

# Git 2 condivisione remota

---

[risorse.vercel.app/lab/2024](https://risorse.vercel.app/lab/2024)

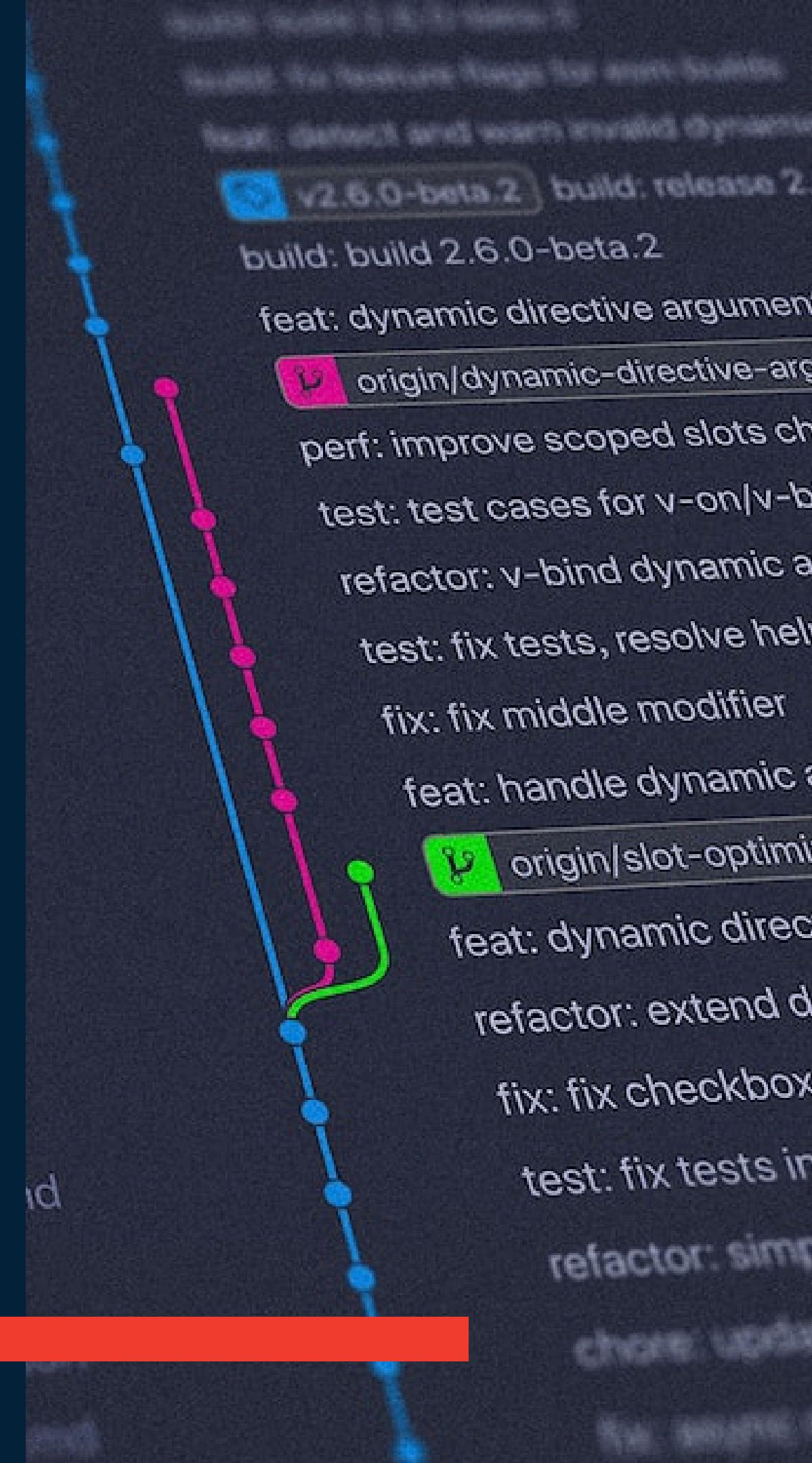
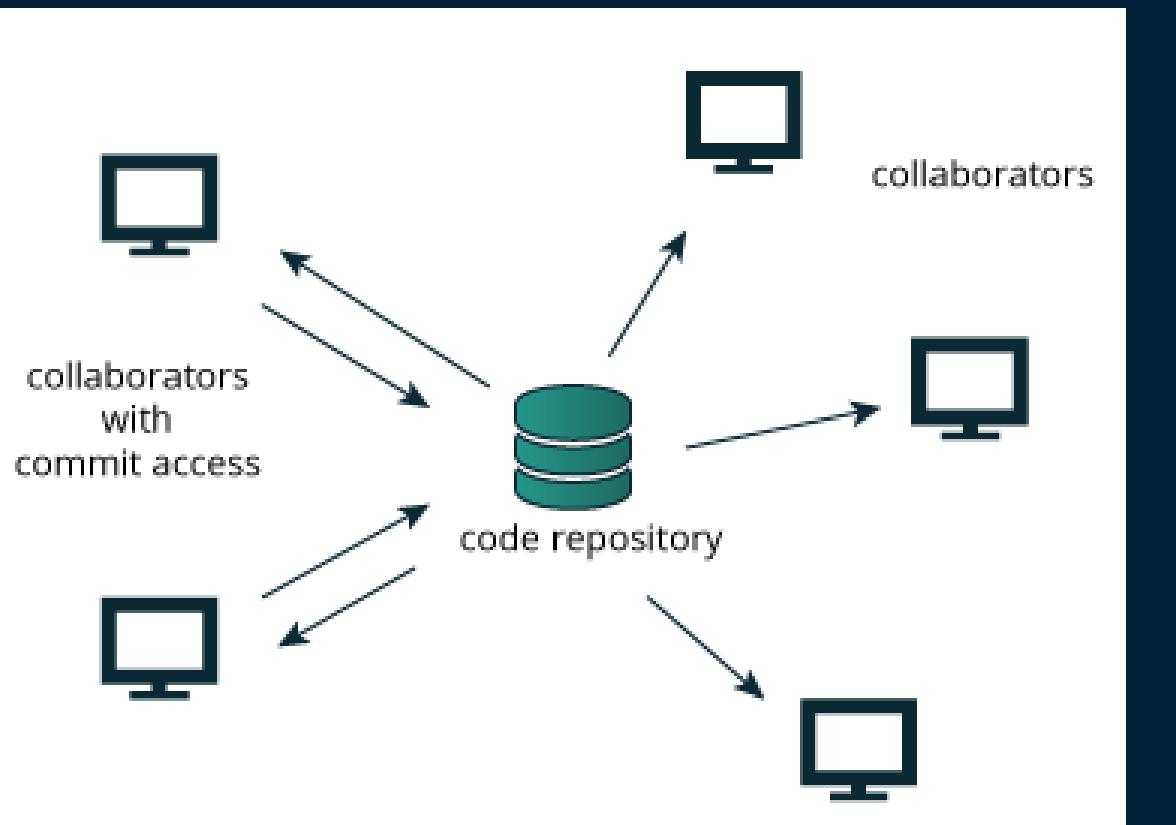
---

Alice Benatti, Samuele Musiani



# Cos'è git?

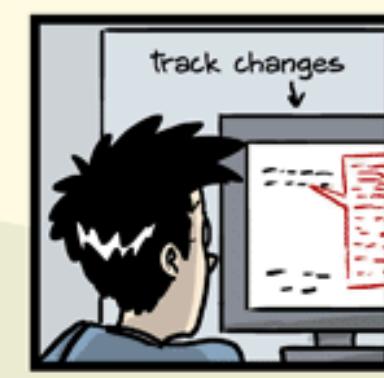
- ✓ • **sistema di versionamento** = tenere traccia delle versioni dei file, ottenendo una cronologia di tutte le modifiche
- **distribuito** = chiunque ha una copia completamente locale della *repository*



# Perché usarlo?

- ✓ • Cronologia trasparente su tutti i cambiamenti dei file
- ✓ • Ripristinare subito un qualsiasi **stato precedente**
- Collaborare a copie dinamiche diverse dello stesso progetto

"FINAL".doc



JORGE CHAM © 2012

# Ambiente di lavoro

- 1.1. Account dipartimentale
    - a. client ssh
  - 1.2. Git installato sul proprio dispositivo
- 
- 2. Utente su una piattaforma di hosting per progetti git  
[GitHub, GitLab, ...]



