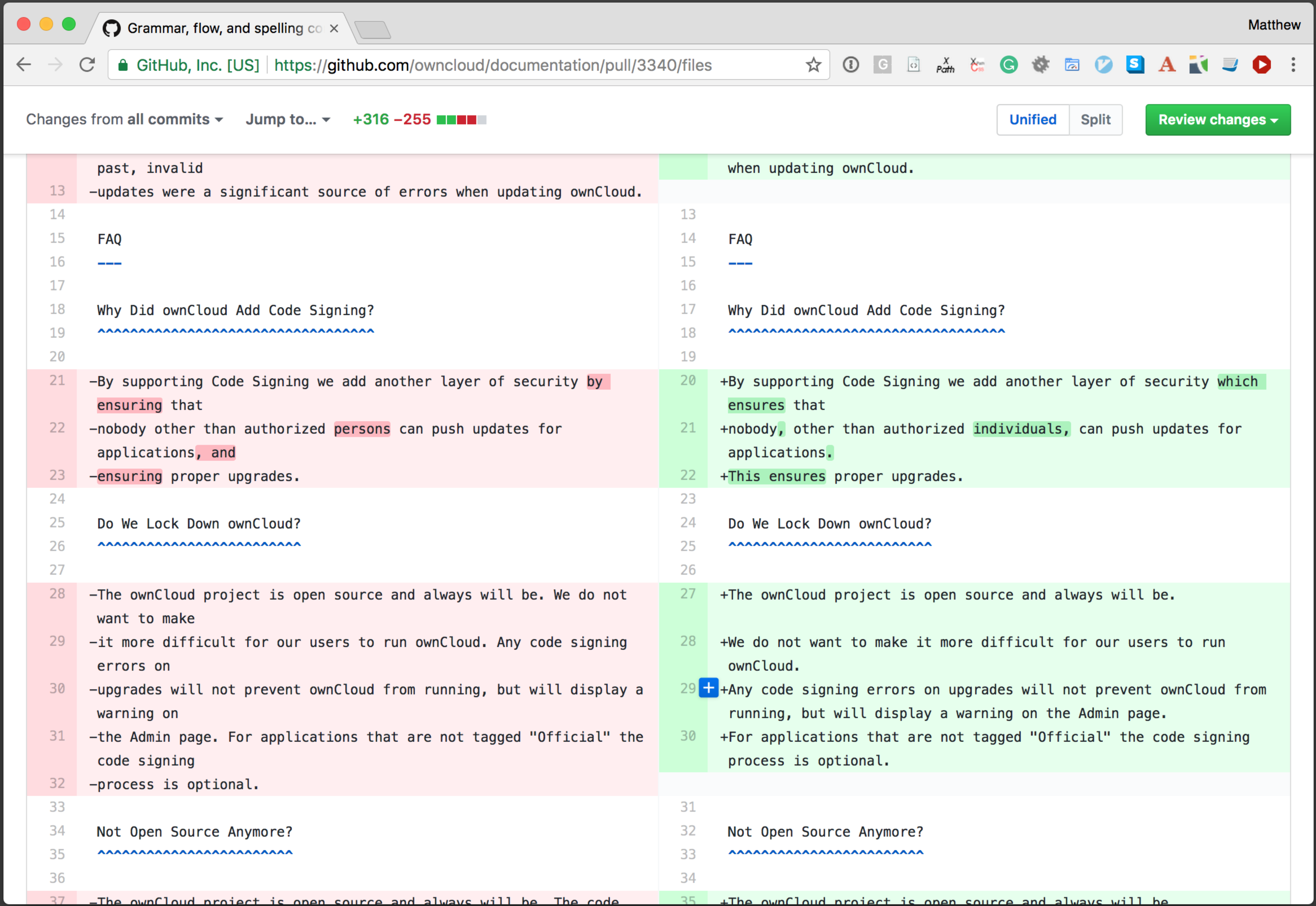
What is merge?

What is rebase

What is git rebase? From a content perspective, rebasing is changing the base of your branch from one commit to another making it appear as if you’d created your branch from a different commit. Internally, Git accomplishes this by creating new commits and applying them to the specified base

Conflicts

A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other.


