Git 2
condivisione remota

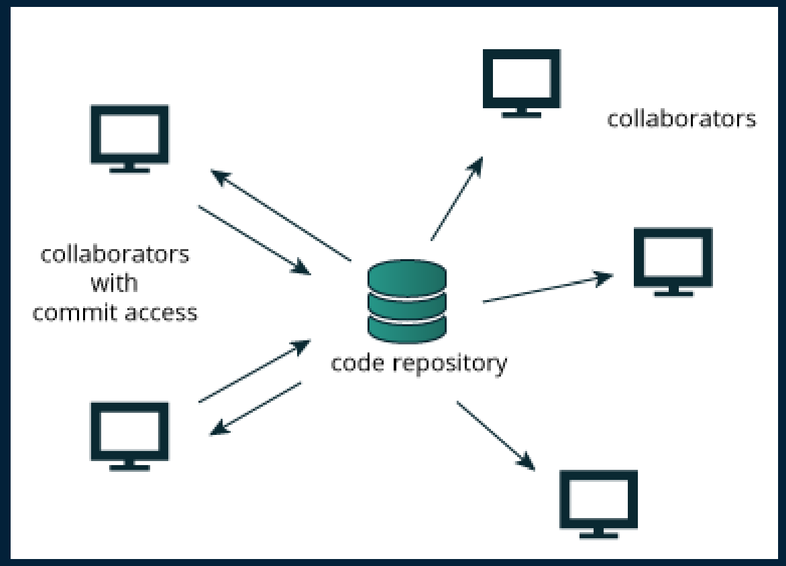
risorse.students.cs.unibo.it/lab/2024

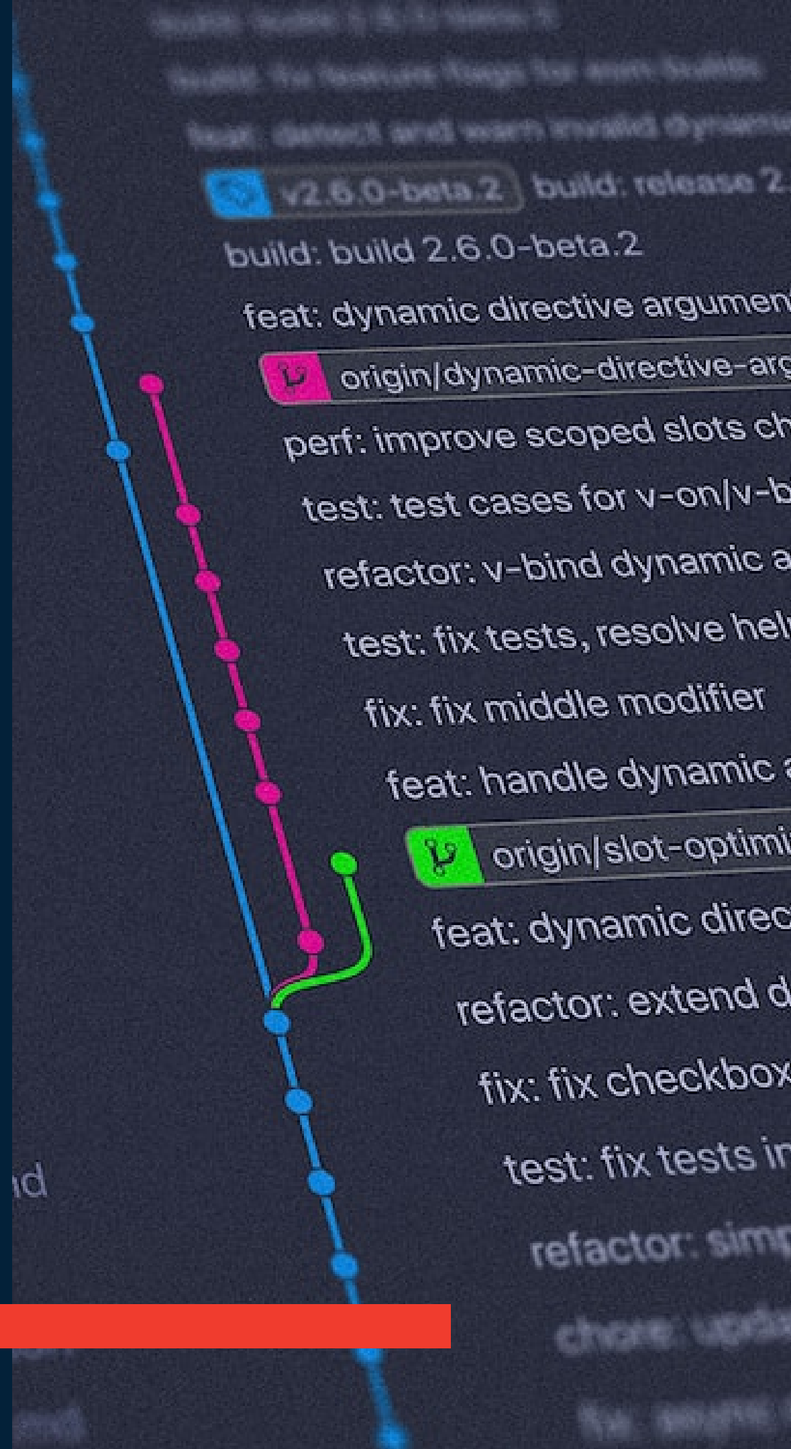
Alice Benatti, Samuele Musiani

Cos’è git?

- **sistema di versionamento** = tenere traccia delle versioni dei file, ottenendo una cronologia di tutte le modifiche

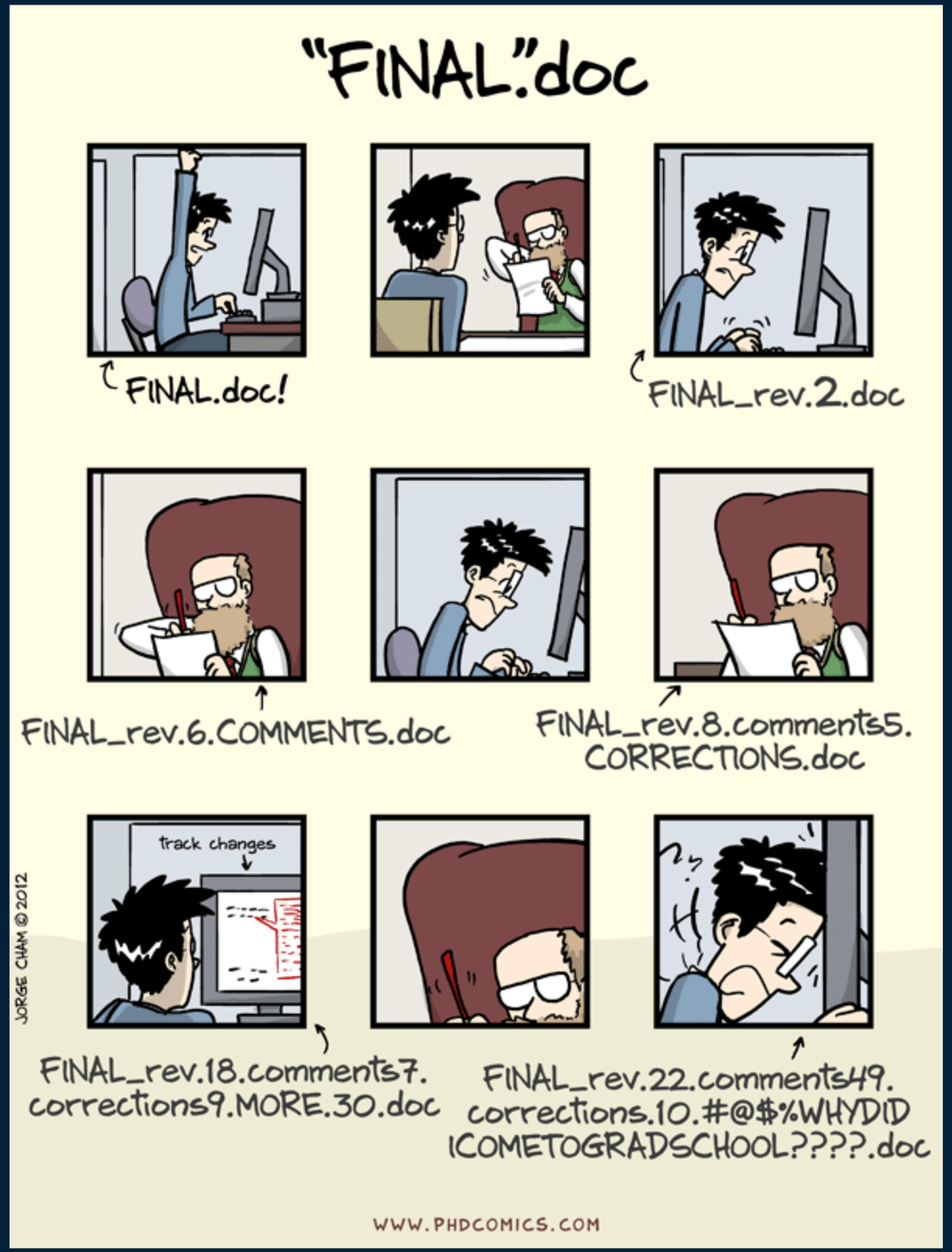
- **distribuito** = chiunque ha una copia completamente locale della *repository*





Perché usarlo?