# Installare git

## Linux con Ubuntu/Debian

```
$ sudo apt-get update  
$ sudo apt-get install git
```

[per altre distribuzioni [clicca qui](#)]

## Windows

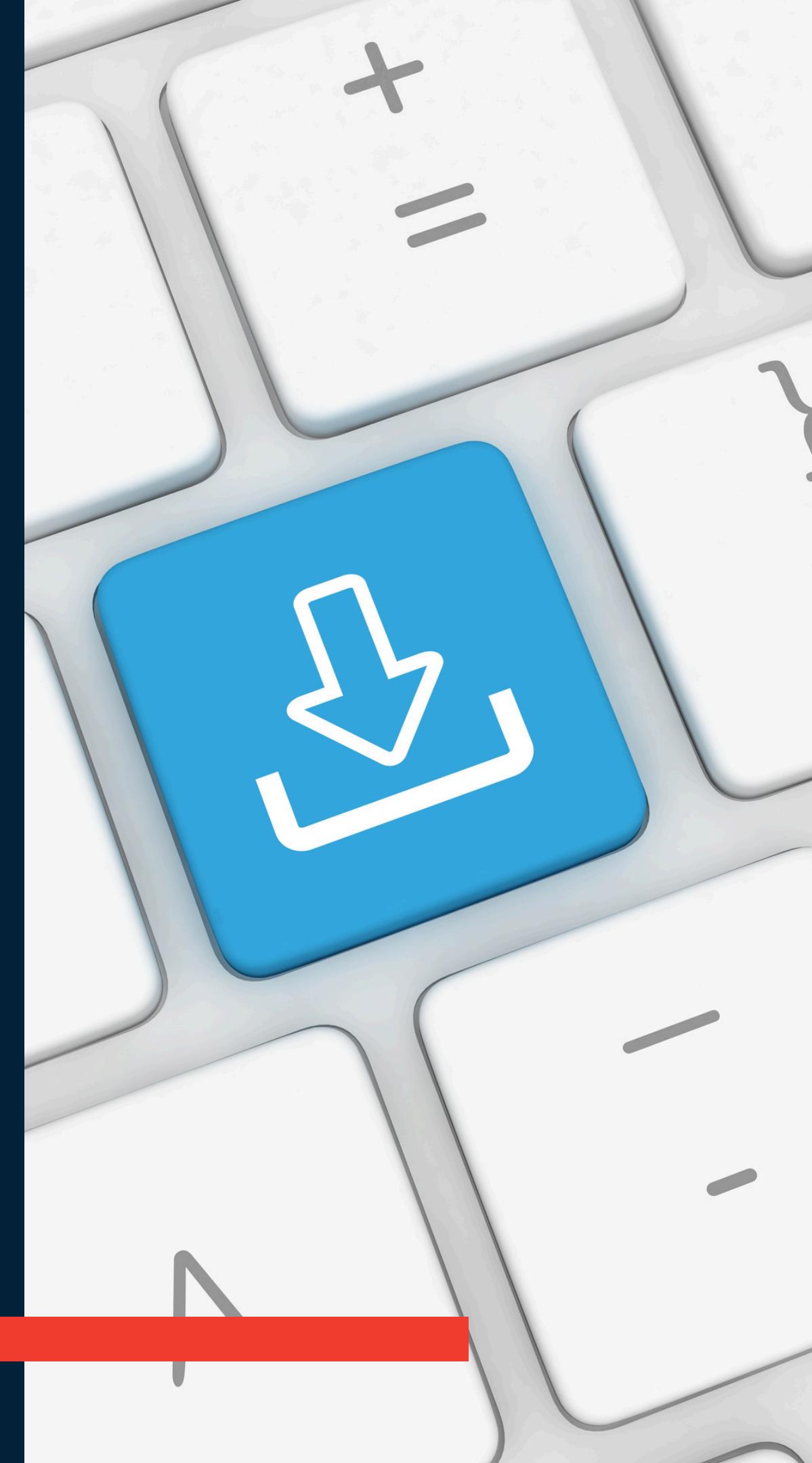
- via Command Line con *winget*

```
winget install --id Git.Git -e --source winget
```

- scaricare dal [link del sito ufficiale](#)

[Click here to download](#) the latest (2.42.0) 64-bit version of Git for Windows. This is the most recent [maintained build](#). It was released about 2 months ago, on 2023-08-30.

[Other Git for Windows downloads](#)



# Installare git

## MacOS

Ci sono diversi package manager che permettono di installare facilmente git, scegli quello che preferisci:

- ***Homebrew***

Installa [homebrew](#) se non lo hai già, e lancia il comando:

```
$ brew install git
```

- ***MacPorts***

Installa [MacPorts](#) se non lo hai già, e lancia il comando:

```
$ sudo port install git
```

- ***Xcode***

Apple fornisce un pacchetto binario Git con [Xcode](#).

[per ulteriori info seguite la [guida sul sito ufficiale](#)]



# Registrazione a GitHub

Se non hai già un account su GitHub registrati tramite il link:

[github.com/signup](https://github.com/signup)

- utilizza la mail che preferisci

Se sei già registrato fai il login su [github.com/login](https://github.com/login)

## Hai la chiave ssh?

1. generare una chiave ssh (se non l'avete già)

```
$ ssh-keygen -t ed25519 -C \
"nome.cognome@studio.unibo.it"
```

La avete già? Controlliamo con

```
$ ls ~/.ssh
```

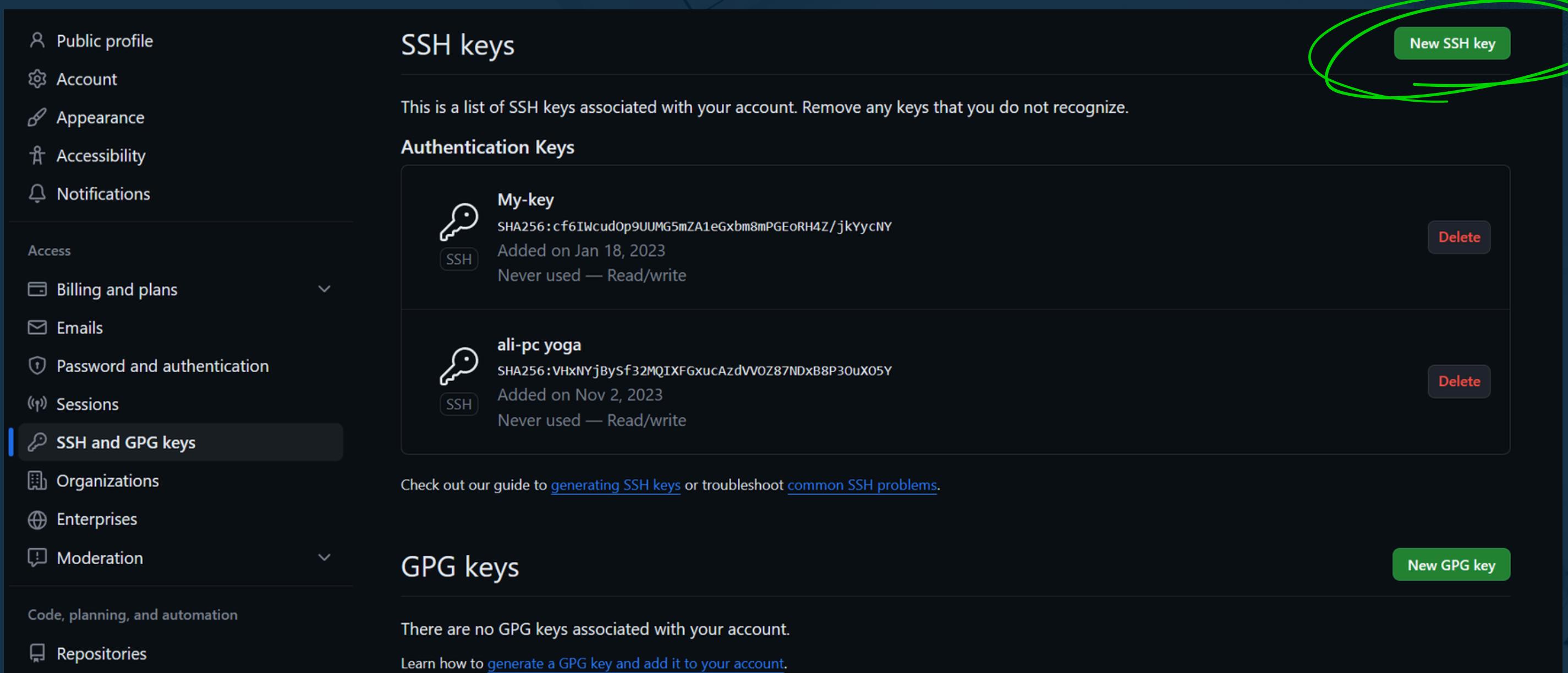
Stampiamo la chiave con  
e copiamola.

```
$ cat ~/.ssh/id_ed25519.pub
```

# Ambiente di lavoro

# Aggiungiamo la chiave ssh a GitHub

Vai su [github.com/settings/keys](https://github.com/settings/keys)



The screenshot shows the GitHub user settings interface, specifically the 'SSH and GPG keys' section. On the left, a sidebar lists various account management options like 'Public profile', 'Account', and 'SSH and GPG keys' (which is currently selected). The main area is titled 'SSH keys' and contains a heading 'Authentication Keys'. It lists two existing SSH keys: 'My-key' and 'ali-pc yoga', each with its SHA256 fingerprint, date added, and access rights (Never used — Read/write). At the top right of this section is a green button labeled 'New SSH key'. Below the SSH keys section is a 'GPG keys' section which states 'There are no GPG keys associated with your account.' and provides a link to generate one.

SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

**Authentication Keys**

Key Name	SHA256 Fingerprint	Added On	Access Rights	Action
My-key	SHA256:cf6IWcud0p9UUMG5mZA1e6xbm8mPGEoRH4Z/jkYycNY	Added on Jan 18, 2023	Never used — Read/write	Delete
ali-pc yoga	SHA256:VHxNYjBySf32MQIXFGxucAzdVV0Z87NDXB8P30uX05Y	Added on Nov 2, 2023	Never used — Read/write	Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

**GPG keys**

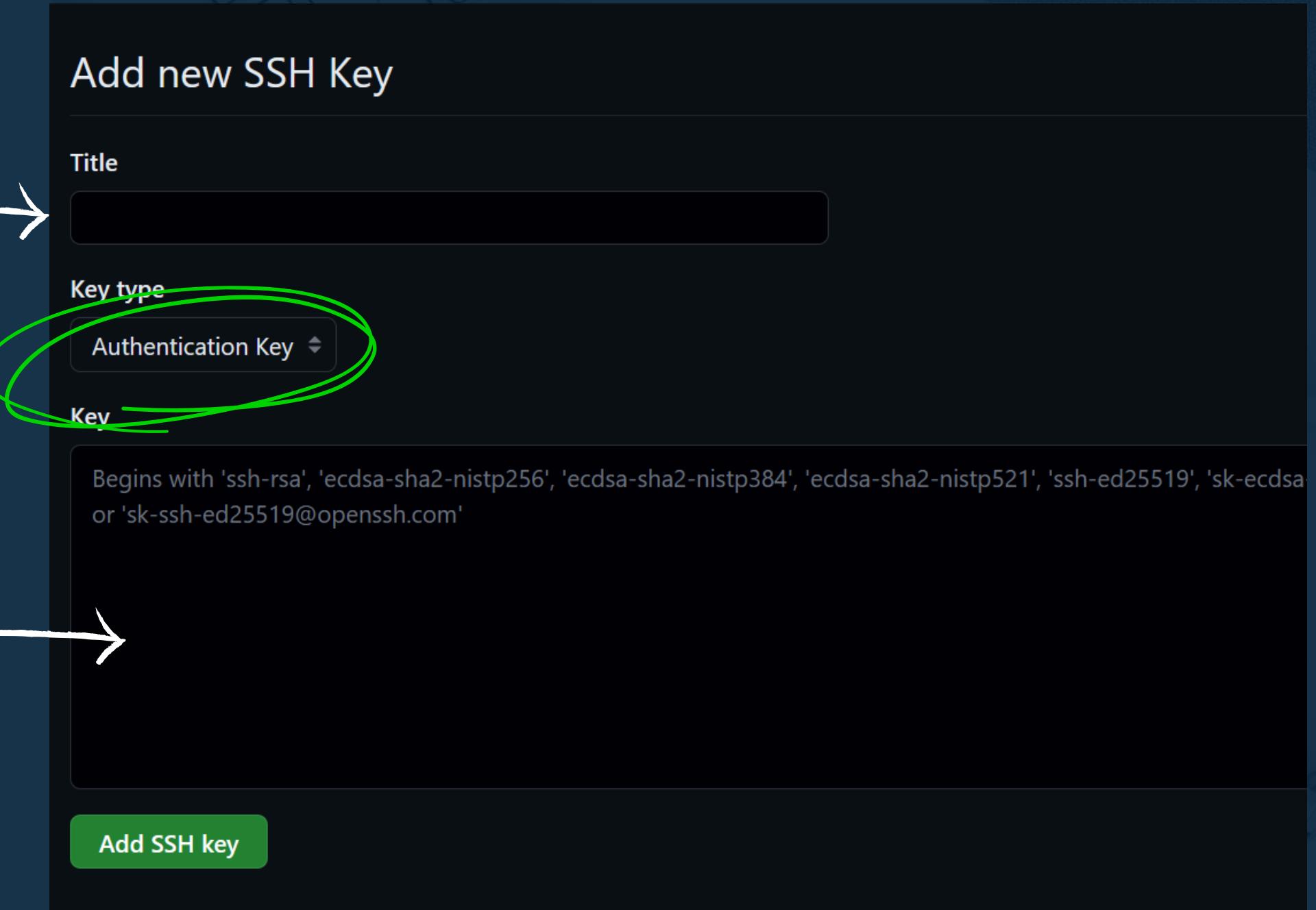
There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

New SSH key

## Aggiungiamo la chiave ssh a GitHub

Dai un titolo riconoscibile  
associato al pc che stai usando



Incolla qui la chiave ssh

Verifichiamo con il comando

```
$ ssh -T git@github.com
```

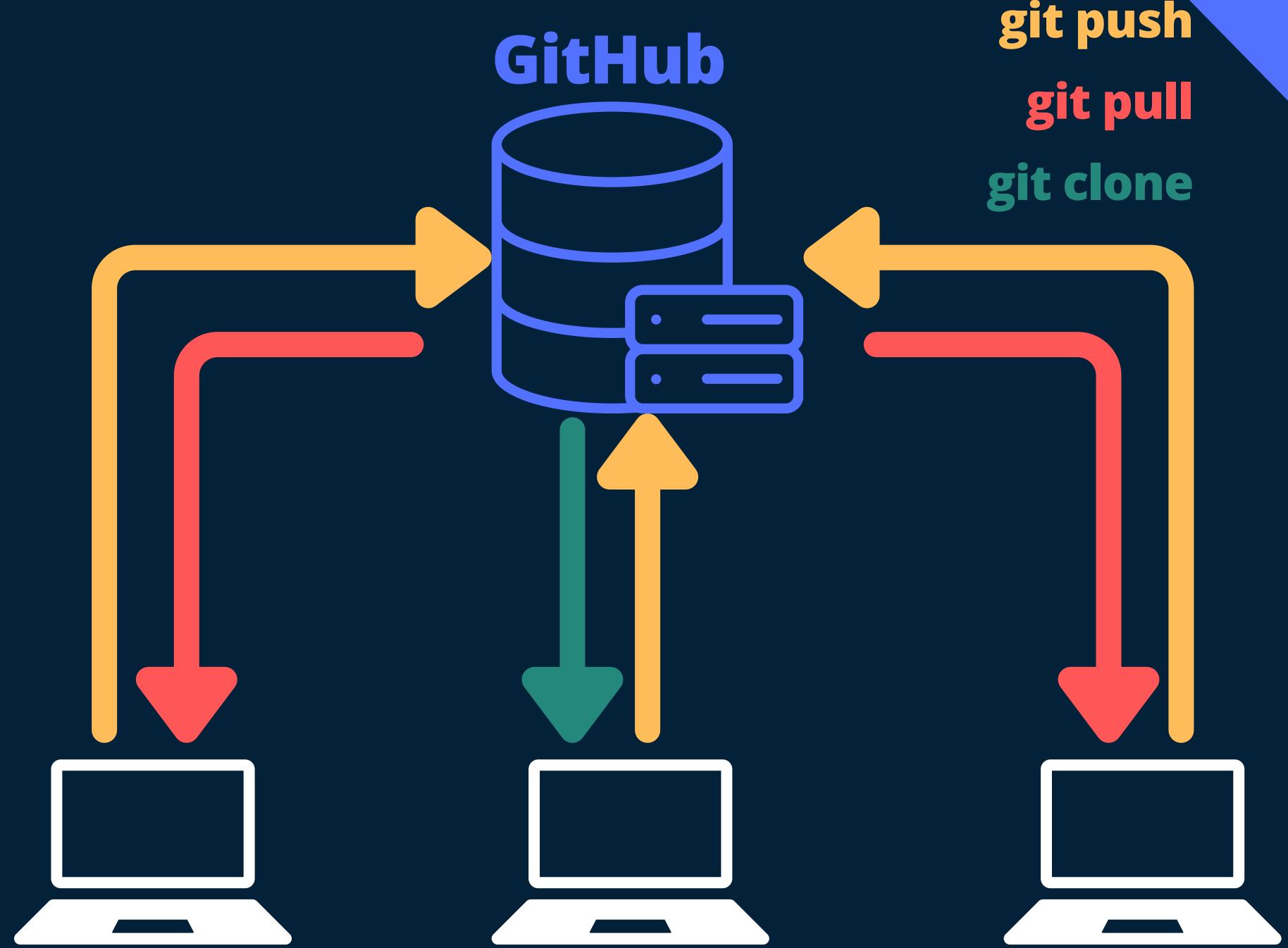
# Come funziona?

GitHub è un server che tiene una copia remota della repository.

Per modificarla è necessario **copiarsela** in locale.