Grammar, flow, and spelling co
Matthew
GitHub, Inc. [US] https://github.com/owncloud/documentation/pull/3340/files
Changes from all commits
Jump to...
+316 −255
Unified
Split
Review changes
past, invalid
13 -updates were a significant source of errors when updating ownCloud.
14
15 FAQ
16 ---
17
18 Why Did ownCloud Add Code Signing?
19 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
20
21 -By supporting Code Signing we add another layer of security by ensuring that
22 -nobody other than authorized persons can push updates for applications, and
23 -ensuring proper upgrades.
24
25 Do We Lock Down ownCloud?
26 ^^^^^^^^^^^^^^^^^^^^^^^^^
27
28 -The ownCloud project is open source and always will be. We do not want to make
29 -it more difficult for our users to run ownCloud. Any code signing errors on
30 -upgrades will not prevent ownCloud from running, but will display a warning on
31 -the Admin page. For applications that are not tagged "Official" the code signing
32 -process is optional.
33
34 Not Open Source Anymore?
35 ^^^^^^^^^^^^^^^^^^^^^^^^
36
37 -The ownCloud project is open source and always will be. The code
when updating ownCloud.
13
14 FAQ
15 ---
16
17 Why Did ownCloud Add Code Signing?
18 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
19
20 +By supporting Code Signing we add another layer of security which ensures that
21 +nobody, other than authorized individuals, can push updates for applications.
22 +This ensures proper upgrades.
23
24 Do We Lock Down ownCloud?
25 ^^^^^^^^^^^^^^^^^^^^^^^^^
26
27 +The ownCloud project is open source and always will be.
28 +We do not want to make it more difficult for our users to run ownCloud.
29 +Any code signing errors on upgrades will not prevent ownCloud from running, but will display a warning on the Admin page.
30 +For applications that are not tagged "Official" the code signing process is optional.
31
32 Not Open Source Anymore?
33 ^^^^^^^^^^^^^^^^^^^^^^^^
34
35 +The ownCloud project is open source and always will be.