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



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

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

- utilizza la mail che preferisci

Se sei già registrato fai il login su 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

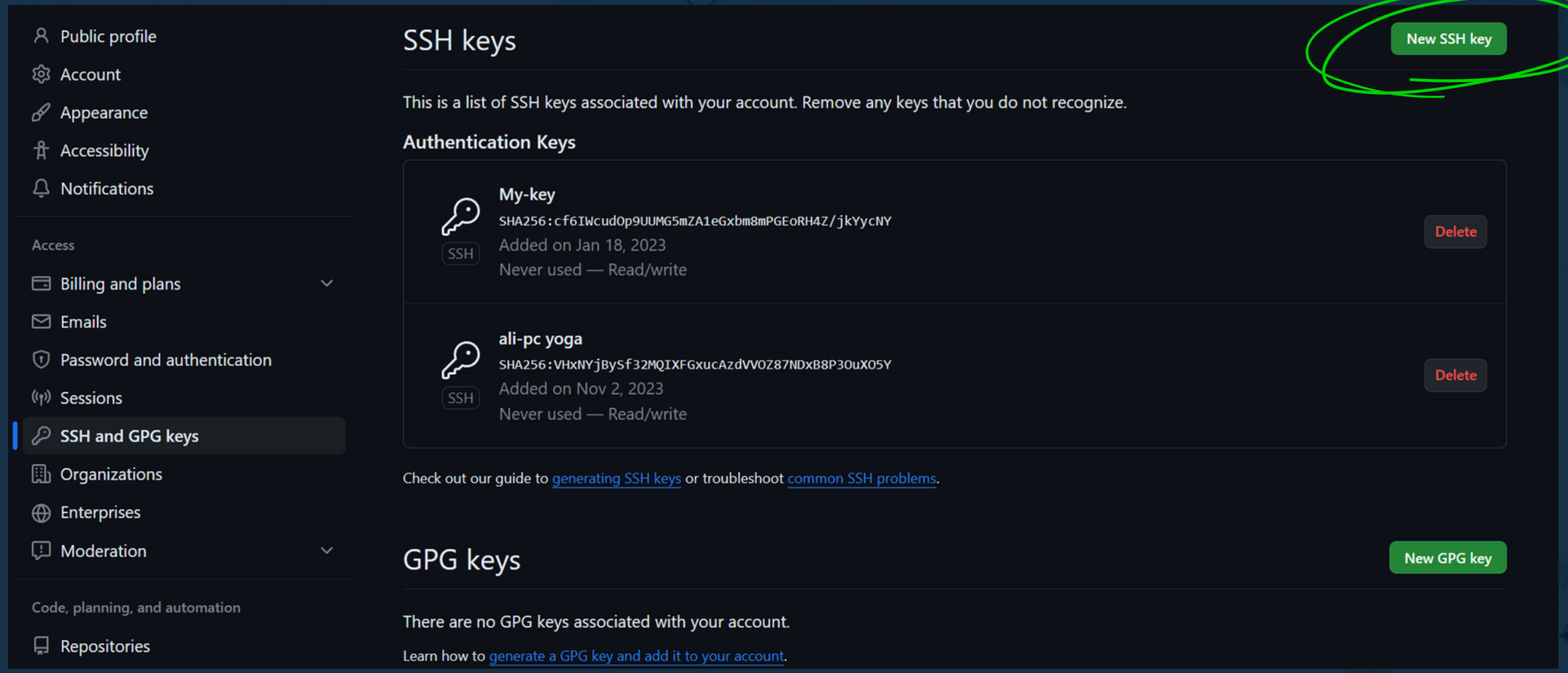
```
$ ls ~/.ssh
```

Stampiamo la chiave con
e copiamola.