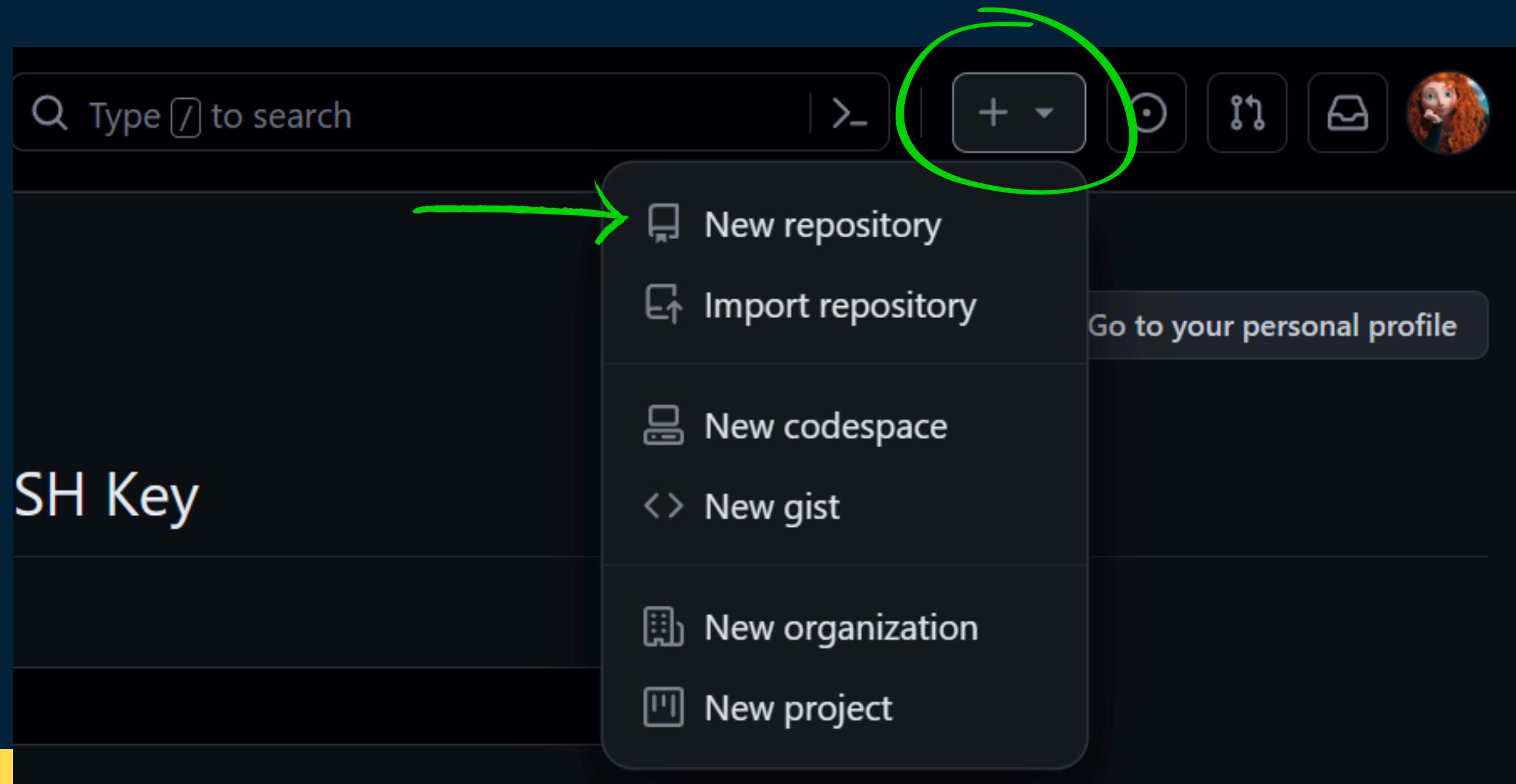
Una volta registrate le modifiche con **git commit**, è necessario **aggiornare** la repository remota.

Dopo che il server viene modificato, serve **aggiornare** la propria copia locale.



# *Creiamo una repository su GitHub*

**repository è un progetto mantenuto con git**



# *Creiamo una repository su GitHub*

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (\*).

Owner \*      Repository name \*

 nopec42154 /

Great repository names are short and memorable. Need inspiration? How about [studious-spoon](#) ?

Description (optional)

 Public  
Anyone on the internet can see this repository. You choose who can commit.

 Private  
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file  
This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

(i) You are creating a public repository in your personal account.

**Create repository**

# *Creiamo una repository su GitHub*

Copiamo l'ssh



# git clone

**Copiamo in locale la repository remota su GitHub**

```
$ git clone <url_repository_remota>
```

Creiamo un README.md

```
$ nano README.md
```

e scriviamo del contenuto.

Aggiungiamo alla *staging area*

```
$ git add README.md
```

Creiamo un commit

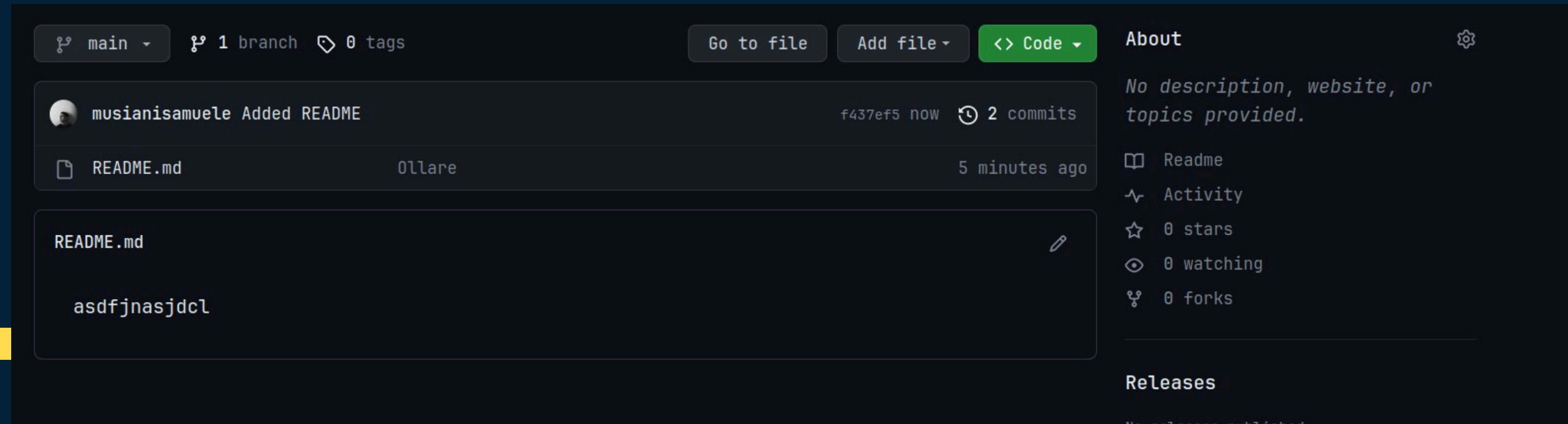
```
$ git commit -m "add: README.md"
```

# git push

**Spedisce i commit alla repository remota su GitHub**

```
$ git push
```

Aggiornando la pagina della repository su GitHub troverete le modifiche e il vostro commit:





# GitHub

*hosting service per git*

# GitHub



- è un **servizio**
- hostato su **web**
- ha anche un'**interfaccia grafica**
- ha uno spazio per caricare copie di repository Git
- offre altre funzionalità:
  - VCS
  - Gestione Codice sorgente
    - pull request
    - issues
    - ...

# Git



- è un **software**
- installato **localmente** sulla macchina
- solo con **command line**
- gestisce diverse versioni di modifiche in **una repository**
- offre altre funzionalità:
  - VCS
  - Gestione Codice Sorgente basilare

# Non solo GitHub

Utilizzeremo GitHub in quanto è il più popolare e csunibo si trova lì.

