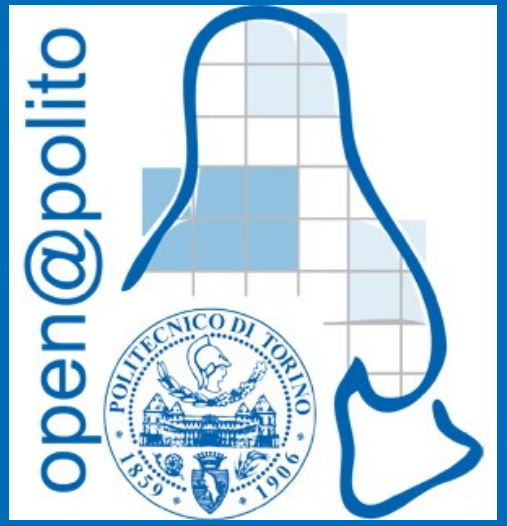
CONTROLLO VERSIONE CON GIT

Di Filippo Giacomin

Con il supporto di:

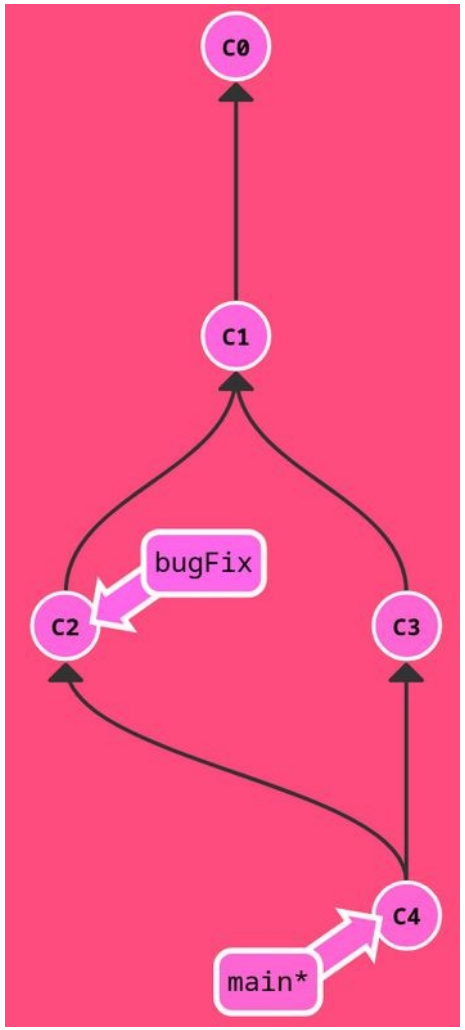


Cosa è Git?

Git è un software di controllo versione distribuito, open source e libero creato da Linus Torvalds nel 2005 per supportare lo sviluppo di Linux.

Elementi chiave

Con git è possibile registrare delle modifiche creando un **commit**. Ogni commit rappresenta uno stato del repository nel tempo. È possibile rappresentare un **repository** in maniera grafica con un grafo. Ogni commit può essere padre o figlio di più di un commit. Queste operazioni si chiamano rispettivamente **branch** e **merge**.



Caratteristiche di Git

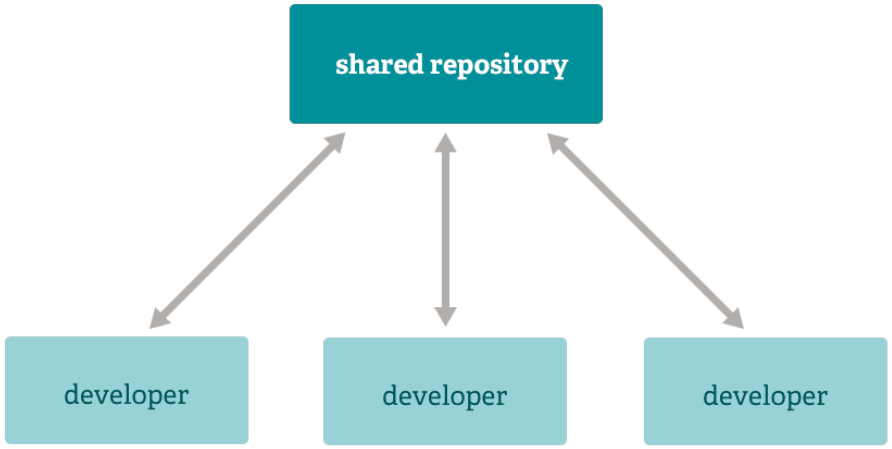
- Estremamente veloce
- Estremamente scalabile
- Altissima sicurezza dei dati
- Completamente distribuito
- Sviluppo non lineare facilitato
- Software aperto e libero