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

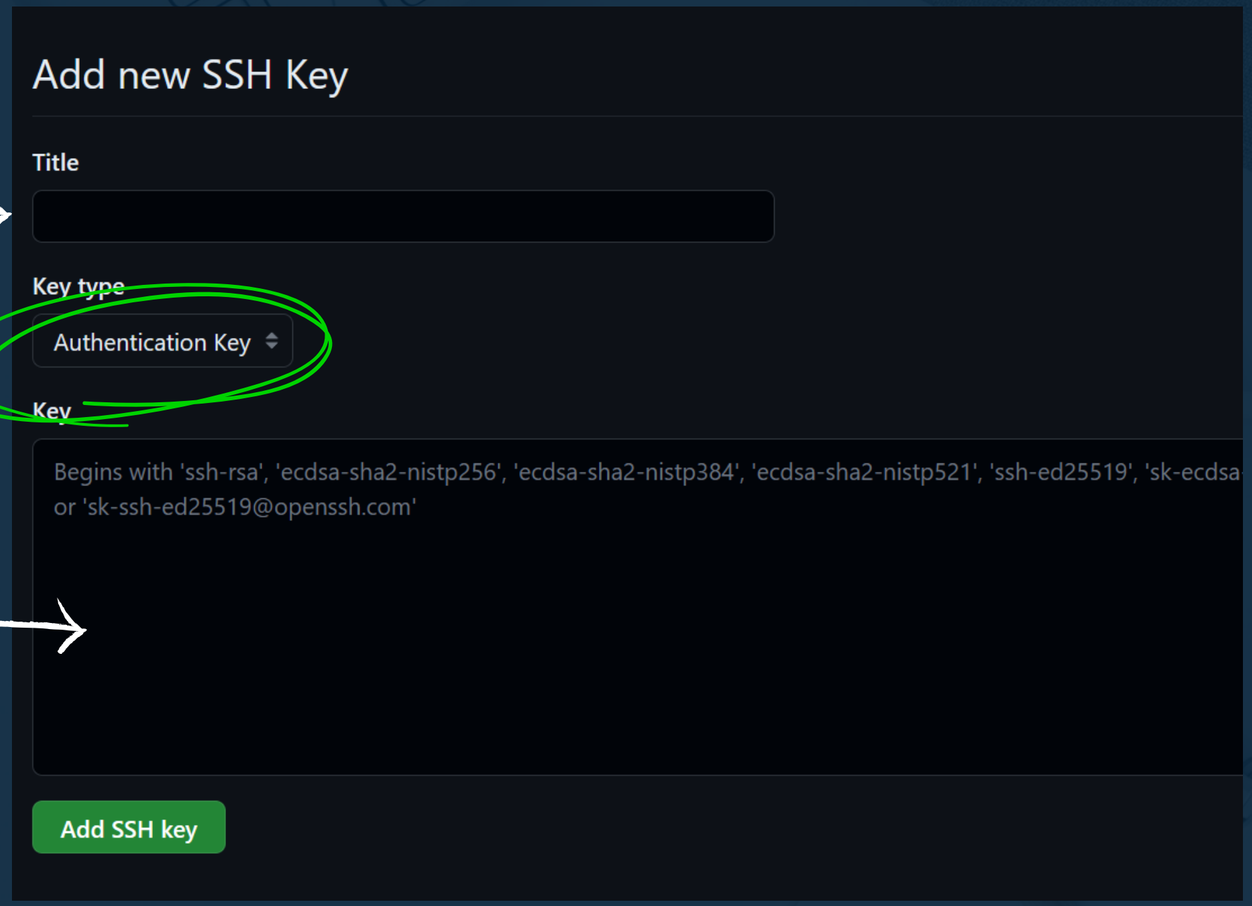
Aggiungiamo la chiave ssh a GitHub

Vai su github.com/settings/keys



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
```

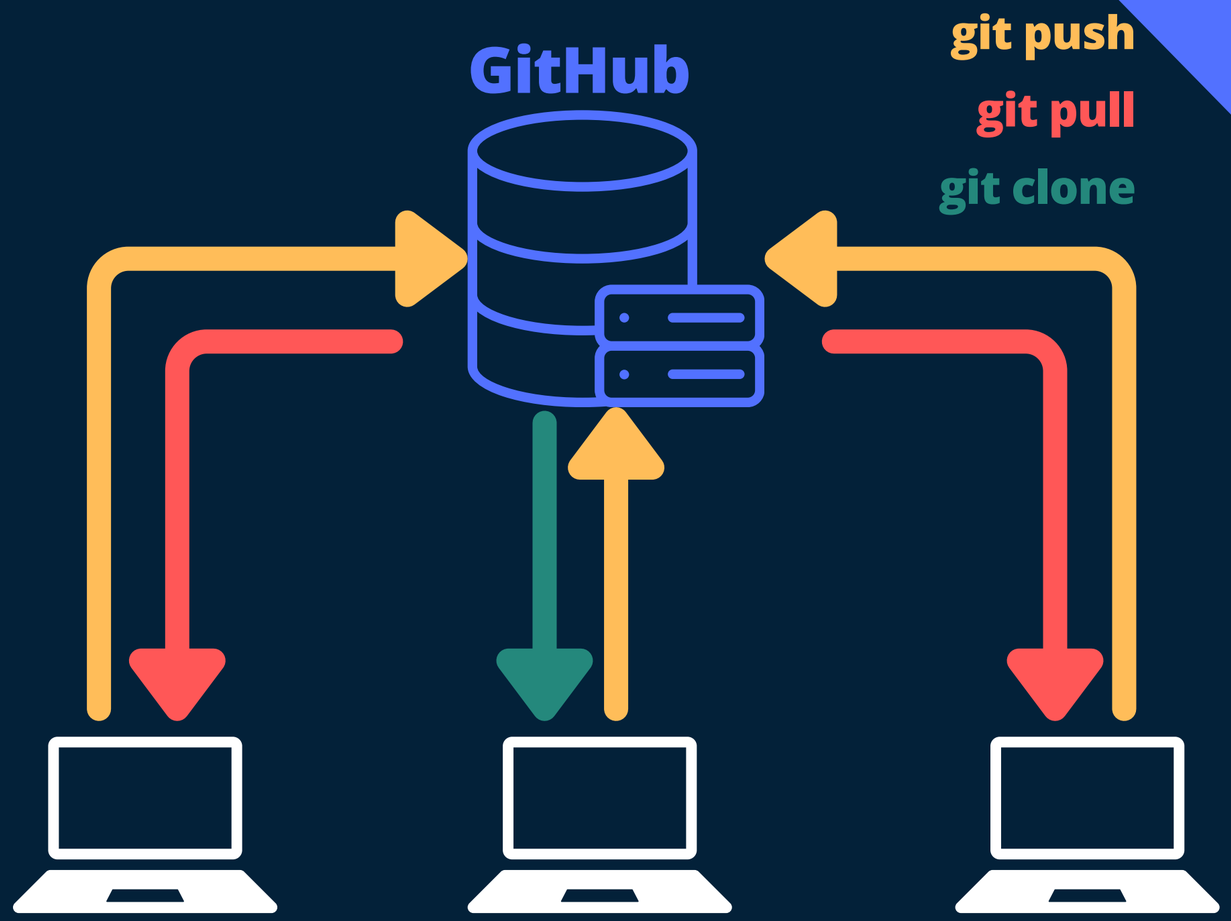
Come funziona?

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

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

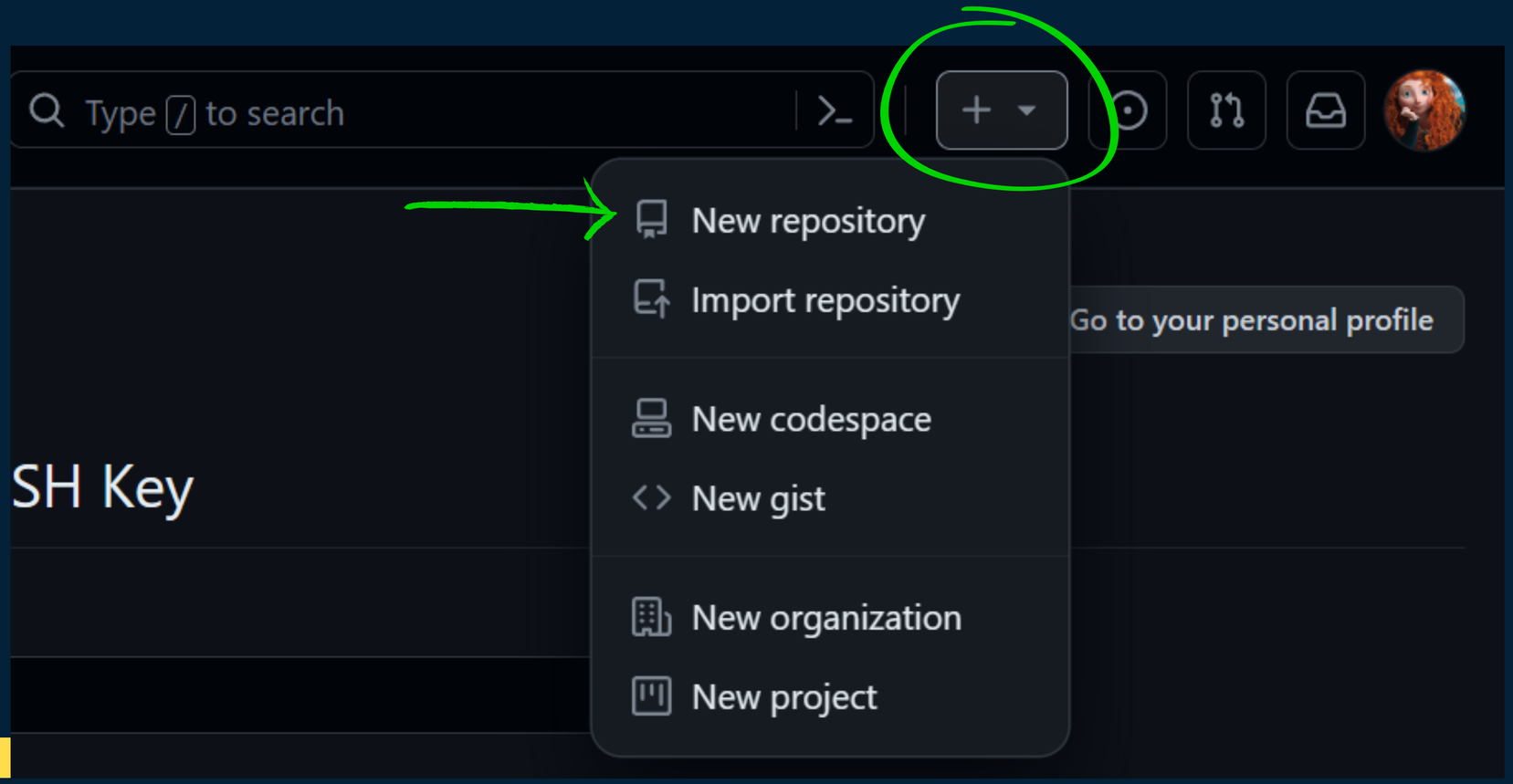
Una volta registrate le modifiche con *git commit*, è necessario **aggiornare** la repository remota.

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

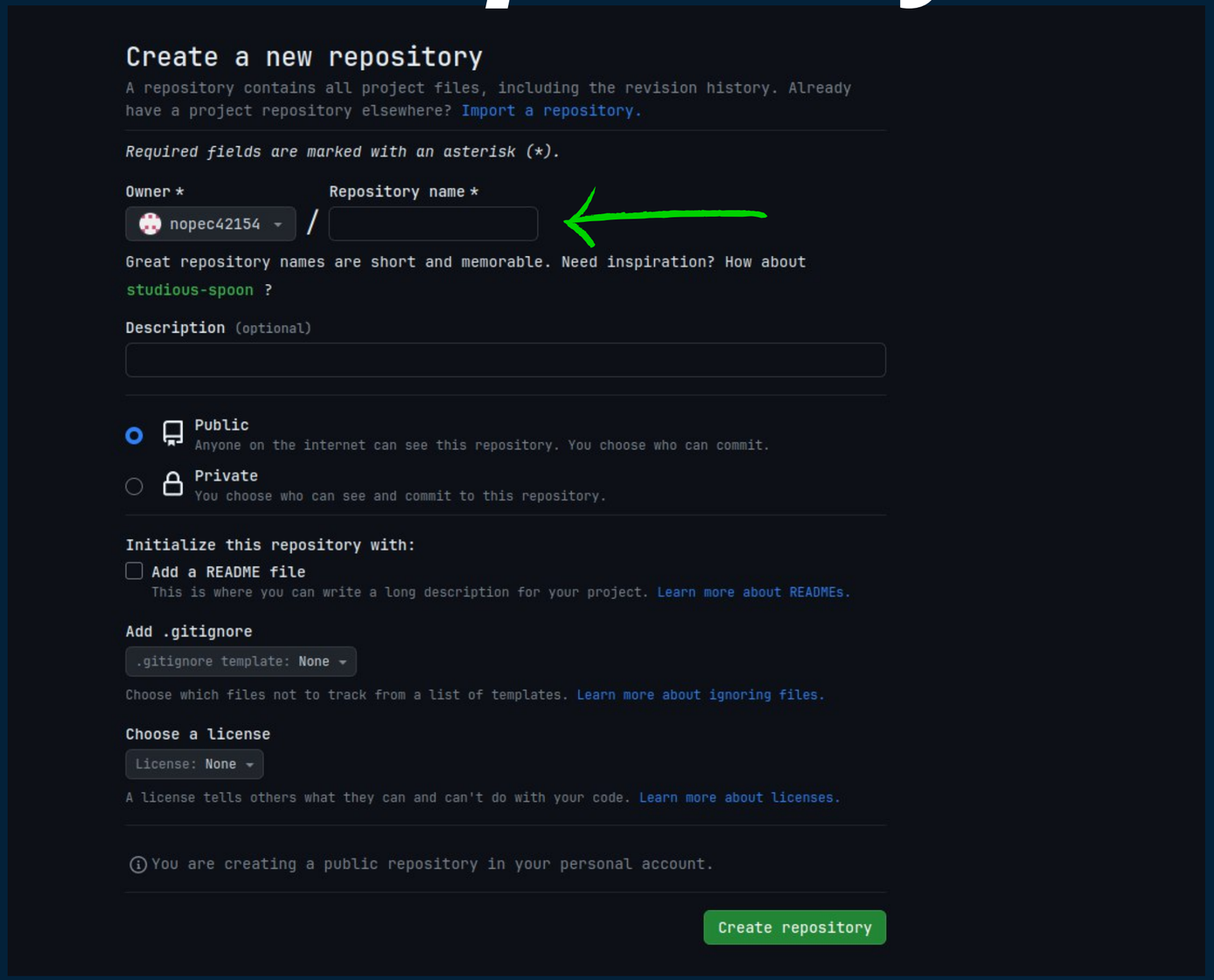


Creiamo una repository su GitHub