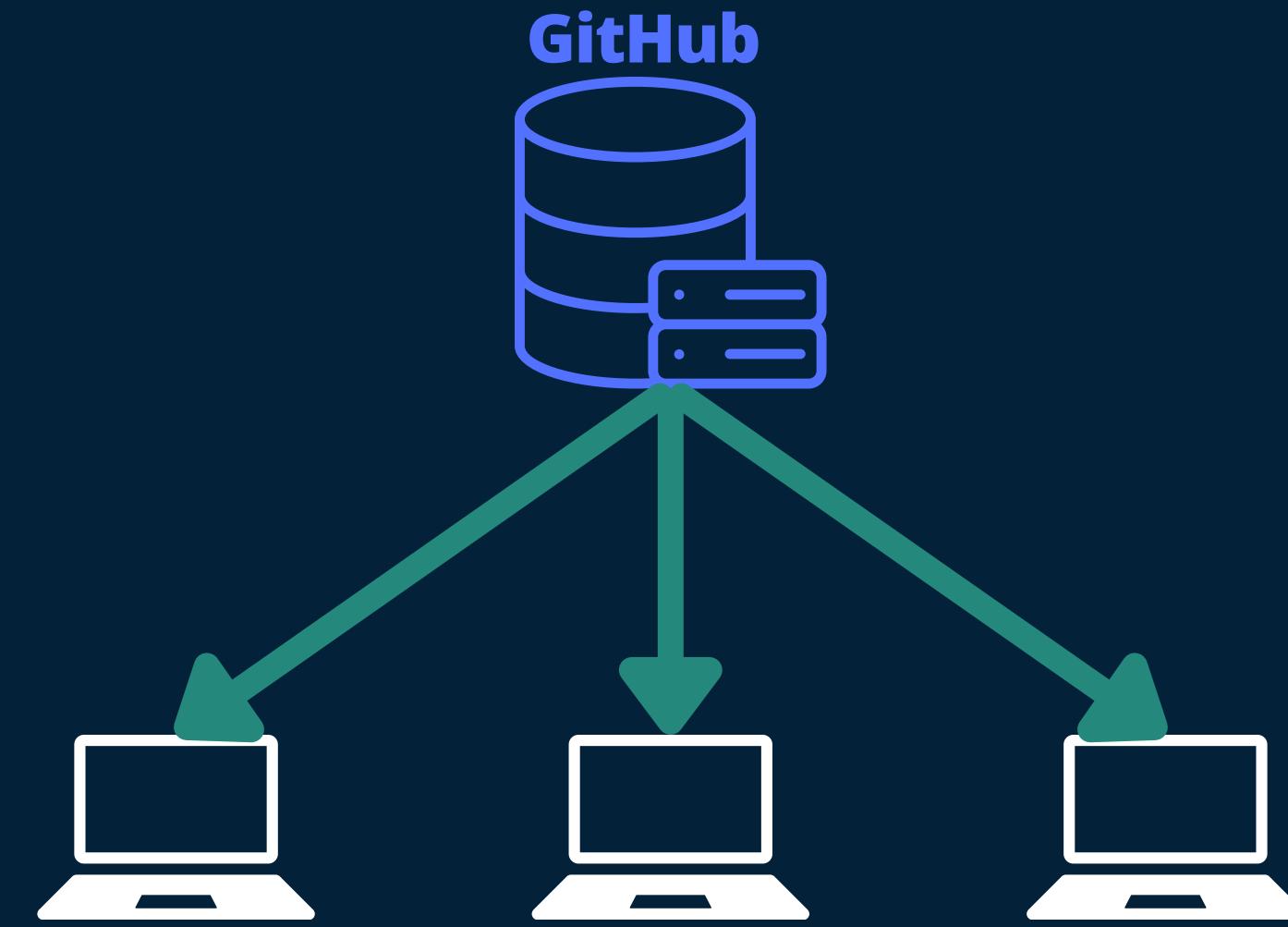
Bitbucket	GitHub	GitLab	So, what does it mean?
Pull Request	Pull Request	Merge Request	In GitLab a request to merge a feature branch into the official master is called a Merge Request.
Snippet	Gist	Snippet	Share snippets of code. Can be public, internal or private.
Repository	Repository	Project	In GitLab a Project is a container including the Git repository, discussions, attachments, project-specific settings, etc.
Teams	Organizations	Groups	In GitLab, you add projects to groups to allow for group-level management. Users can be added to groups and can manage group-wide notifications.

# git clone

01

Copia di una repository remota in una directory locale

```
$ git clone <url_repository_remota>
```