Alcuni punti di forza

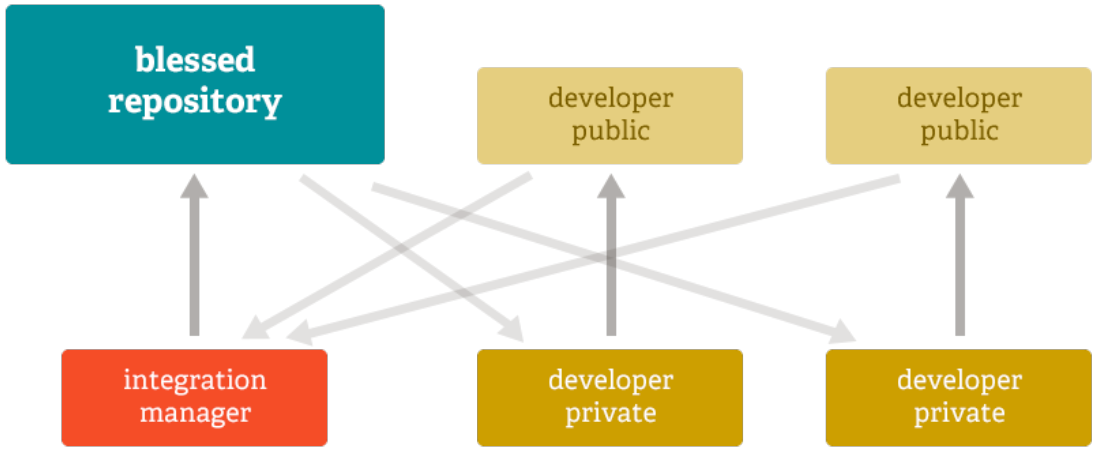
- Velocità:
  Quasi tutte le operazioni svolte da Git sono locali, rimuovendo la dipendeza dalla velocità del server e della connessione. Inoltre Git è stato sviluppato tenendo a mente l'efficienza fin dall'inizio, in quanto doveva supportare lo sviluppo di un grande progetto come il kernel Linux.
- Sicurezza:
  La quasi totalità dei comandi di git aggiungono dati al repository (anche il comando `git rm`!) senza rimuovere mai informazioni, così da poter tornare indietro in caso di necessità. Inoltre ogni commit possiede un hash generato da se stesso e dai commit precedenti, rendendo evidente quando un vecchio commit è stato alterato.

Cosa significa “distribuito”?

Con Git ogni utente scarica l’intero repository sul proprio dispostivo, dove può applicare qualsiasi cambiamento senza influenzare il lavoro degli altri utenti. Con un controllo versione è possibile implementare diversi tipi di workflow:



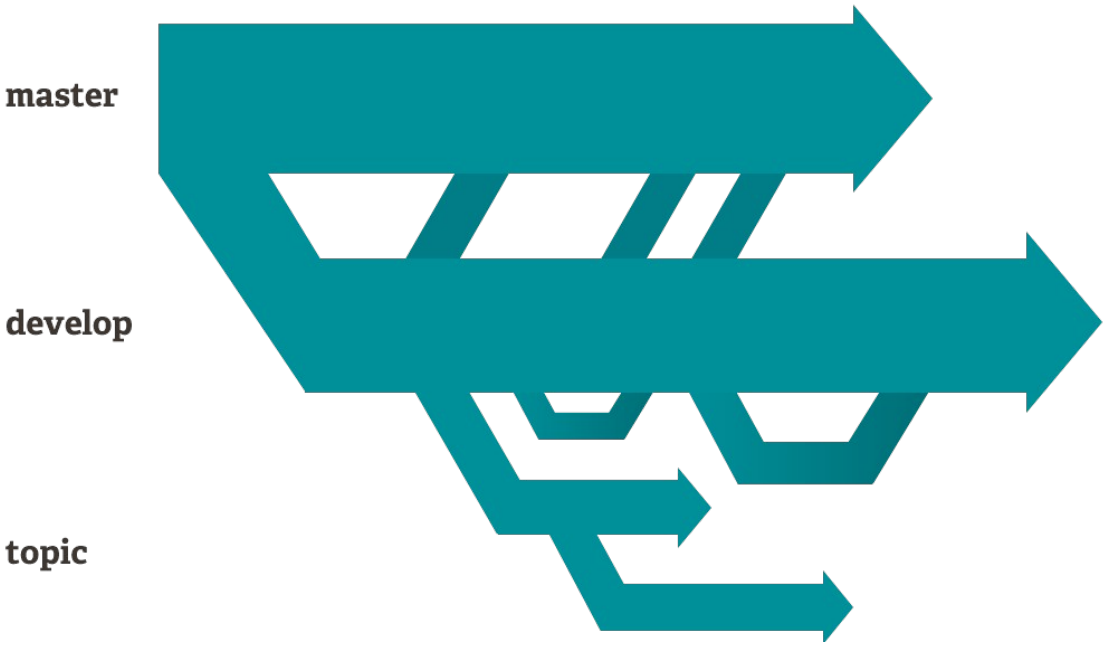
Centralized workflow



Integration manager workflow

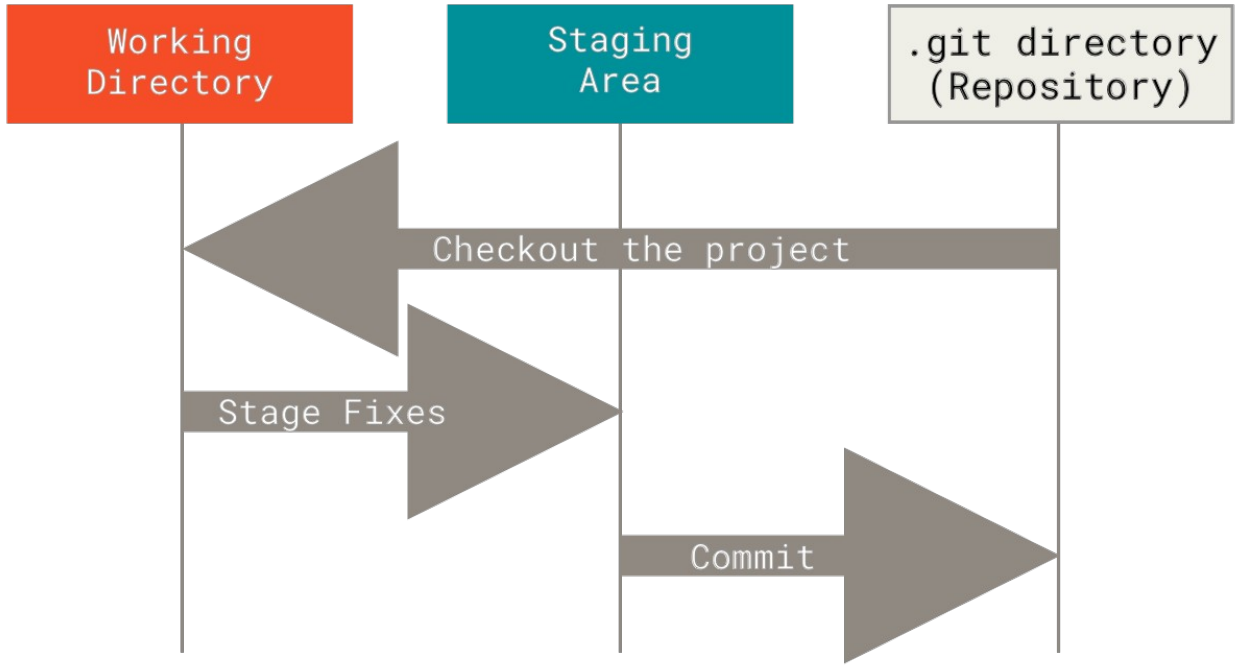
Braching e merging

Il branching in Git è ***facile e veloce***. Avendo diversi branch all'interno dello stesso repository è possibile mantenere diverse versioni del codice, ad esempio una principale, una di sviluppo e diverse per le nuove feature.