repository è un progetto mantenuto con git



Creiamo una repository su GitHub



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`

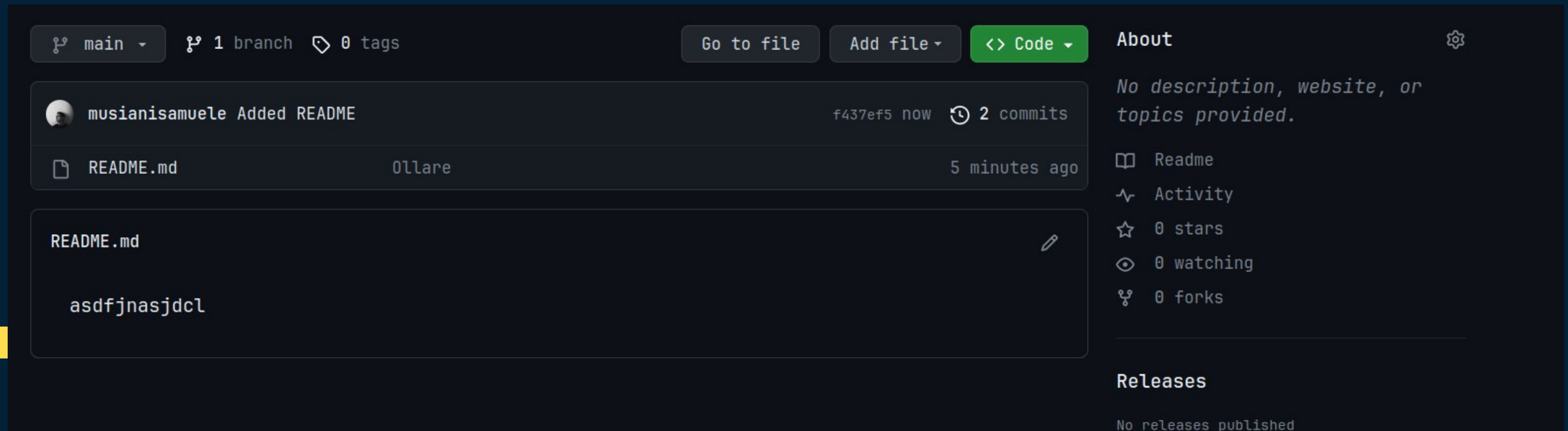
Creiamo un commit `$ git commit -m “add: README.md”`

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.

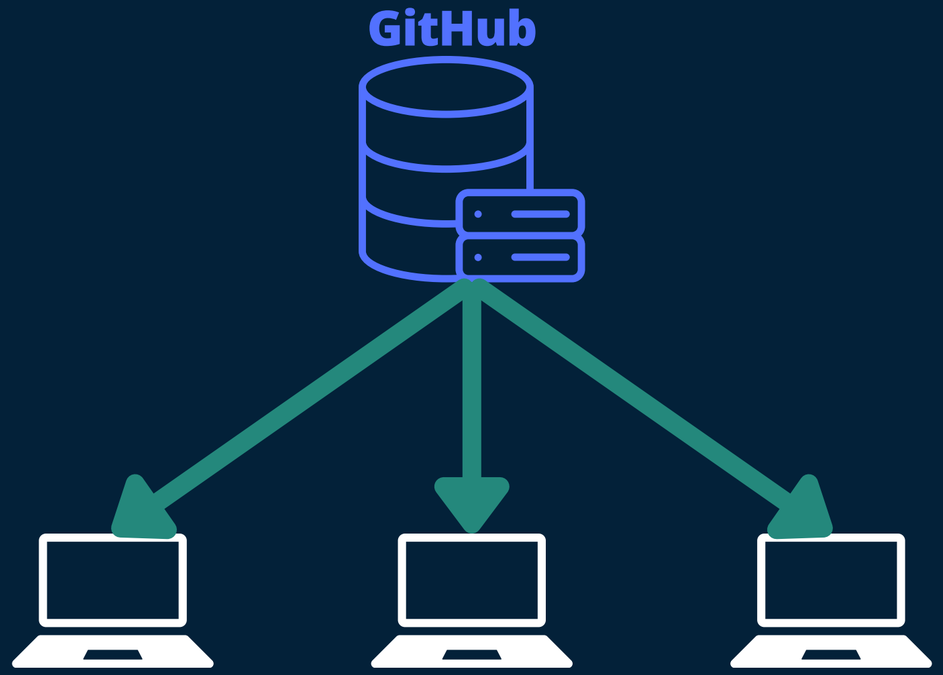
Bitbucket	GitHub	GitLab	So, what does it mean?
Pull Request	Pull Request	Merge Request	In GitLab a request to merge a feature branch into the official master is called a Merge Request.
Snippet	Gist	Snippet	Share snippets of code. Can be public, internal or private.
Repository	Repository	Project	In GitLab 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>
```