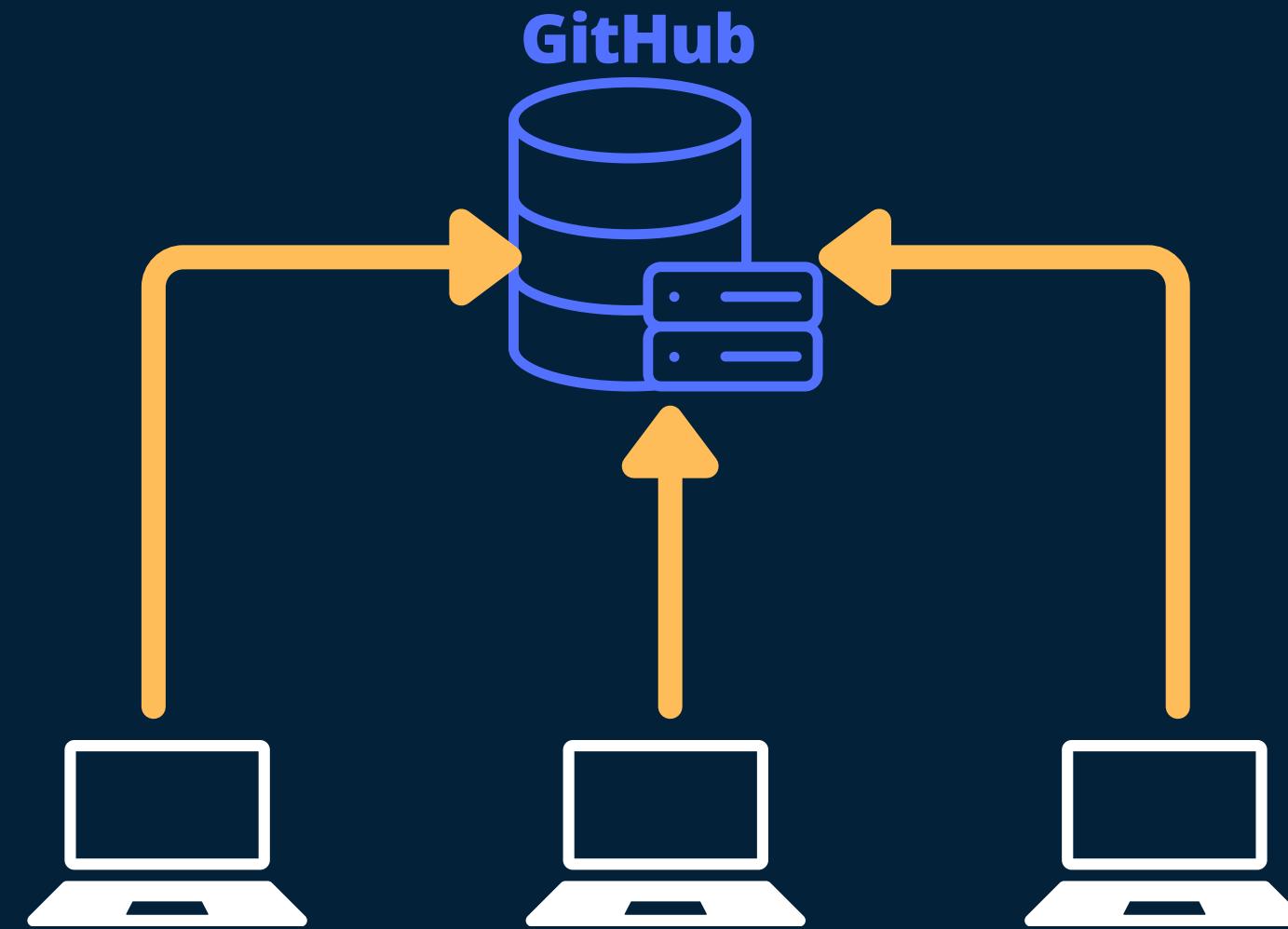


# git push

02

Inviare i commit registrati alla repository remota

```
$ git push
```

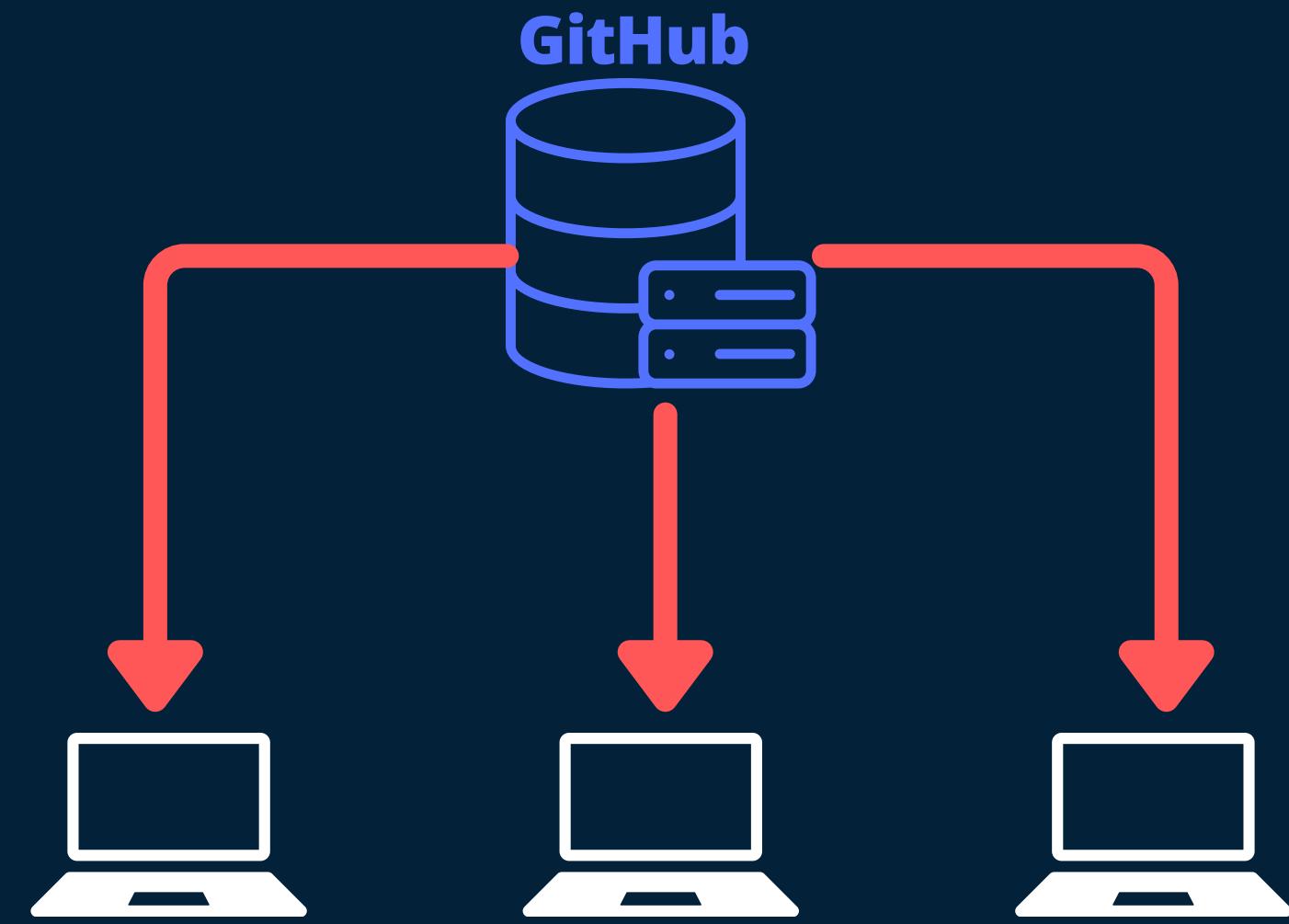


# git pull

03

Aggiorniamo al repository locale di modifiche remote

```
$ git pull
```



# 04

Creiamo una branch “*feature*” e ci spostiamo sopra

```
$ git switch -c feature
```

Creiamo un nuovo file e inseriamo qualcosa

```
$ nano HelloWorld.cpp
```

Aggiungiamo alla staging area il file e committiamo

```
$ git add HelloWorld.cpp  
$ git commit -m "feat: new HelloWorld"
```

Aggiorniamo la repository remota con le nuove modifiche:

**Inviamo i commit**

```
$ git push
```

**Creiamo la branch anche in remoto**

```
$ git push origin branch
```

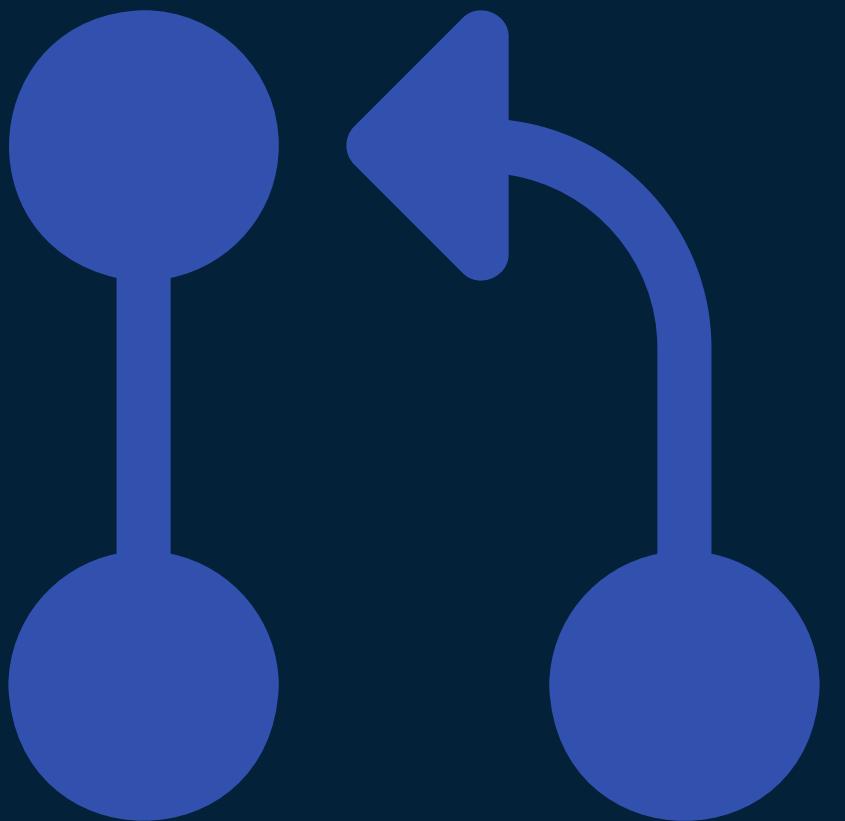
Verifichiamo cosa sta succedendo graficamente: `$ git log --graph --all`

# Pull Request

Le pull request consentono di comunicare ad altri utenti le modifiche di cui è stato eseguito il push in una branch secondaria in una repository GitHub.

Una volta aperta una *pull request*, è possibile:

- discutere
- rivedere le potenziali modifiche con i collaboratori
- aggiungere commit di follow-up prima che le modifiche vengano unite nella branch main.



In pratica è come fare un merge in futuro:

1. commentiamola con i nostri compagni
2. servono altre modifiche prima del merge?
3. mergiamo

# 05

Andiamo su GitHub all'interno della nostra repository

The screenshot shows the GitHub interface for a repository. At the top, there is a navigation bar with links: Code, Issues, Pull requests (which is underlined in orange), Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation bar are several filters: 'Filters ▾' with a dropdown arrow, a search bar containing 'is:pr is:open', a 'Labels' button with a count of 9, a 'Milestones' button with a count of 0, and a prominent green button with white text that says 'New pull request'. A large green oval highlights this 'New pull request' button. Below these filters, there are two status counts: '0 Open' and '0 Closed'. The main content area features a large, light-gray rectangular box with a faint 'git pull' icon at the top center. Inside this box, the text 'Welcome to pull requests!' is displayed in bold. Below this, a descriptive paragraph reads: 'Pull requests help you collaborate on code with other people. As pull requests are created, they'll appear here in a searchable and filterable list. To get started, you should [create a pull request](#)'. The entire screenshot is set against a dark blue background.

## Creiamo la Pull Request e aggiungiamo dettagli:

## Comparing changes

**Controlliamo di usare la nostra branch e di usare la giusta destinazione**

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.

၃

## Add a title

## Create File

## **Usiamo un titolo riassuntivo**

## Add a description

## Write

Preview



Add your description here.

**Descriviamo al meglio quali modifiche sono state effettuate,  
motivandole**

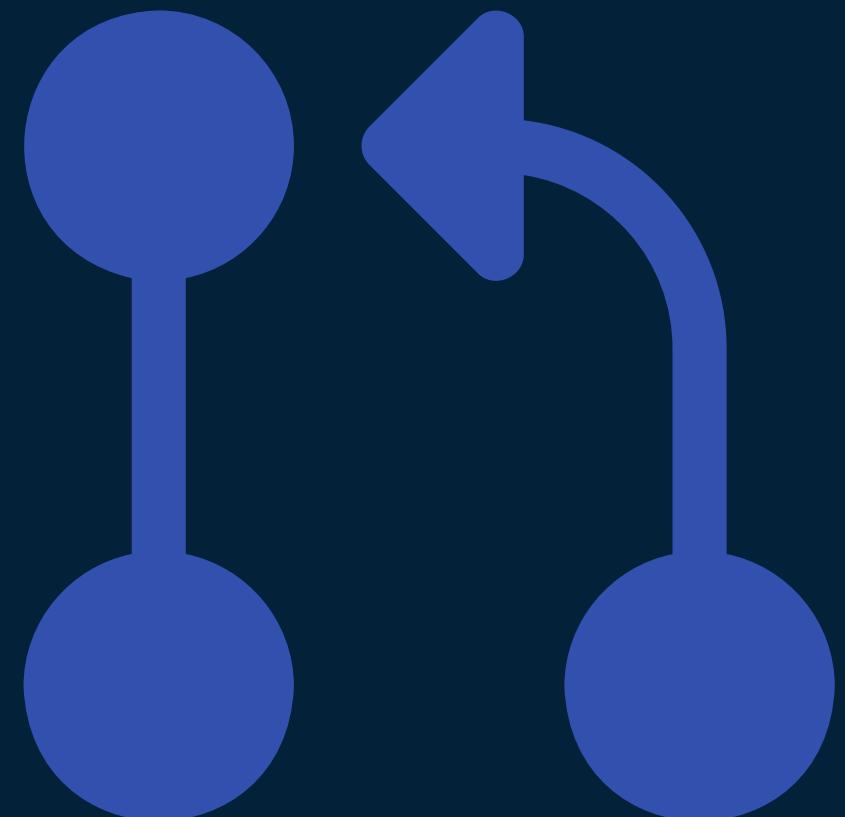
**Inserisci qui eventuali domande/dubbi da porre al team**

Create pull request

## Helpful resources

GitHub Community Guidelines

A questo punto eventuali commit ulteriori sulla branch in cui abbiamo fatto  
*pull request* verranno automaticamente aggiunti alla PR



# Issues

Permettono di tracciare idee, feedback, tasks, bug...  
da sviluppare nella nostra repository.

Permettono di:

- discutere e commentare l'argomento
- assegnare/dividere il lavoro nel team
- tenere uno storico

In pratica sono strutturate come le PR, ma non si basano su del codice già scritto/registrato su GitHub.



# Permessi

Dove abbiamo il permesso di modifica/fare *git push*?