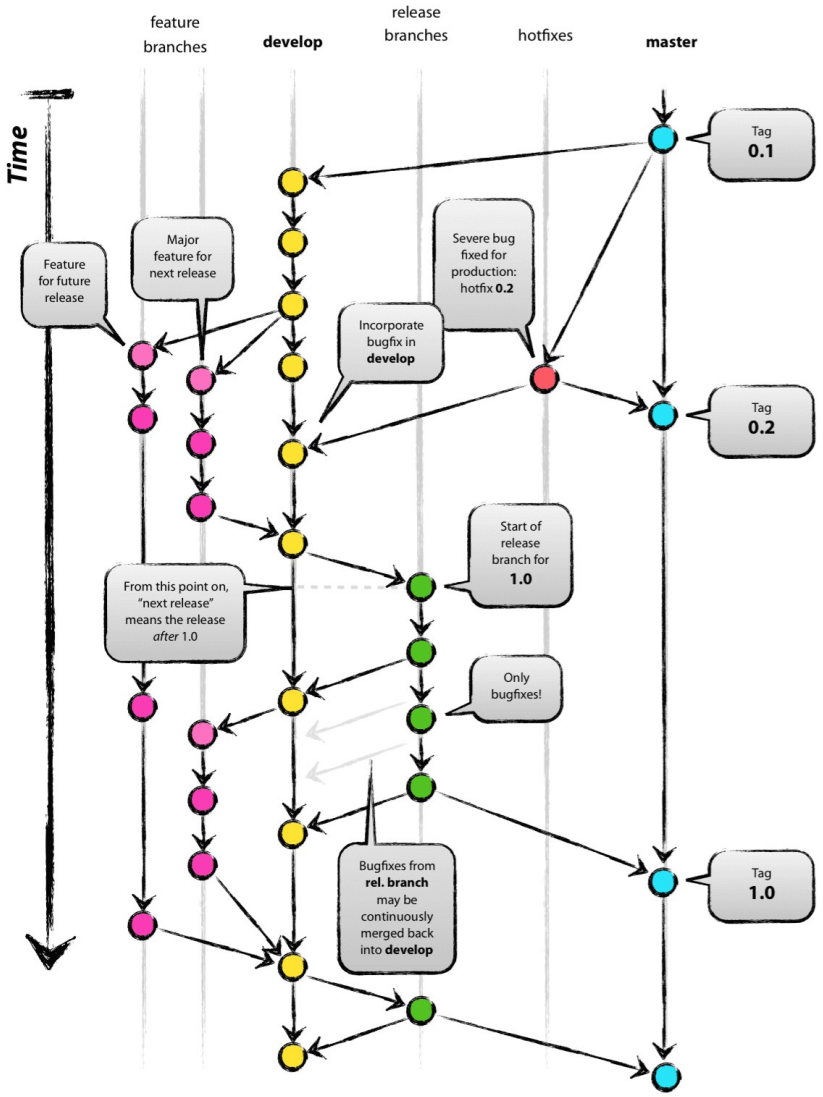
L’area di lavoro

- Working directory:
  La directory tracciata da Git
- Staging area:
  Le modifiche dall’ultimo commit tracciate da Git
- Repository:
  Dove Git memorizza tutta la cronologia del progetto