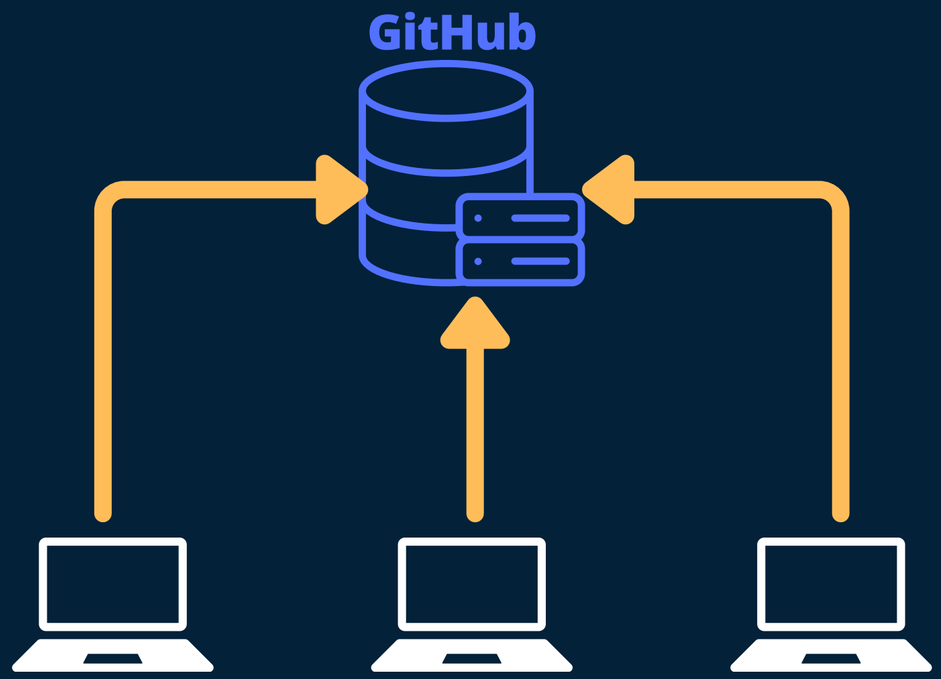


git push

02

Inviare i commit registrati alla repository remota

```
$ git push
```

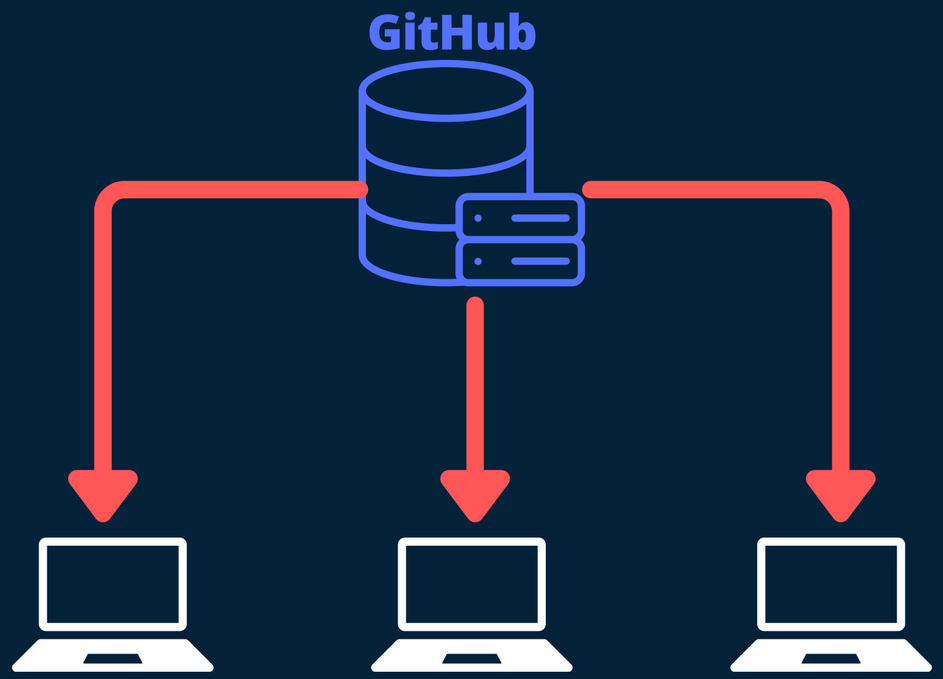


git pull

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
```

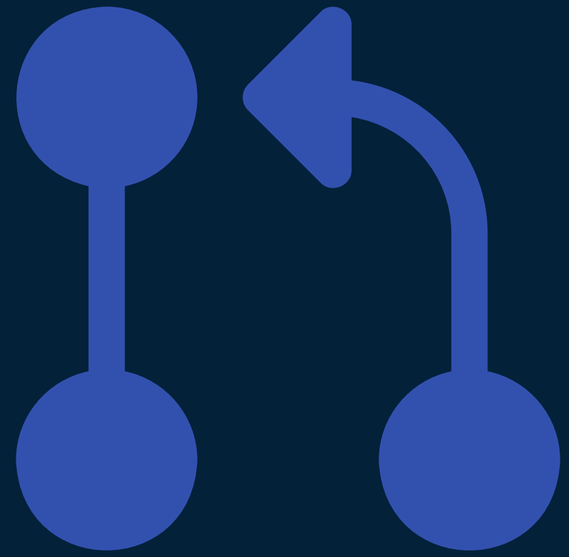
Creiamo la branch anche in remoto

```
$ 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:

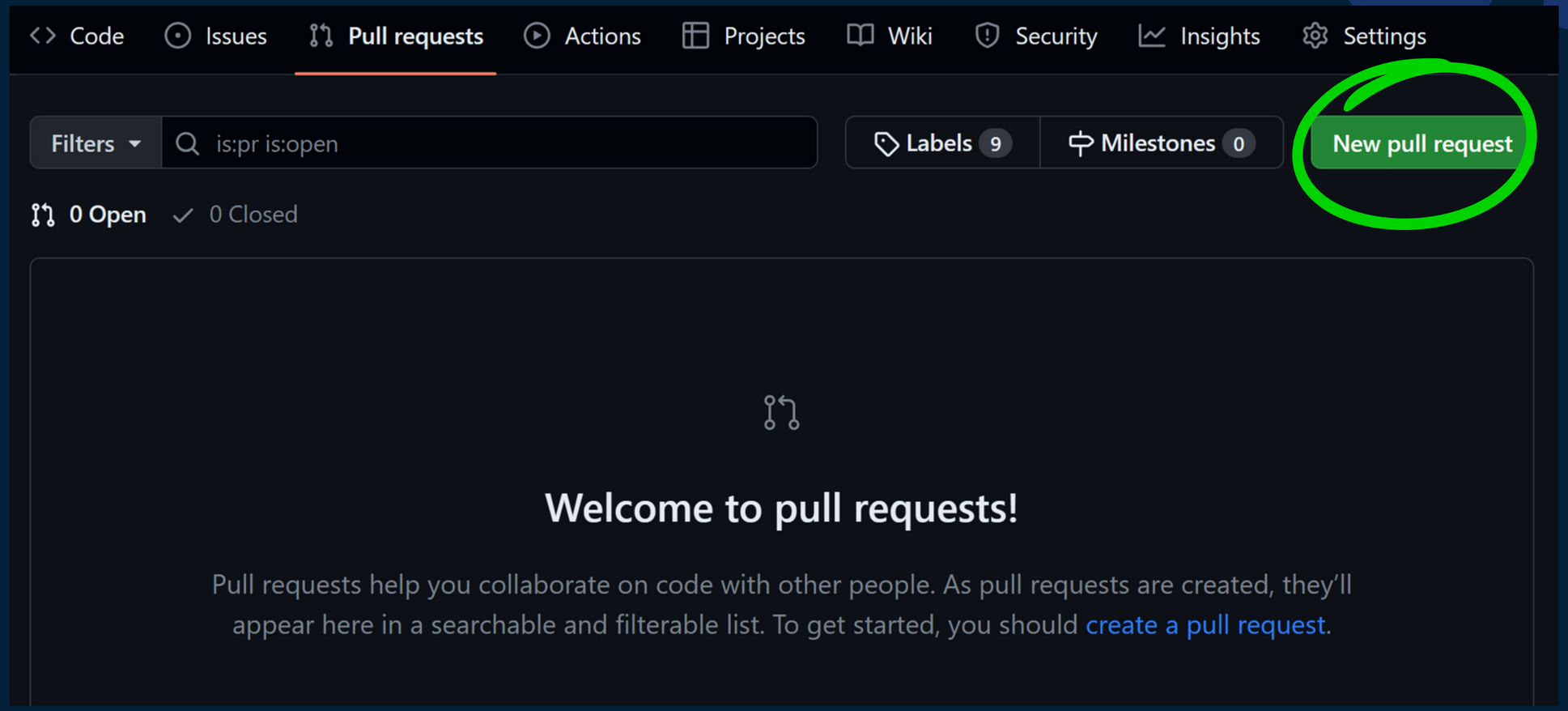
- discutere
- rivedere le potenziali modifiche con i collaboratori
- aggiungere commit di follow-up prima che le modifiche vengano unite nella branch main.

In pratica è come fare un merge in futuro:

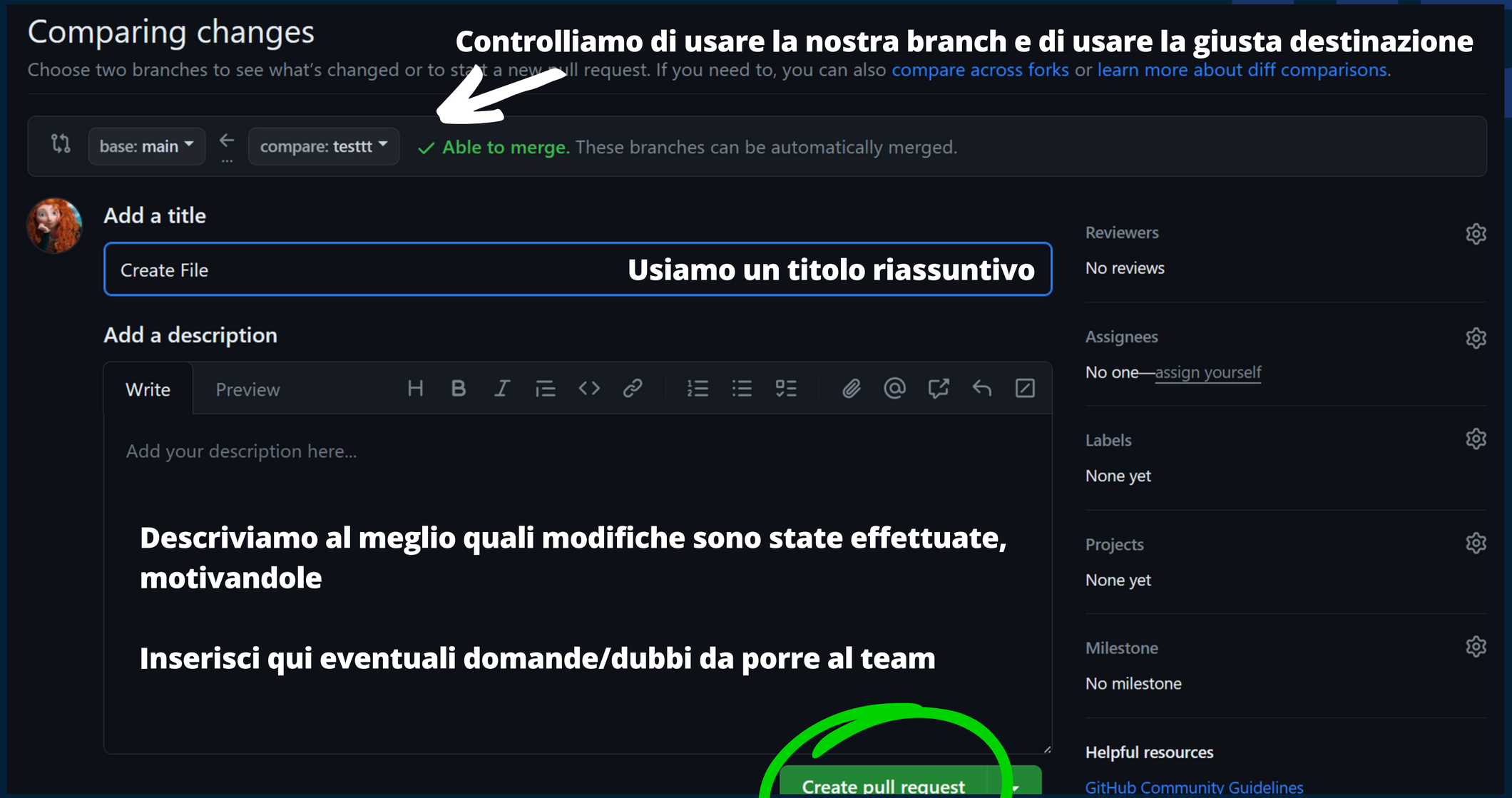
1. commentiamola con i nostri compagni
2. servono altre modifiche prima del merge?
3. mergiamo