- in repo create da noi
- in repo in cui siamo stati aggiunti come collaboratori
- se faccio parte di un'organizzazione e il mio team è stato abilitato ai permessi
- se sono proprietario di un'organizzazione

Le *issue* possono essere create da chiunque solitamente.



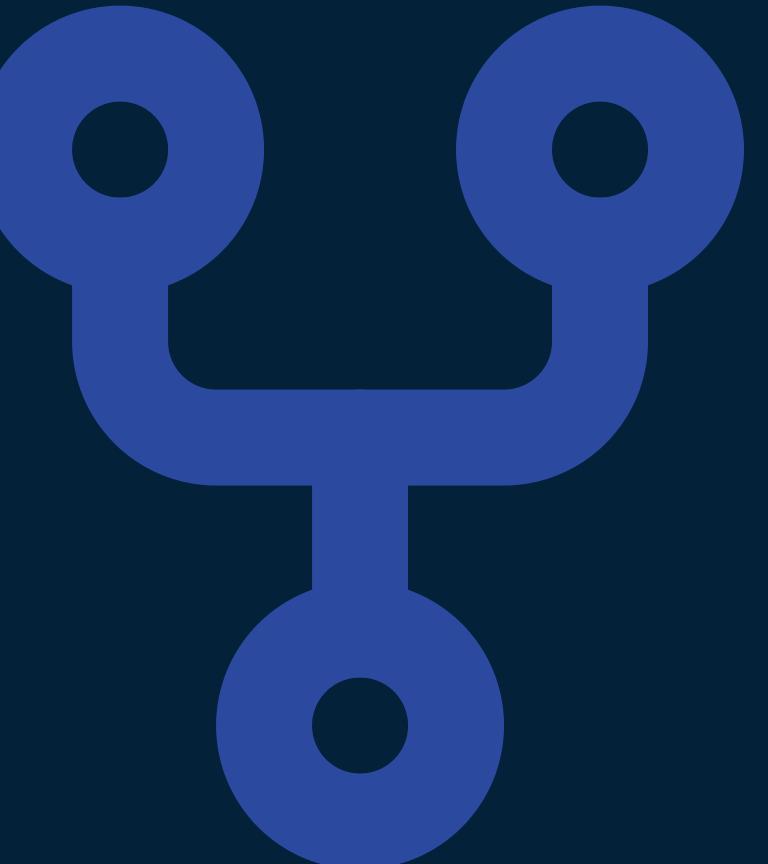
Per contribuire ad una repository in cui non abbiamo i permessi come [csunibo/lab-git2](#)  
usiamo il meccanismo delle ***fork***

# Fork

Una *fork* è un nuovo repository che condivide il codice e le impostazioni di visibilità con il repository originale.

Si usano per:

- proporre modifiche in una repository di qualcun’altro [su cui non si ha i permessi di scrittura]
- usare una repository come base per sviluppare un proprio progetto



# 08

## Andiamo su GitHub nella repository da forkare

The screenshot shows a GitHub repository page for 'lab-git2'. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation, the repository name 'lab-git2' is displayed with a 'Public' badge. A green circle highlights the 'Fork' button in the top right corner of the header, which has a value of '0'. A large green arrow points from the bottom left towards this 'Fork' button. The main content area shows a commit by 'ali-benny' with the message 'Initial commit' made 13 minutes ago. A file named 'README.md' is listed with the message 'Initial commit' and a timestamp of 13 minutes ago. On the right side, there is a sidebar with options like 'Readme', 'Activity', '0 stars', '1 watching', '0 forks', and 'Report repository'. The title of the repository is 'Laboratorio fra pari di Git2 - condivisione remota'.

09

Ci aprirà una pagina per creare una repository remota sul nostro profilo

**Create a new fork**

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

*Required fields are marked with an asterisk (\*).*

Owner \*      Repository name \*

ali-benny / lab-git2 

lab-git2 is available.

userà il nome originale della repo da forkare

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

Repository per il Laboratorio fra pari di Git2 - condivisione remota

Copy the main branch only

Contribute back to csunibo/lab-git2 by adding your own branch. [Learn more.](#)

**Attenzione!**

 Se vogliamo prendere anche le branch è da unflaggare

 ⓘ You are creating a fork in your personal account.

**Create fork**

Andiamo a clonare la repository forkata sul nostro profilo

```
$ git clone git@github.com:<nome_utente>/<nome_repo>.git
```

Facciamo le modifiche che vogliamo

Aggiungiamole alla staging area e committiamo

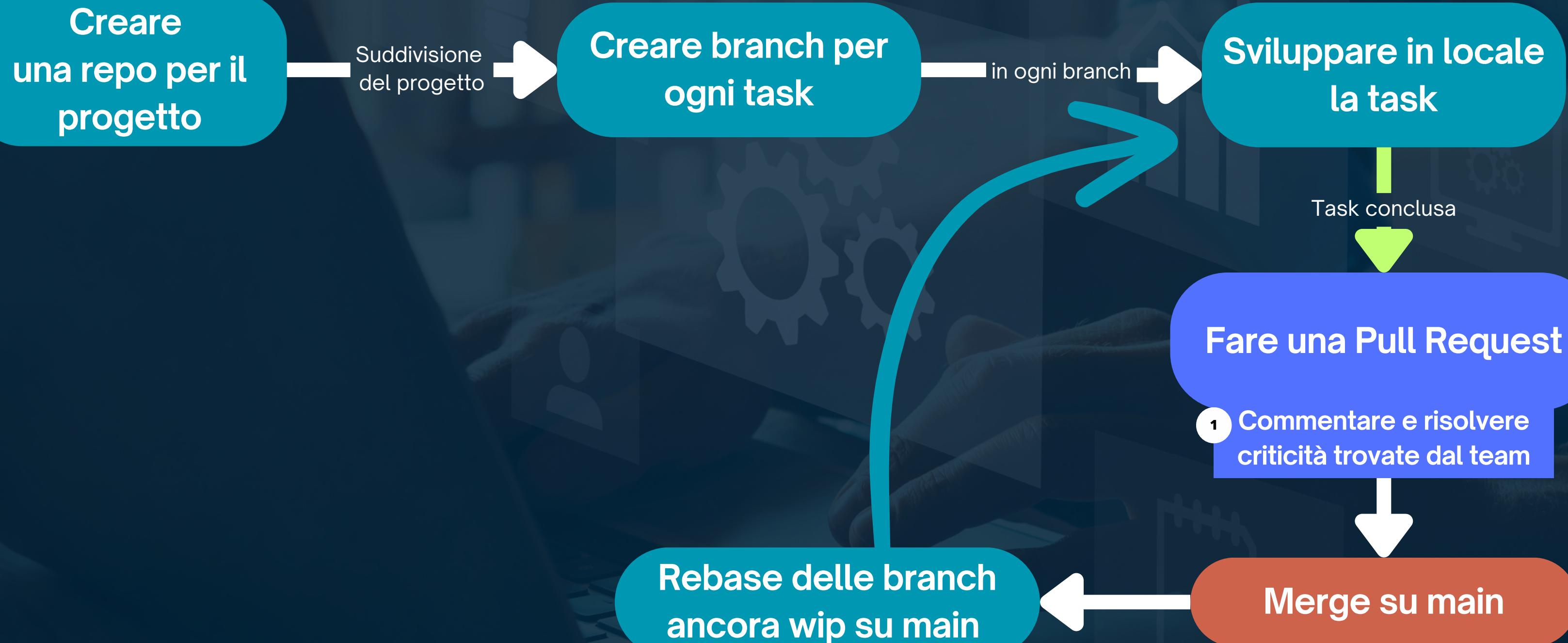
Aggiorniamo la repository nostra remota con le nuove modifiche:

```
$ git push
```

Vediamo su GitHub cos'è successo nella repository forkata [quella sul nostro profilo per intenderci]

Ci proporrà di fare una Pull Request sulla repository originale.

# Workflow in team su GitHub



# Conflitti

Abbiamo visto che si può lavorare su più branch in parallelo.

Al momento del merge/rebase di queste branch si possono presentare conflitti.

Il motivo è che si possono avere più versioni di uno stesso file

- git non sa quale delle due è quella giusta da salvare

È necessario risolverli prima di completare le operazioni di merge/rebase che si stavano eseguendo.

# Creiamo un conflitto

Creiamo una altra branch a partire da main

```
$ git switch -c <nuova_branch> main
```

Modifichiamo un file

Torniamo sulla branch main e modifichiamo lo stesso file in modo diverso

Facciamo il merge della nuova branch su main con

```
$ git merge <nuova_branch>
```

A questo punto dovremmo avere dei conflitti sul file modificato.

# Risolvere i conflitti

1. command line
2. vscode o altri IDE

# Conflitti command line

1. per vedere i file che sono in conflitto usiamo *git status*

Dovremmo vedere una cosa del genere:

```
$ git status  
[...]  
Unmerged path:  
    both modified: <nome_file_conflitto>
```

2. modifichiamo i file in conflitto con un editor di testo a scelta, git avrà aggiunto delle righe dove si trovano i conflitti in cui specifica quali parti vengono da quale versione:

```
<<<<<< HEAD  
Modifiche correnti  
=====  
Modifiche “incoming”  
>>>>> nome_branch
```

# Conflitti command line

A questo punto resta a noi scegliere quali righe tenere/modificare/eliminare.