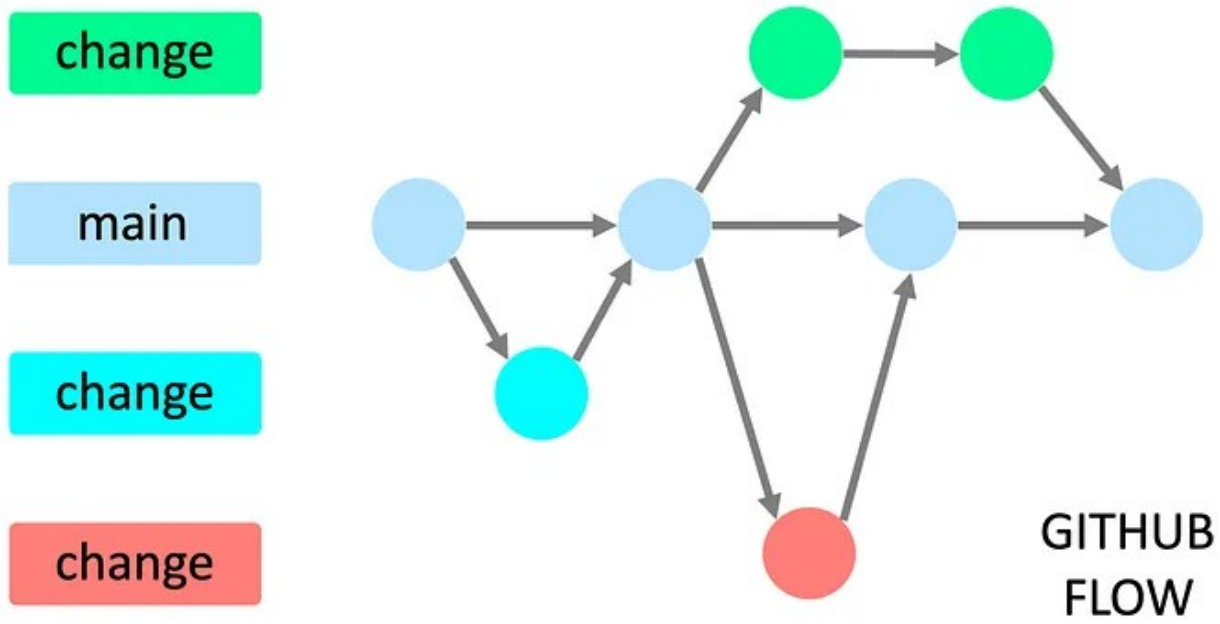
Diversi workflow: Git flow

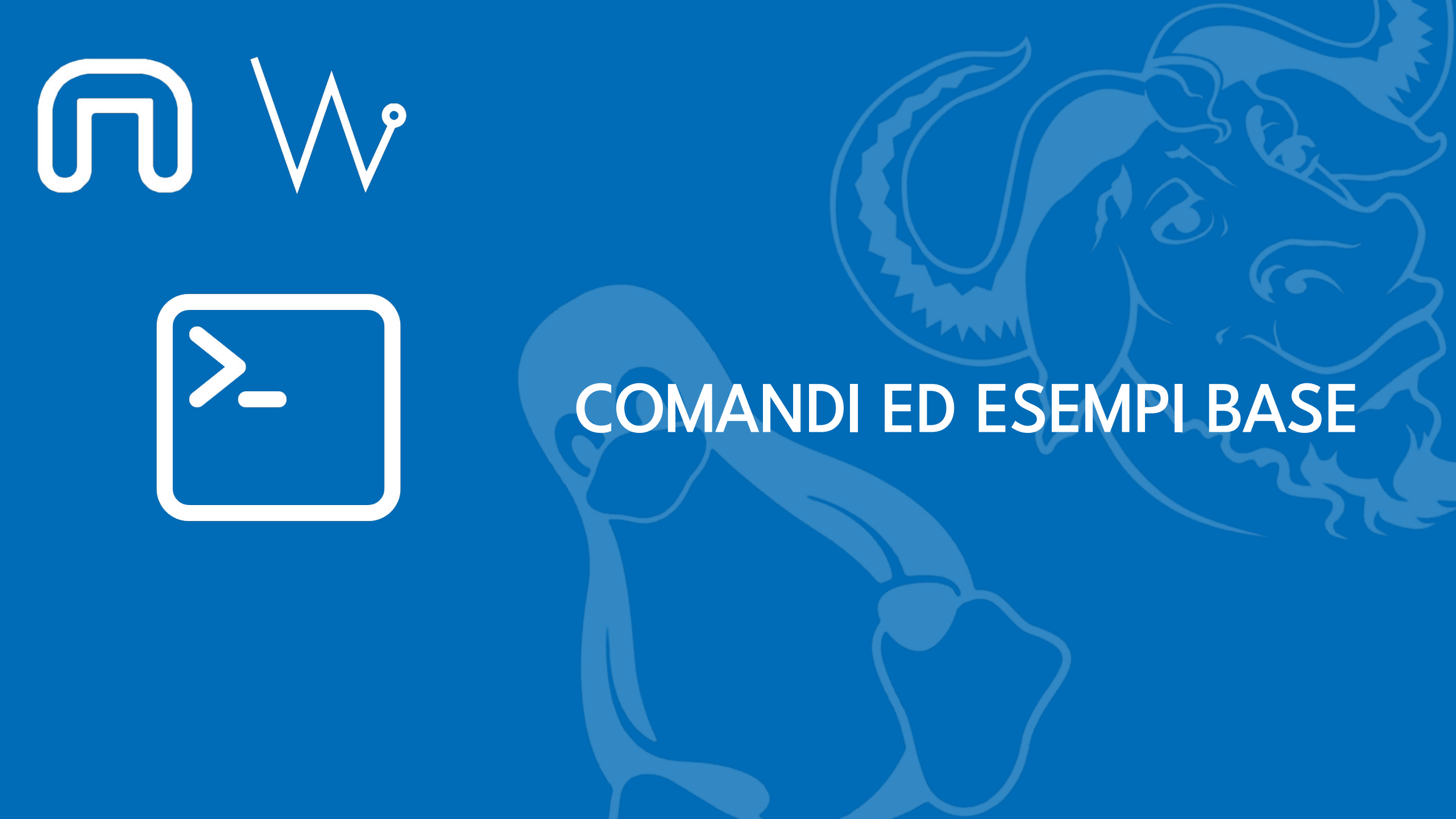
- Numerosi brach:
  - main: produzione
  - Develop: sviluppo generico
  - Feature: sviluppo per feature specifica
  - Hotfix: patch urgenti per main
  - Release: preparazione alla release
- Utile per team medio/grandi e rilasci strutturati
- Complesso



Diversi workflow: GitHub flow

- Un singolo branch duraturo
- Utile per team più piccoli
- Il merge in main richiede una pull request
- Feedback immediato
- Rilascio di nuove feature immediato





COMANDI ED ESEMPI BASE

Configurazione iniziale

Prima di fare un commit bisogna fornire un nome utente e una mail con cui firmare il commit:

```
git config user.name <nome utente>
git config user.email <email>
```

Per cambiare le impostazioni solo per l'intero sistema basta aggiungere:

```
--global
```

Come iniziare

Creare un repository locale è facilissimo:

```
git init
```

Clonare un repository remoto è altrettanto facile:

```
git clone <url>
```

Come iniziare

Per visualizzare lo stato dell'area di lavoro si usa:

```
git status
```

Come iniziare

Per aggiungere dei file alla staging area si usa il comando:

```
git add <file1> <file2> <etc>
```

È necessario aggiungere sia i file non tracciati che i file che sono stati modificati dall'ultimo commit, altrimenti le modifiche non saranno registrate.

Per aggiungere tutti i file nella directory si può usare:

```
git add .
```

Come iniziare

Per spostare le modifiche dalla staging area al repository si usa:

```
git commit -m <messaggio>
```

È importante usare un messaggio breve, ma che spieghi chiaramente le modifiche effettuare al codice.

Per visualizzare la cronologia dei commit si usa:

```
git log
```

Usare i tag

Si può dare un nome ad un commit speciale creando un tag. Esistono tag annotated (salva autore, data e un messaggio) e lightweight (non salva nessun dato aggiuntivo). Per taggare l'ultimo commit si usa:

Annotated:

```
git tag -a <nome> -m <messaggio>
```

Lightweight:

```
git tag <nome>
```

Per taggare un commit più vecchio basta aggiungere i primi caratteri dell'hash del commit desiderato alla fine del comando.