05

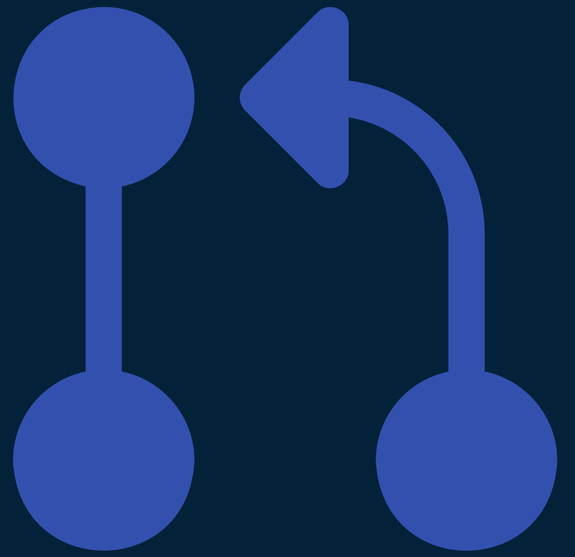
Andiamo su GitHub all’interno della nostra repository



Creiamo la Pull Request e aggiungiamo dettagli:



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

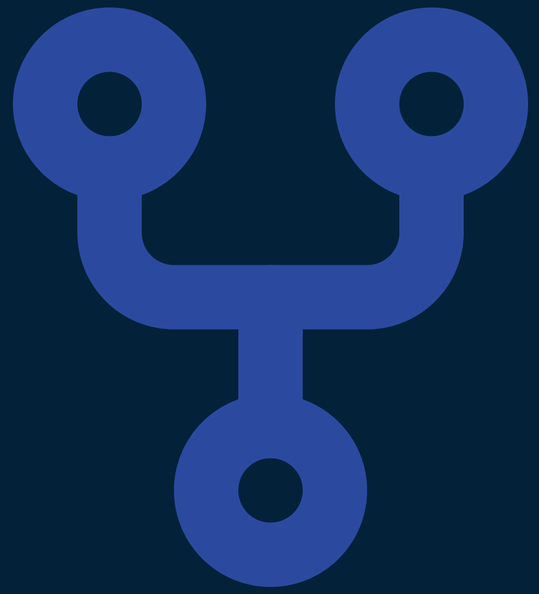
usiamo il meccanismo delle ***fork***

Fork

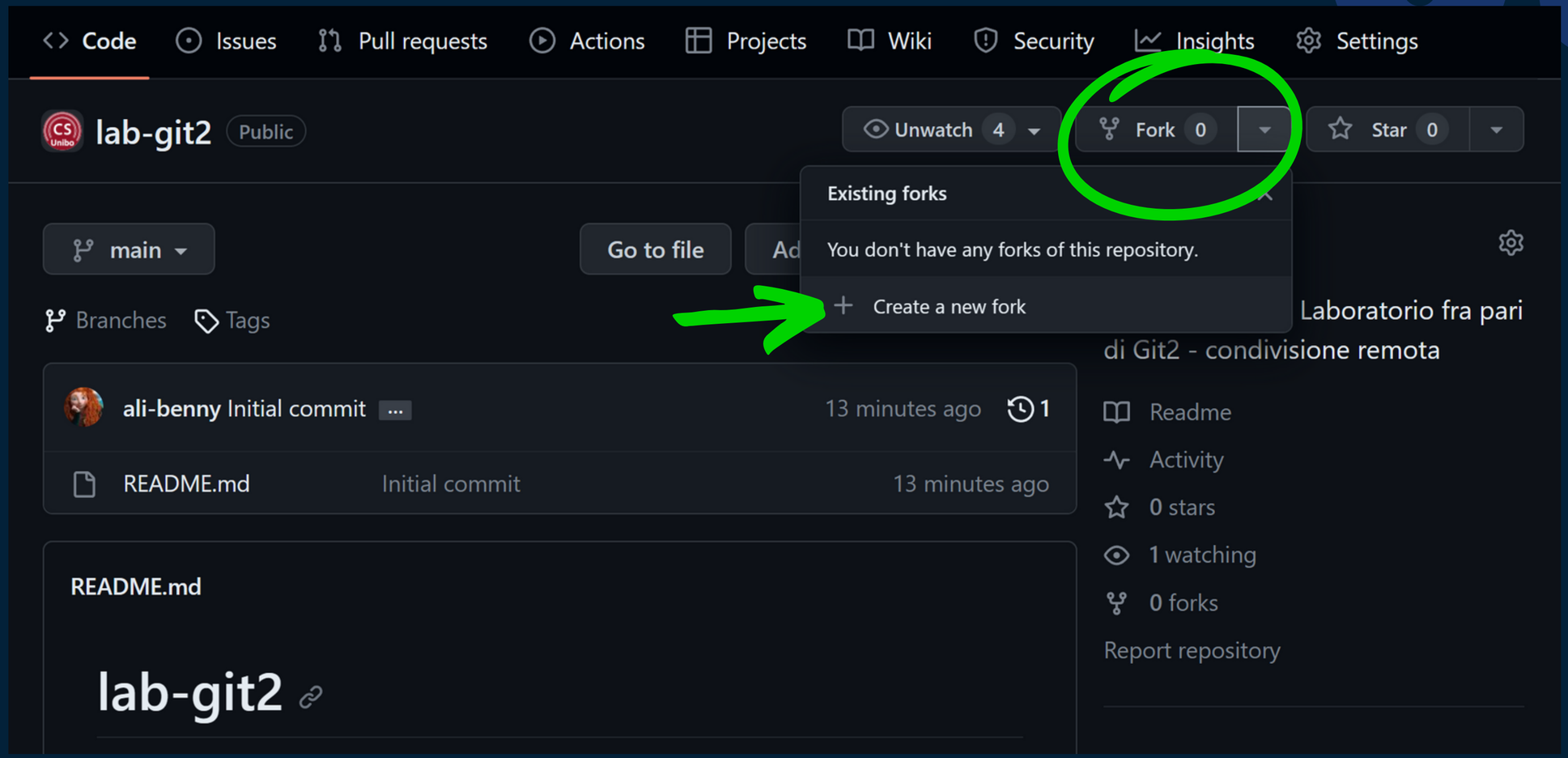
Una *fork* è un nuovo repository che condivide il codice e le impostazioni di visibilità con il repository originale.

Si usano per:

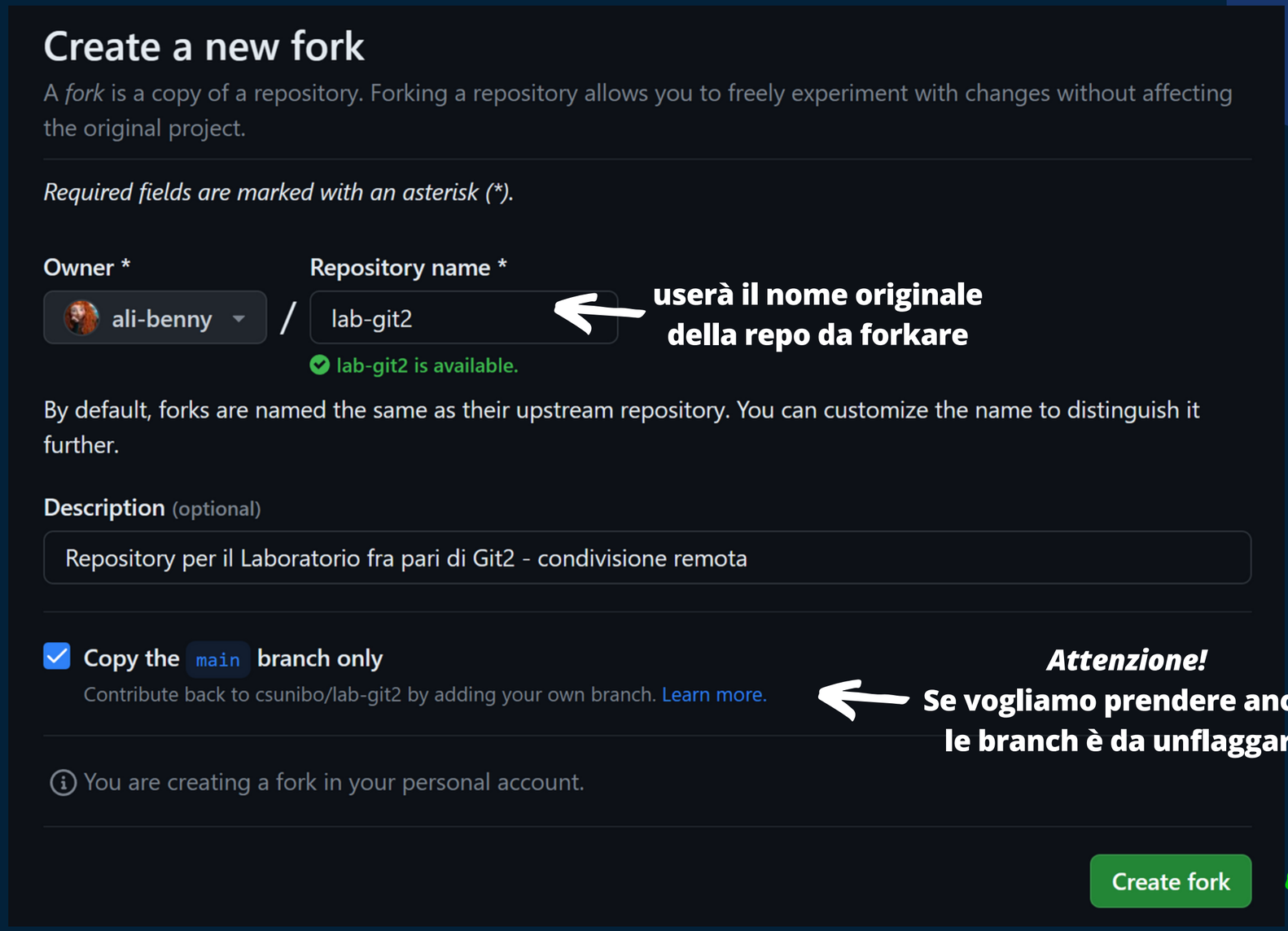
- proporre modifiche in una repository di qualcun’altro
  [su cui non si ha i permessi di scrittura]
- usare una repository come base per sviluppare un proprio progetto