ENERGY 99
010
GIT
IS NOT
A SAVEPOINT

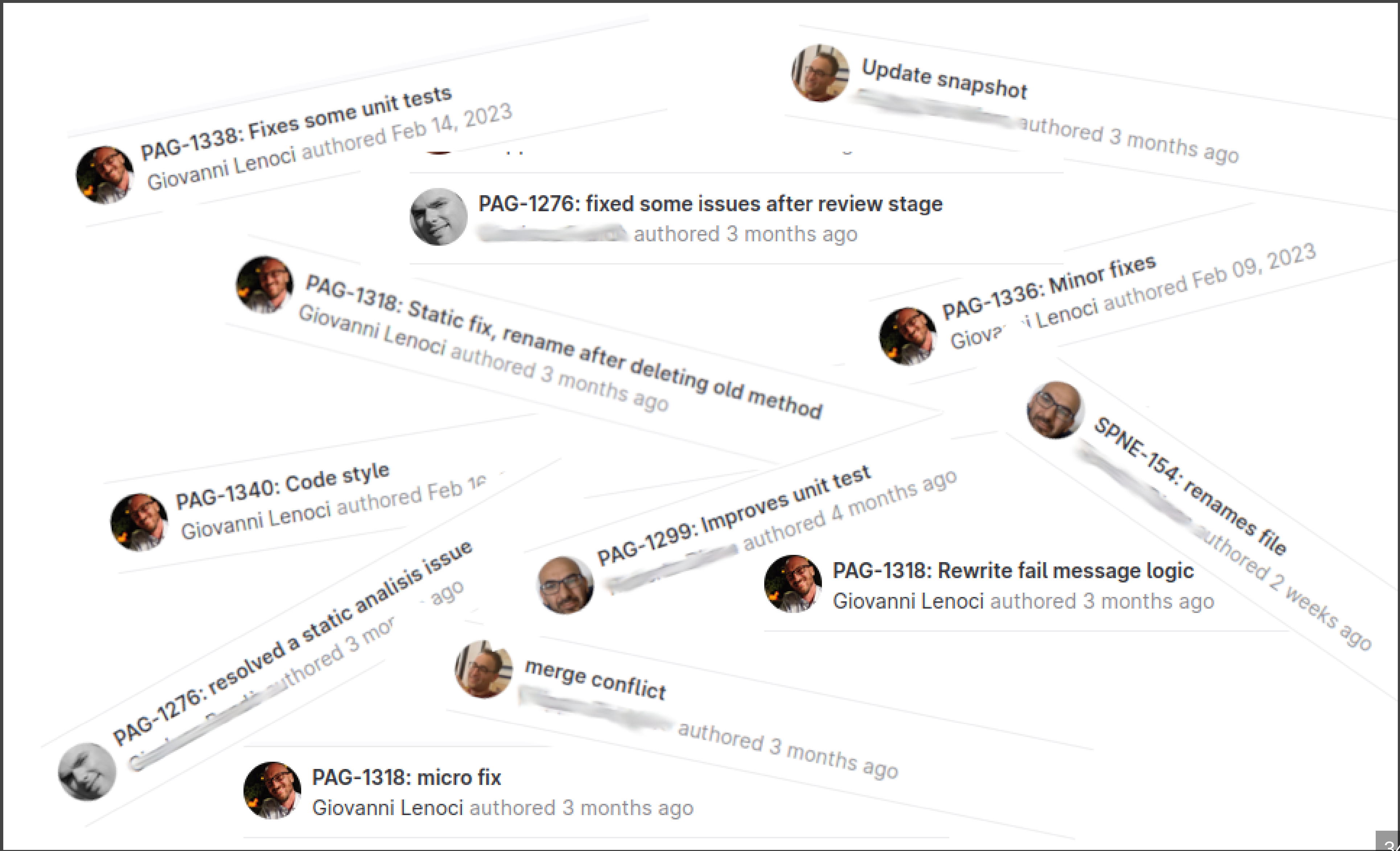
Git is not a savepoint

Atomic commits

Why atomic?

Because an atom can’t be divided!

An example of bad commit messages



Conventional commits

The Conventional Commits specification is a lightweight convention on top of commit messages.

It provides an easy set of rules for creating an explicit commit history.

The commit message should be structured as follows:

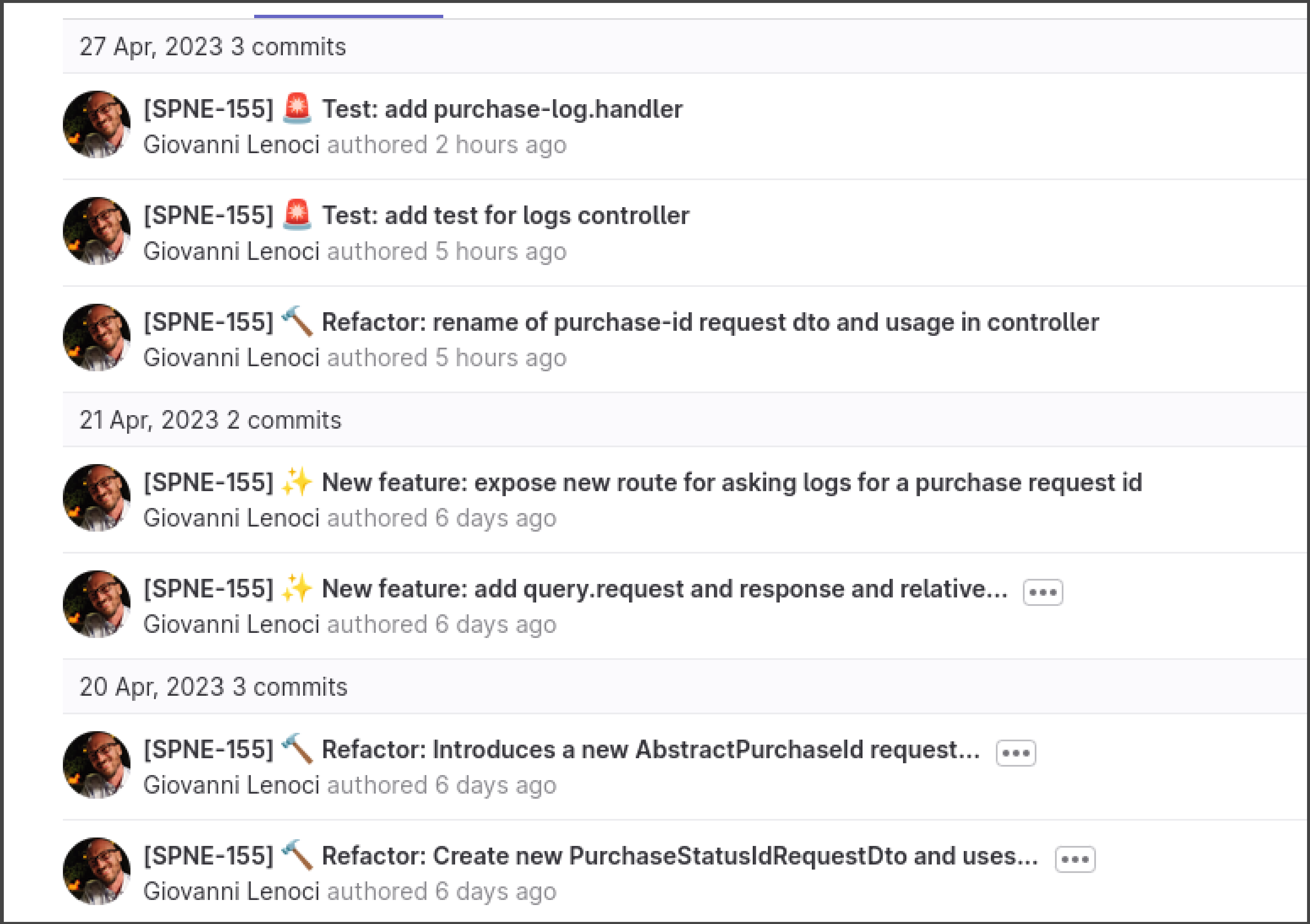
```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

es.

New Feature: $ISSUE_CODE: Add to the XapiClient the delete method

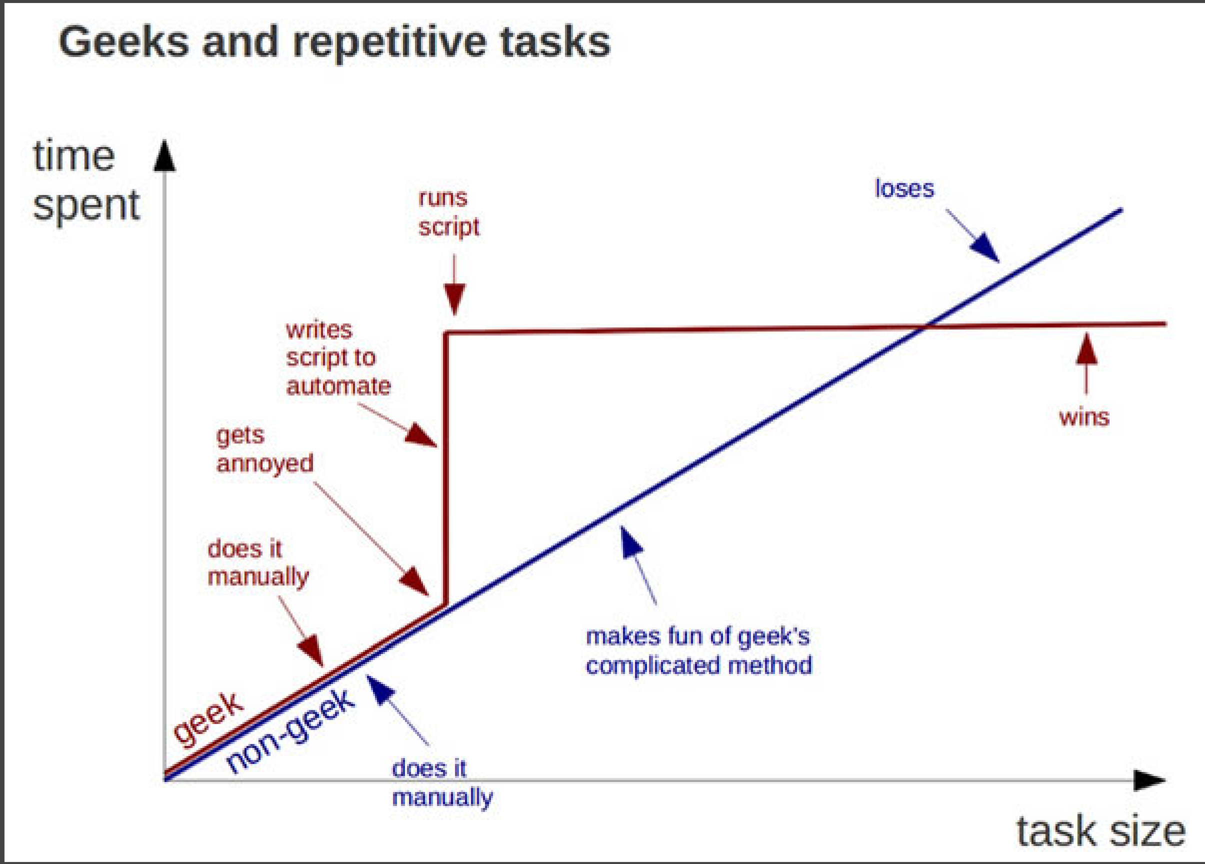
Emoji for conventional commit


gitmoji
An emoji guide for your commit messages
GitHub
Tweet
Your new development career awaits. Check out the latest listings.
ADS VIA CARBON
Search your gitmoji...
Ctrl K
:art:
Improve structure / format of the code.
:zap:
Improve performance.
:fire:
Remove code or files.
:bug:
Fix a bug.



An example of a better commit history

Geeks and Repetitive Tasks



We want to write less and do more:

From this.