Usare i tag

Per vedere tutti i tag creati è sufficiente usare:

```
git tag
```

Per visualizzare le informazioni di un tag specifico si usa:

```
git show <nome tag>
```

Generalmente i tag sono utilizzati per marcare un commit che rappresenta una specifica versione del programma, ad esempio v1.4.5d

Repository remoti

Se il repo è stato creato con `git clone`, i remote sono già configurati, altrimenti possono essere aggiunti o rimossi con:

```
git remote add <nome> <url>
git remote rm <nome>
```

Repository remoti

Per collaborare con altri utenti bisogna sincronizzare le modifiche con i vari remote:

Scaricare le nuove informazioni da un remote:

```
git fetch <nome>
```

Scaricare le nuove modifiche da un remote:

```
git pull <nome>
```

Caricare le proprie modifiche locali su un remote:

```
git push <nome>
```

Repository remoti

⚠️ ATTENZIONE ⚠️

Il comando push non carica i tag creati localmente. Per caricarli si può usare:

Tag specifico:

```
git push <nome remote> <nome tag>
```

Tutti i tag:

```
git push <nome remote> --tags
```

Stash

Se c'è la necessità di salvare le modifiche nell'area di lavoro e e riportarla allo stato dell'ultimo commit si può usare il comando:

```
git stash
```

Per richiamare le modifiche si usa:

```
git stash apply <stash>
```

Per visualizzare tutti gli stash:

```
git stash list
```

Per eliminare uno stash:

```
git stash drop <stash>
```

Branch

Si può creare un branch col comando:

```
git branch <nome branch>
```

Tuttavia il branch non sarà automaticamente selezionato. Per selezionarlo si usa:

```
git checkout <nome branch>
```

A questo punto ogni commit sarà creato sul branch selezionato.

Per creare e selezionare un branch con un singolo comando si può utilizzare direttamente:

```
git checkout -b <nome branch>
```

Merge e rebase

Si possono unire due commit in un terzo commit creando un merge:

```
git merge <branch>
```

Le modifiche verranno unite automaticamente dove possibile, altrimenti l'utente dovrà revisionarle manualmente.

Un approccio simile è il rebase, dove però i commit di un branch verranno copiati sull'altro branch:

```
git rebase <branch>
```

⚠️ `git rebase` "abbandona" i vecchi commit, quindi va usato solo localmente! ⚠️

Merge e rebase

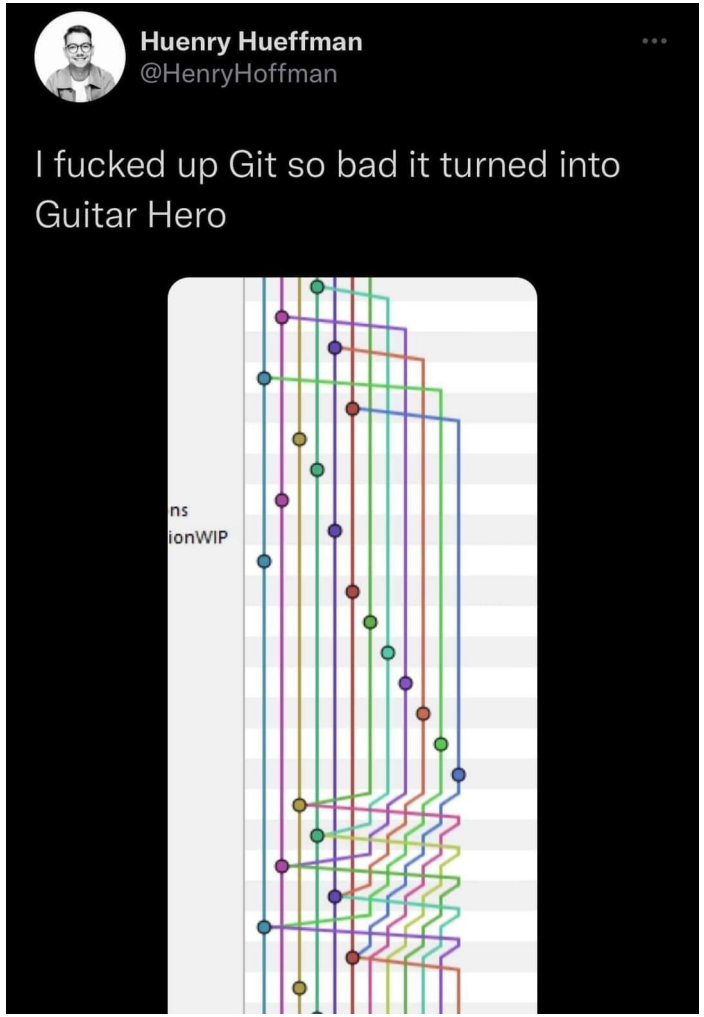
Merge e rebase portano apparentemente allo stesso risultato, tuttavia sono adatti ad ambienti di versi.