Andiamo su GitHub nella repository da forkare



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



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

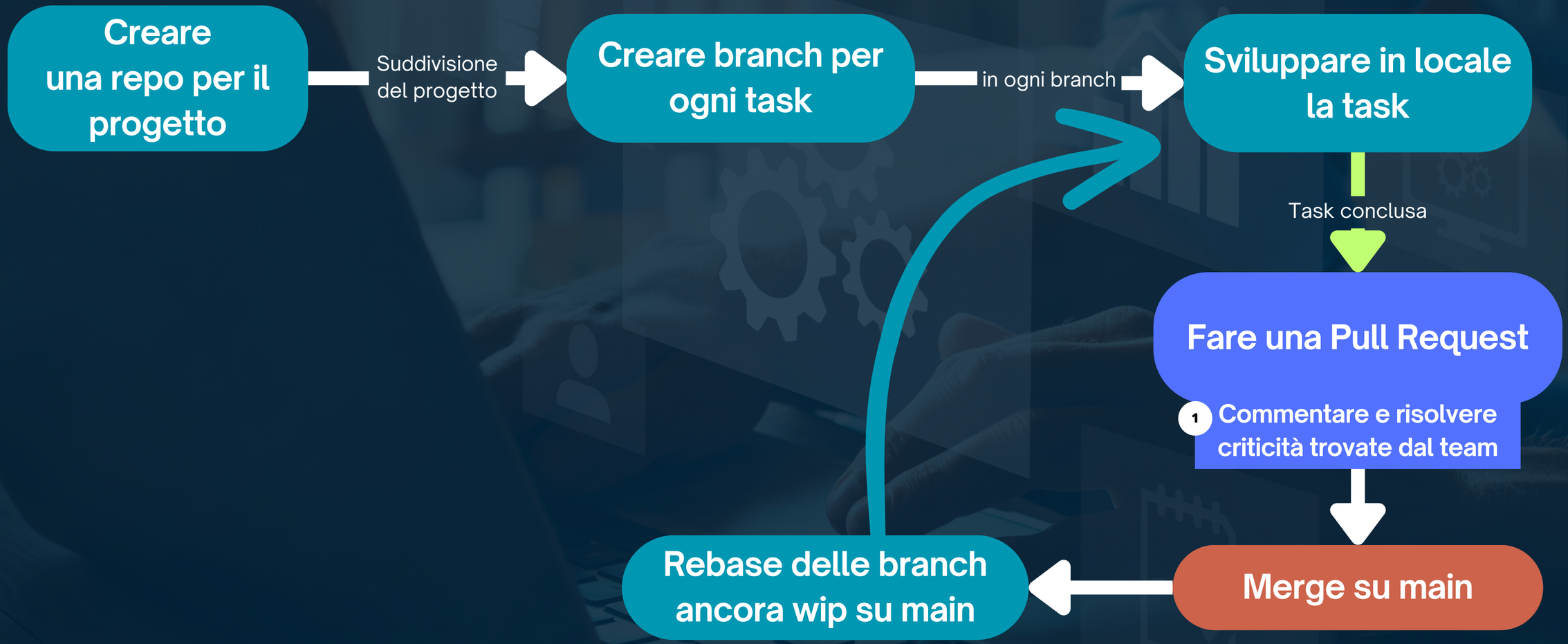
Aggiorniamo la repository nostra remota con le nuove modifiche:

```
$ 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 oprerazioni 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 vegono da quale versione:

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

Conflitti command line

A questo punto resta a noi scegliere quali righe tenere/modifichare/eliminare.
Il file dovrà risultare senza le linee aggiunte da git :

Conflitti presenti

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

Conflitti risolti