Il file dovrà risultare senza le linee aggiunte da git :

```
<<<<< HEAD
```

```
Modifiche correnti
```

```
=====
```

```
Modifiche “incoming”
```

```
>>>>> nome_branch
```

*Conflitti presenti*

*Conflitti risolti*

```
Modifiche “incoming”
```

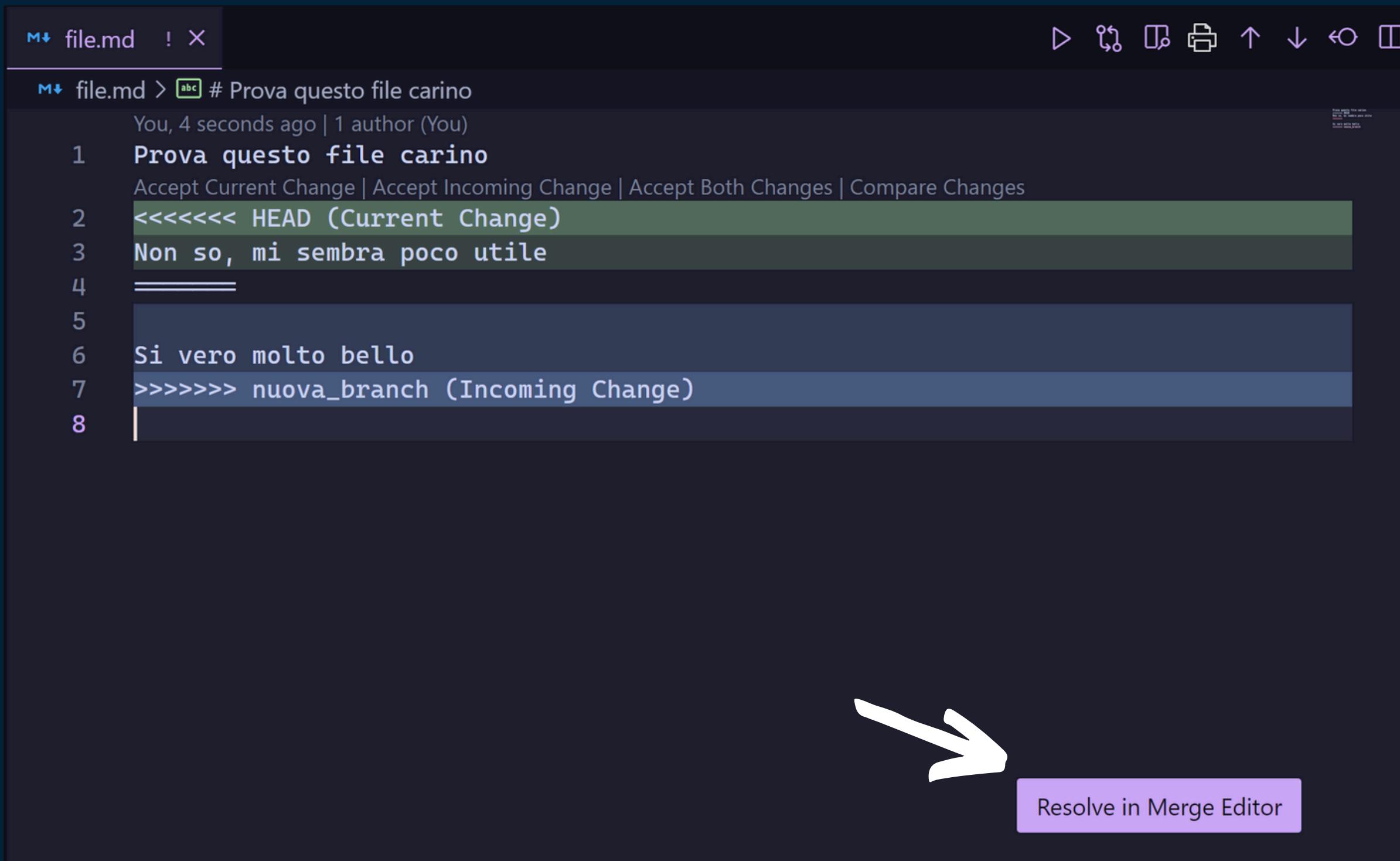
3. aggiungiamo alla staging area il file
4. in base all'azione che stavamo facendo eseguiamo:

Nel nostro caso usiamo: `$ git merge --continue`

`$ git rebase --continue`

# Conflitti vscode

Vscode ci aiuta e segna con un “!” i file che presentano un conflitto, aprendoli troviamo una cosa del genere:



```
file.md ! X
M+ file.md > abc # Prova questo file carino
You, 4 seconds ago | 1 author (You)
1 Prova questo file carino
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
2 <<<<< HEAD (Current Change)
3 Non so, mi sembra poco utile
4 ====
5
6 Si vero molto bello
7 >>>>> nuova_branch (Incoming Change)
8 |
```

Resolve in Merge Editor

# Conflitti vscode

Nell'editor dei conflitti visualizziamo il file nelle due versioni presenti che generano il conflitto

The screenshot shows the VS Code Merge Editor interface. At the top, it says "Merging: file.md ! X". Below that, there are two panes: "Incoming" (left) and "Current" (right).  
In the "Incoming" pane (branch `9b40130`, commit `nuova_branch`):

- Line 1: **Prova questo file carino** (highlighted in yellow)
- Line 2: **Si vero molto bello** (highlighted in green)

A context menu is open over line 1, with options: "Accept Incoming" (disabled), "Accept Combination (Incoming First)", and "Ignore".  
In the "Current" pane (branch `b46f770`, commit `main`):

- Line 1: **Prova questo file carino** (highlighted in yellow)
- Line 2: **Non so, mi sembra poco utile** (highlighted in green)

A context menu is open over line 1, with options: "Accept Current" (disabled), "Accept Combination (Current First)", and "Ignore".  
At the bottom, the "Result file.md" section shows:

- Line 1: **Prova questo file carino** (highlighted in yellow)

With the message: "You, 1 second ago • Uncommitted changes" and "No Changes Accepted".  
A large white arrow points from the bottom right towards the "Complete Merge" button, which is highlighted in purple.

Versione finale che si salverà in  
base alla risoluzione del conflitto

# Conflitti vscode

Facciamo un commit aggiungendo le modifiche del merge

The screenshot shows the VS Code interface with a dark theme. On the left, the Source Control sidebar displays a repository named 'repo\_test' with a 'main+' branch. A 'Merge branch 'nuova\_branch'' message is shown, and a purple button labeled 'Commit' is highlighted. In the center, the code editor shows a file named 'file.md' with the following content:

```
C: > Users > alice > Desktop > repo_test > file.md
1 1 Prova questo file carino
2 - Non so, mi sembra poco utile
2+ You, 4 minutes ago • edit file
3+ Si vero molto bello
```

The second line is marked as conflicted ('-'). On the right, a status bar shows icons for copy, paste, cut, print, and navigation.

# Usare Git vscode

Git log --graph --all su vscode

The screenshot shows the 'Commit Graph' view in VS Code for a repository named 'repo\_test'. The current branch is 'main'. A merge commit has been made from the 'nuova\_branch' into 'main'. The commit graph visualizes the history, showing the flow from the initial commit through various edits and finally the merge. The commit details are listed below the graph.

BRANCH / TAG	GRAPH	COMMIT MESSAGE	AUTHOR	CHANGES	COMMIT DAT...
✓ <input checked="" type="checkbox"/> main		Merge branch 'nuova_branch'	You		9 seconds... 3b5...
		file edited	You	<input type="checkbox"/> 1	3 minutes... b46...
		edit file useless	You	<input type="checkbox"/> 1	4 minutes... c3e...
<input type="checkbox"/> nova_branch		edit file	You	<input type="checkbox"/> 1	5 minutes... 9b4...
		Initial commit	You	<input type="checkbox"/> 1	6 minutes... 339...

# Flag speciali pt2

Scopri tu cosa fanno di magico ✨

```
$ git fetch
```

```
$ git fetch origin pull/<numero>/head:<nome_branch>
```

```
$ git push --force ⚡!
```

```
$ git stash -u
```

```
$ git diff --cached
```

```
$ git bisect
```

# Flag speciali

dello scorso incontro

\$ git add -p [<path>] aggiunge alla staging area solo alcune parti di un file specificato

\$ git rebase -i <hash\_code\_commit> fa rebase “*interattivo*” a partire da un specifico commit, modificando la history a piacimento

\$ git cherry-pick <hash\_code\_commit> prende il commit specificato e lo applica alla branch corrente

\$ git reset --hard <hash\_code\_commit> riporta la branch corrente allo stato del commit specificato cancellando tutte le modifiche successive ad esso.

\$ git reset --soft <hash\_code\_commit> riporta la branch corrente allo stato del commit specificato spostando tutte le modifiche successive ad esso nella staging area

\$ git merge --squash <branch\_da\_mergiare> fa un merge in cui tutti i commit della branch vengono uniti in un unico commit