Usa merge se:

- Vuoi documentare lo sviluppo parallelo
- Stai lavorando in team

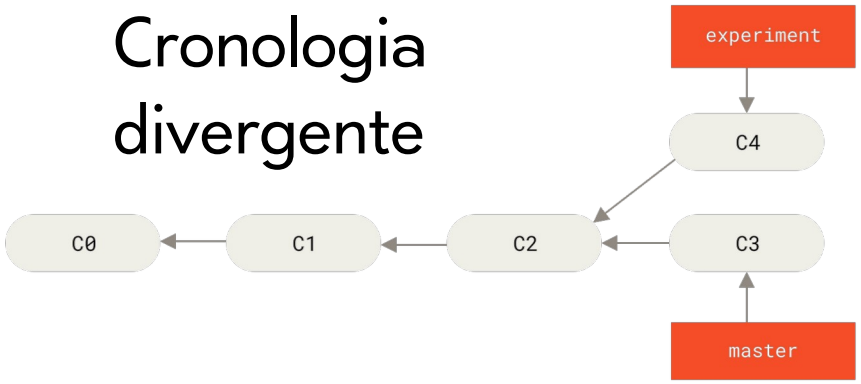
Usa git rebase se:

- Preferisci mostrare una cronologia lineare
- Stai lavorando da solo

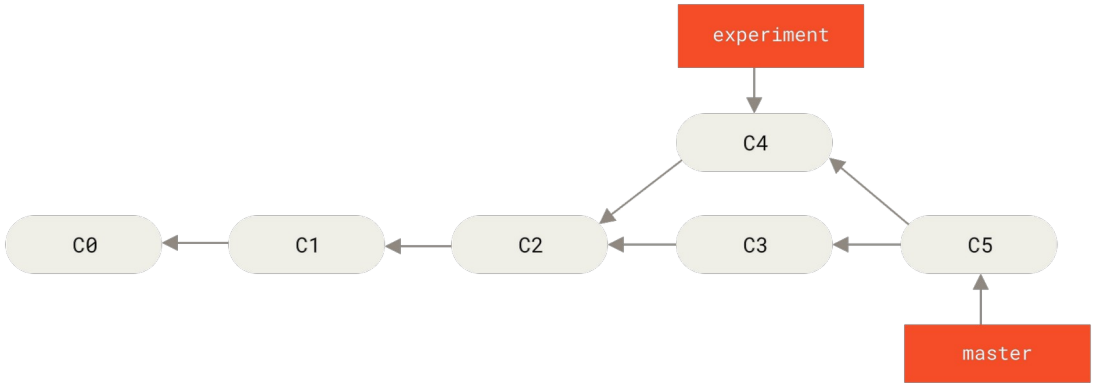


Merge e rebase

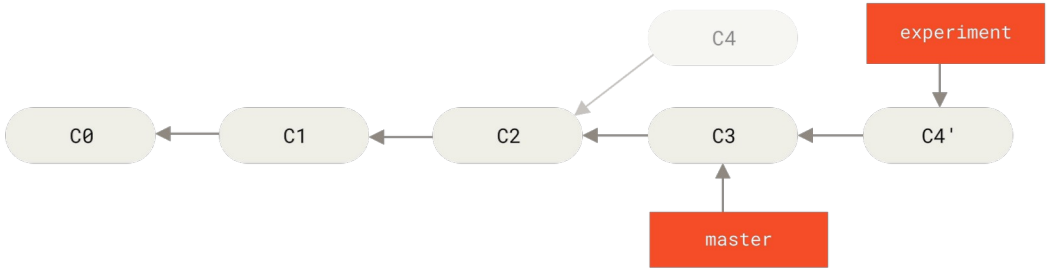
Cronologia divergente



Merge



Rebase



Merge conflict

Si parla di merge conflict quando Git non riesce ad unire automaticamente le modifiche durante un merge. In questi casi l'utente deve controllare e unire le modifiche manualmente.

Per risolvere facilmente un merge conflict si può usare il comando:

```
git mergetool
```

Git aprirà le modifiche in uno dei difftool installati nel sistema, col quale l'utente potrà facilmente gestire le modifiche.

ANNULLARE LE MODIFICHE

Modificare l'ultimo commit

Se è stato commesso un'errore o è stato dimenticato di aggiornare un file nell'ultimo commit, è possibile modificarlo. Per modificare un commit basta inserire le modifiche desiderate nella staging area e poi usare:

```
git commit --amend -m 'messaggio'
```

Per non modificare il messaggio basta sostituirlo con `--no-edit`.