```
git commit -m "T add purchase-log.handler"
```

To this.

```
commit c2ecbe11aad7d0cce5c39e70e68e2481eb2b3f29 (HEAD -> SPNE-155-expose-logs, origi
Author: Giovanni Lenoci <giovanni.lenoci@facile.it>
Date:   Thu Apr 27 12:26:00 2023 +0200

    [SPNE-155] :rotating_light: Test: add purchase-log.handler
```

Here you can find a summary table of what we want to achieve. And here you can find the prepare-commit-message script which must be put in `.git/hooks` directory

Rebase > Merge (but with a condition)

Rebase allows you to integrate the changes from one branch into another.

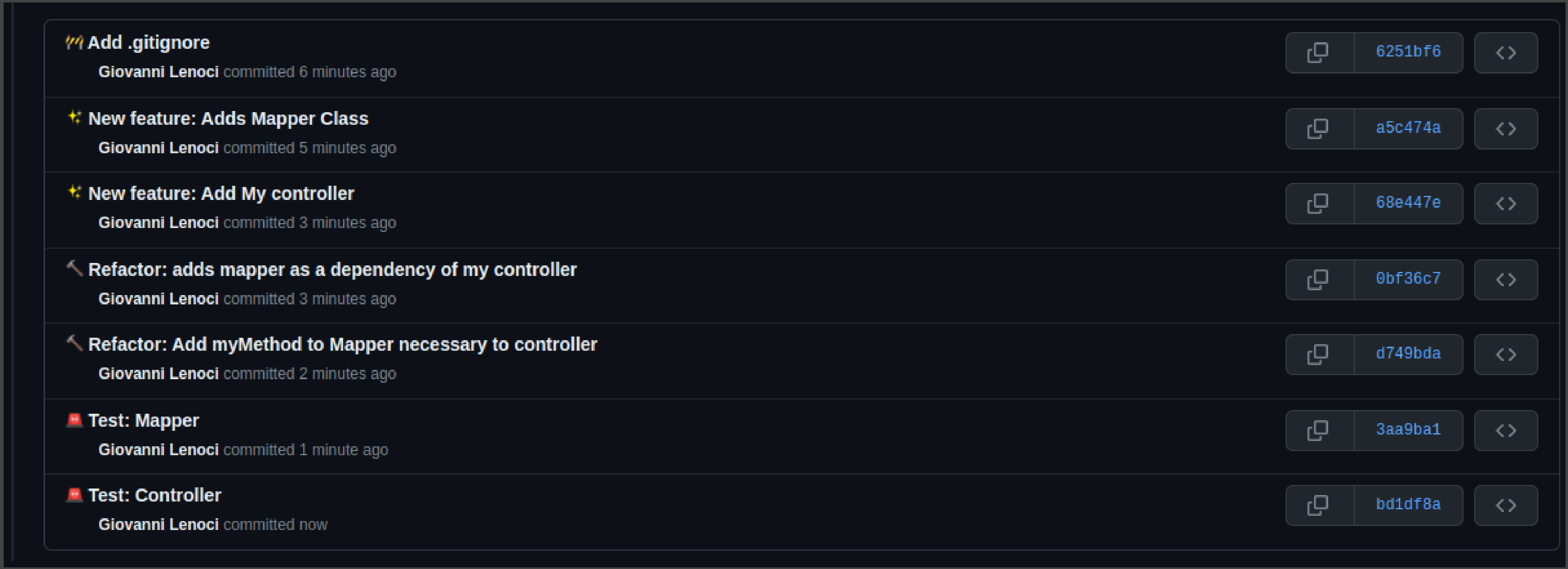
Pros

- Code history remains flat and readable
- Manipulating a single commit is easy

Cons

- It’s more work
- Rebase rewrite history

1.Rebase -i



A not so good commit history

2.Rebase -i

`git rebase -i HEAD~7`

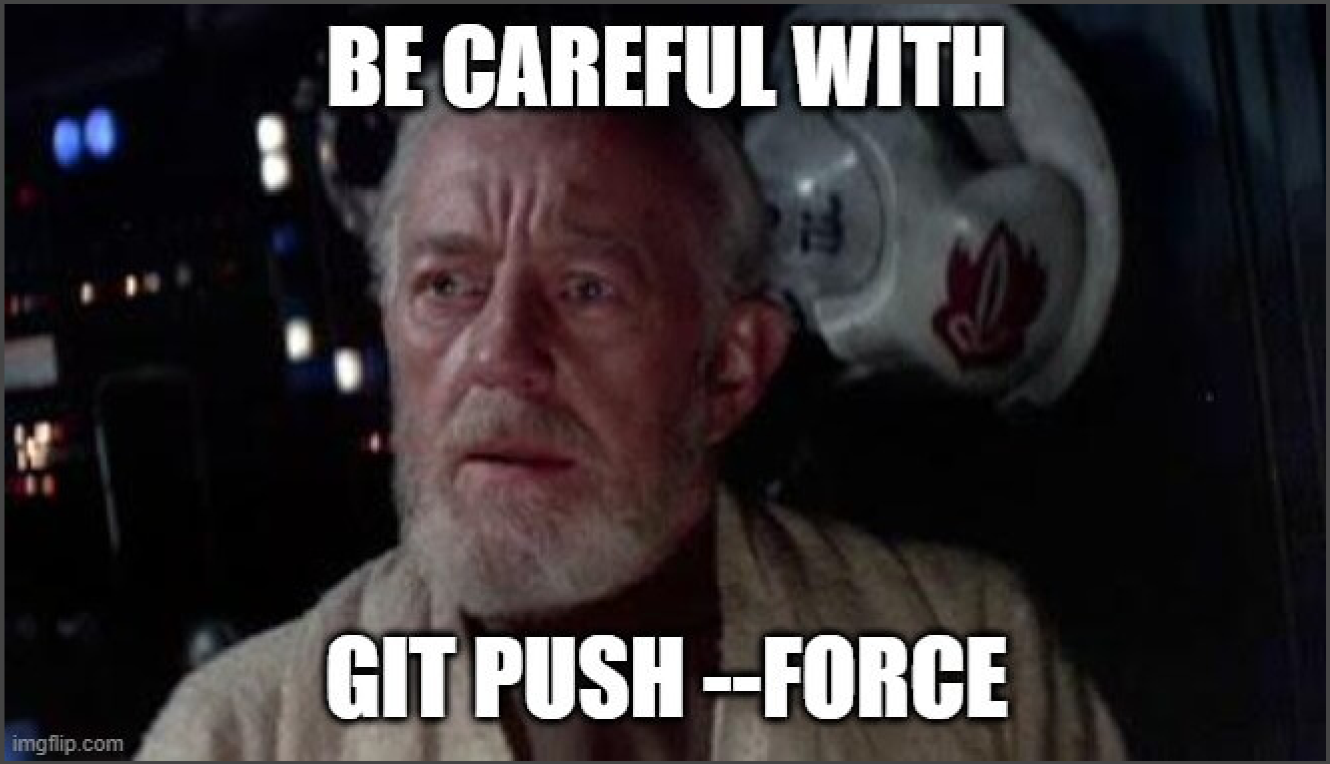
```
  GNU nano 6.2                    /home/glenoci/projects/personal/my-killer-app/.git/rebase-merge/git-rebase-todo
pick 6251bf6 :construction: Add .gitignore
pick a5c474a :sparkles: New feature: Adds Mapper Class
pick 68e447e :sparkles: New feature: Add My controller
pick 0bf36c7 :hammer: Refactor: adds mapper as a dependency of my controller
pick d749bda :hammer: Refactor: Add myMethod to Mapper necessary to controller
pick 3aa9ba1 :rotating_light: Test: Mapper
pick bd1df8a :rotating_light: Test: Controller