```
Modifiche “incoming”
```

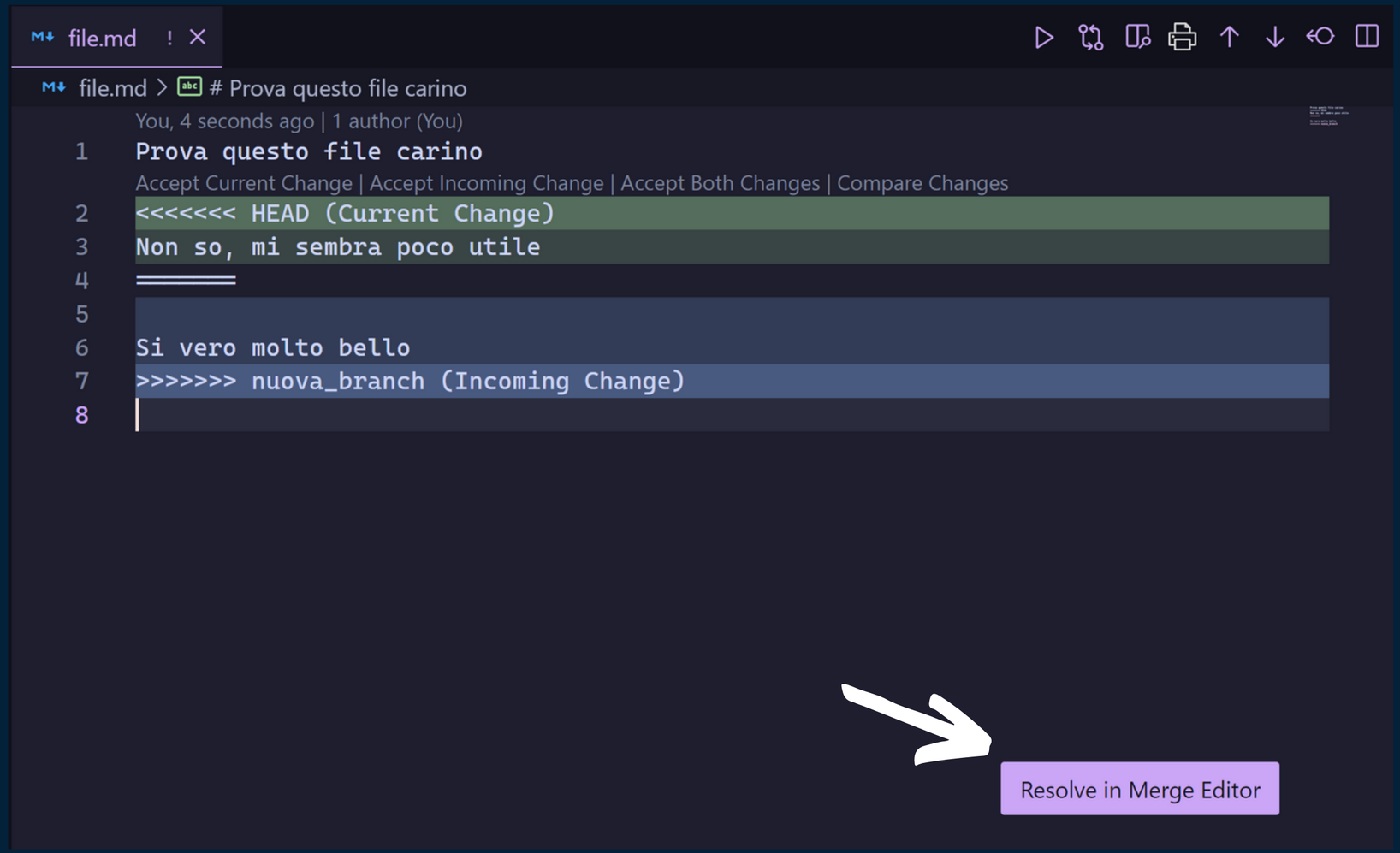
3. aggiungiamo alla staging area il file
4. in base all’azione che stavamo facendo eseguiamo:

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

`$ git rebase --continue`

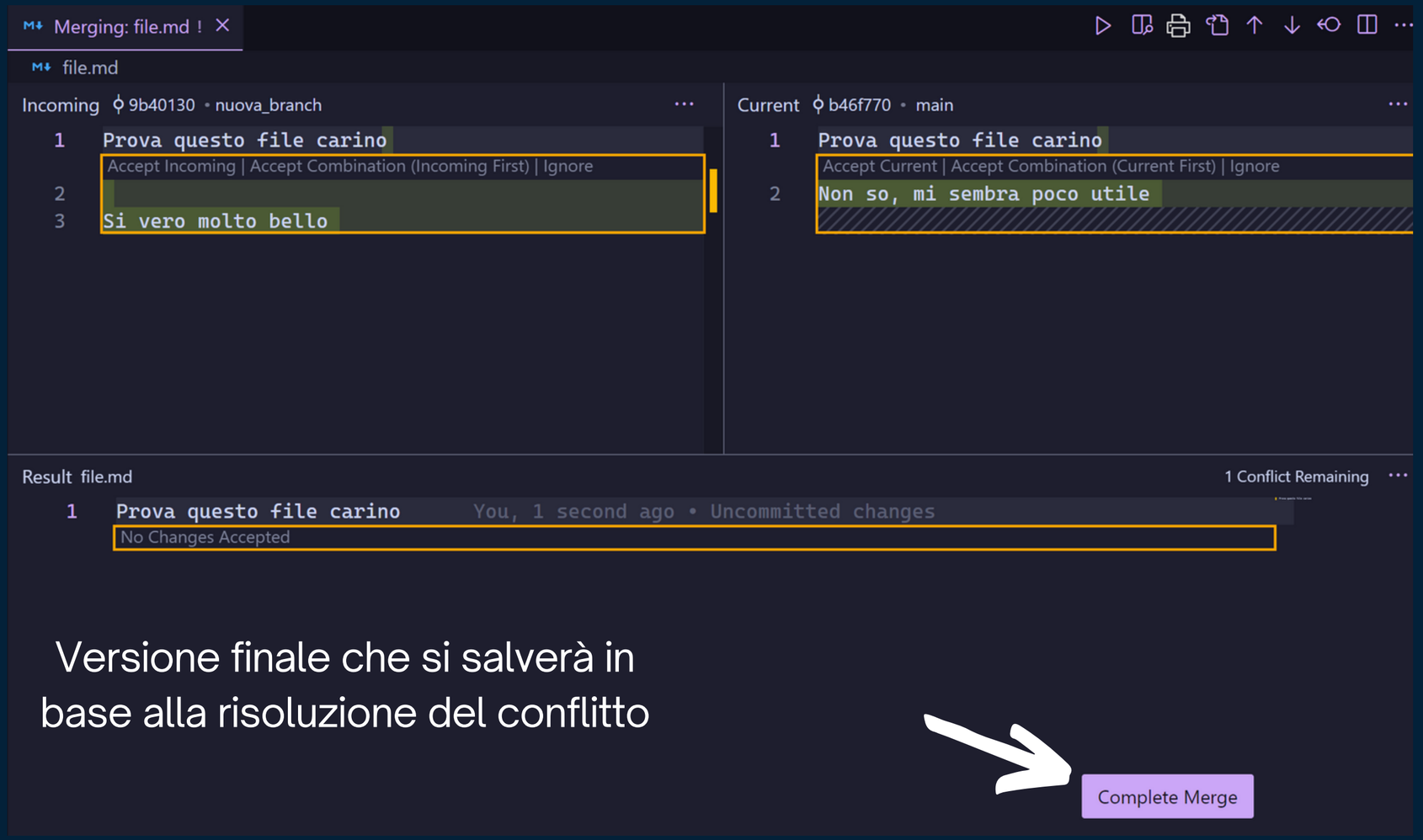
Conflitti vscode

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



Conflitti vscode

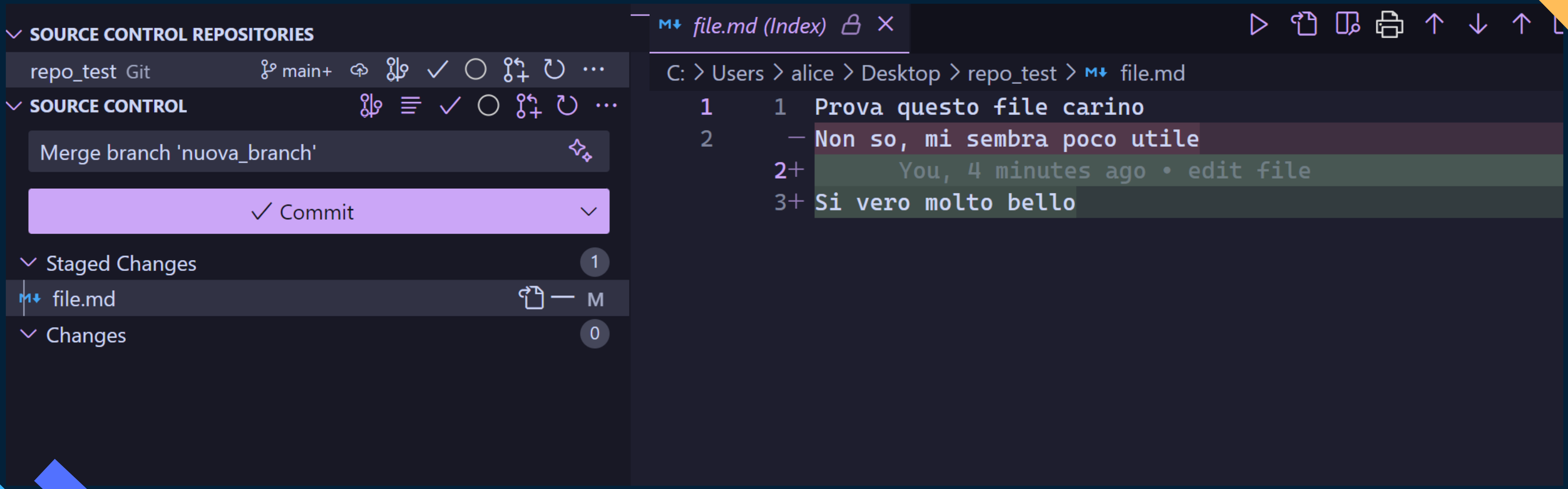
Nell’editor dei conflitti visualizziamo il file nelle due versioni presenti che generano il conflitto



Versione finale che si salverà in base alla risoluzione del conflitto

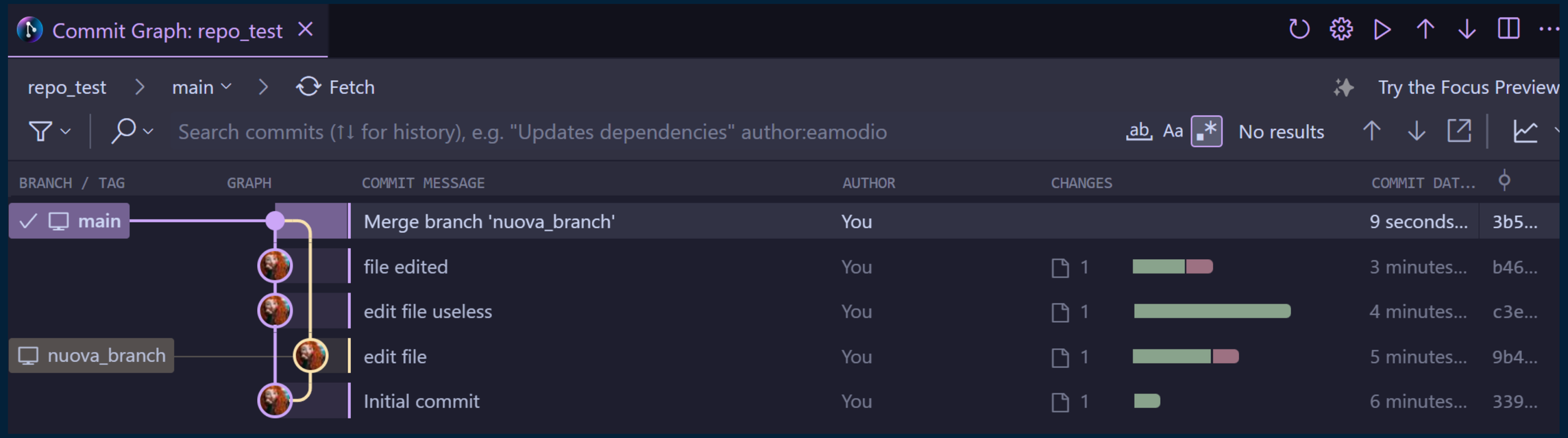
Conflitti vscode

Facciamo un commit aggiungendo le modifiche del merge



Usare Git vscode

Git log --graph --all su vscode



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 spostendo 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