⚠️ NON MODIFICARE UN COMMIT GIÀ PUSHATO ⚠️

Annullare un commit

È possibile annullare le modifiche di un commit in due maniere:

Col comando `git reset` è possibile spostare il branch indietro al commit desiderato. Questa operazione riscrive la cronologia e non va utilizzata su repository condivisi:

```
git reset <commit>
```

Col comando `git revert` è possibile creare un commit che inverte le modifiche effettuate fino al commit scelto:

```
git revert <commit>
```

⚠️ USA SOLO REVERT SU REPO CONDIVISI ⚠️

Staccare HEAD

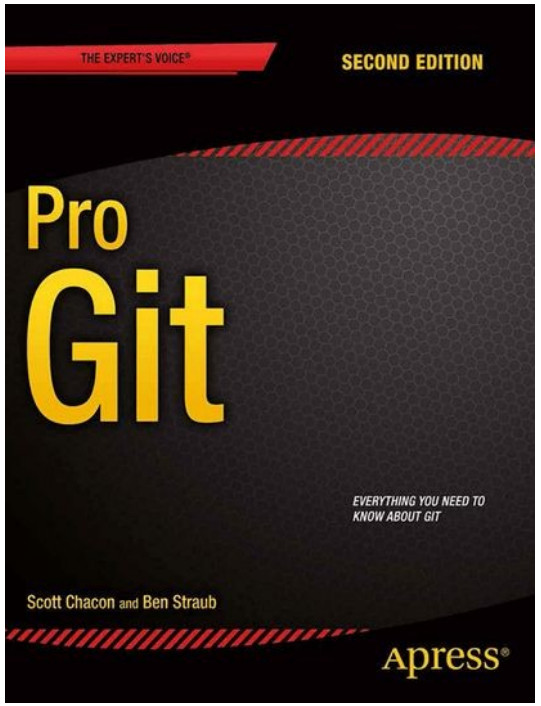
È possibile usare git checkout per selezionare un commit in particolare:

```
git checkout <commit>
```

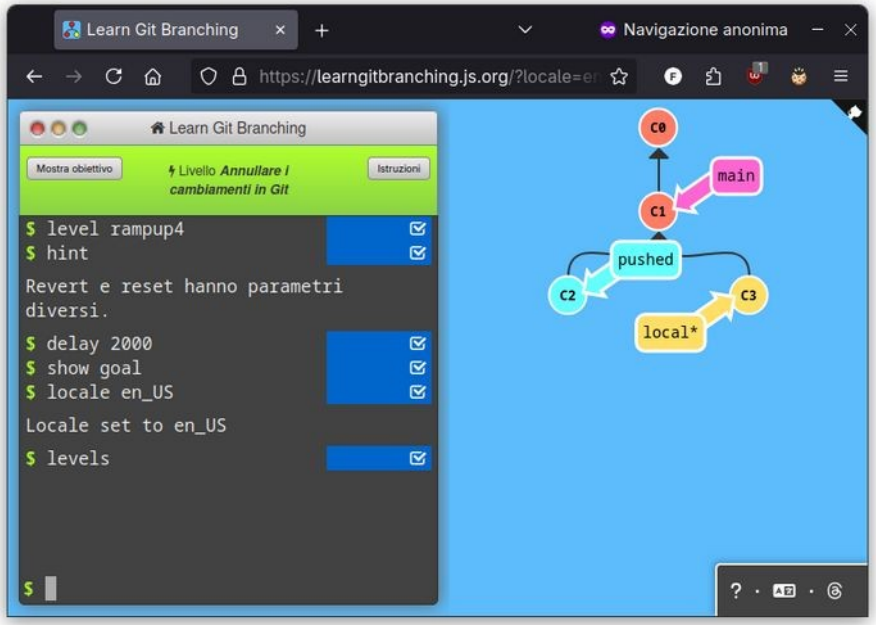
A questo punto è possibile creare un nuovo branch che parta da questo commit.

DOMANDE?

E ora?



Pro Git book
https://git-scm.com/book/en/v2



Learn Git Branching
https://learngitbranching.js.org/

Fonti

Immagini:

- https://www.ted.com/talks/linus_torvalds_the_mind_behind_linux
- https://commons.wikimedia.org/wiki/File:Git-logo.svg#/media/File:Git-logo.svg
- https://git-scm.com/about/distributed
- https://git-scm.com/about/branching-and-merging
- https://nvie.com/posts/a-successful-git-branching-model/
- https://medium.com/@yanminthwin/understanding-github-flow-and-git-flow-957bc6e12220
- https://www.flaticon.com/free-icons/terminal
- https://github.com/hoduche/git-graph
- https://git-scm.com/book
- https://www.flaticon.com/free-icons/revert
- https://x.com/HenryHoffman/status/694184106440200192

Informazioni:

- https://it.wikipedia.org/wiki/Git_(software)
- https://git-scm.com/docs
- https://git-scm.com/book/en/v2
- https://git-scm.com/about
- https://nvie.com/posts/a-successful-git-branching-model/
- https://githubflow.github.io/
- https://learngitbranching.js.org/