# Rebase di 3e46bc8..bd1df8a su 3e46bc8 (7 comandi)
#
# Comandi:
# p, scegli <commit> = usa commit
# r, reword <commit> = usa commit, ma modifica il messaggio di commit
# e, edit <commit> = usa commit, ma fermati per modificare
# s, squash <commit> = usa il commit, ma si fonde con il commit precedente
# f, correzione [-C | -c] <commit> = come "squash" ma mantiene solo il precedente
#                    messaggio di registro di commit, a meno che non venga utilizzato -C, nel qual caso
#                    mantieni solo il messaggio di questo commit; -c è uguale a -C ma
#                    apre l'editor
# x, exec <command> = esegui il comando (il resto della riga) usando la shell
# b, break = fermati qui (continua rebase più tardi con 'git rebase --continue')
# d, drop <commit> = rimuovi commit
# l, label <label> = etichetta HEAD corrente con un nome
# t, reset <label> = reimposta HEAD su un'etichetta
# m, unisci [-C <commit> | -c <commit>] <etichetta> [# <una riga>]
# . creare un commit di unione utilizzando i commit di unione originali
# . messaggio (o l'oneline, se non era presente alcun commit di unione originale
# . specificato); usa -c <commit> per riformulare il messaggio di commit
#
# Queste linee possono essere riordinate; vengono eseguiti dall'alto verso il basso.
#
# Rimuovendo una riga da qui IL COMMIT CORRISPONDENTE ANDRÀ PERDUTO.
#
# Ciò nonostante, se rimuovi tutto, il rebase sarà annullato.
#
```

3.Rebase -i

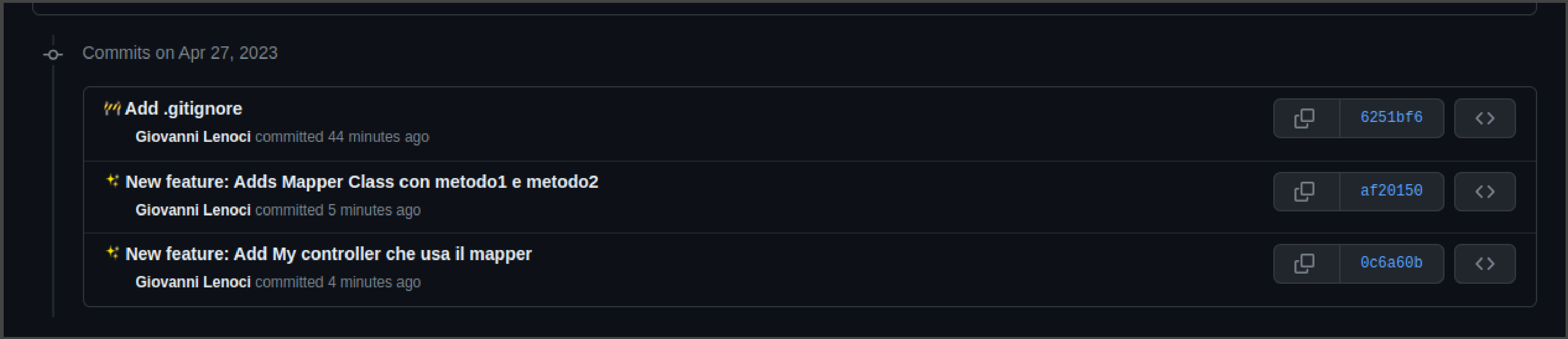
```
~/projects/personal/my-killer-app on my-new-branch ✓ > git rebase -i HEAD~7                                    [15:27:26]
```

1

4.Push --force



5. A better history #2



A new history

Git Workflows

Centralized workflow

Best suited for small teams transitioning from a centralized version control system like SVN. All team members work on a single branch, usually main, and push their changes directly to the central repository.

Forking workflow

Commonly used in open-source projects, this workflow allows external contributors to work without direct access to the main repository. Developers create a fork (a personal copy) of the main repository, make changes in their fork, and then submit a pull request or merge request to have their changes integrated into the main repository.

Git flow workflow

This workflow is best for projects with a structured release cycle. It introduces two long-lived branches: main for production-ready code and develop for integrating features. Additional branches like feature, release, and hotfix are used for specific purposes, ensuring a strict and organized development process.

Feature branch workflow

Developers create separate branches for each feature or bugfix, keeping the ‘main’ branch stable. When a feature is complete, the developer submits a pull request or merge request to integrate the changes back into the main branch after a code review.

Bibliography

- Is now git legit - Pauline Vos
- Conventional commits
- Make Atomic Git Commits
- Emoji for conventional commit
- My proposal for conventional commits
- Prepare commit message bash script
- Git workflows