

INFORMATICA

Intelligenza Artificiale & Data Analytics

Precorsi a.a. 2024/25

Docente: Gloria Pietropolli

5. VERSION CONTROL SYSTEMS

Richiami di markdown e la necessità del
versioning

RECAP: MARKDOWN

Markdown introduce una formattazione basilare all'interno dei file di testo

- Possiamo rendere il testo grassetto, italico e sottolineato
- Possiamo indicare titoli e sottotitoli
- Possiamo indicare formule matematiche

Funzione

Una funzione è una relazione tra due insiemi (<https://it.wikipedia.org/wiki/Insieme>), chiamati dominio e codominio della funzione, che associa a ogni elemento del dominio uno e un solo elemento del codominio.

Sia X il dominio e Y il codominio, la relazione è indicata così: $f: X \rightarrow Y$.

Funzione

Una funzione è una relazione tra due insiemi, chiamati dominio e codominio della funzione, che associa a ogni elemento del dominio uno e un solo elemento del codominio.

Sia X il dominio e Y il codominio, la relazione è indicata così: $f: X \rightarrow Y$.

MARKDOWN PER UN SITO

Markdown viene usato per creare **siti internet**

Siamo un gruppo di studenti che si uniscono a formare un'associazione, di cui dobbiamo gestire un sito

Dobbiamo effettuare una manutenzione costante del sito. Es:

- Aggiungere nuovi eventi
- Aggiornare l'organigramma del direttivo
- Aggiungere nuove pagine per offrire aiuto alle nuove matricole ...

IL PERCHÉ DEL VERSIONING

Roberta viene incaricata di creare una nuova pagina per l'*helpdesk*

Prepara una prima versione e la passa tramite chiavetta USB ad Andrea, il presidente, per la revisione finale

Tuttavia Andrea non è soddisfatto e inizia ad operare modifiche alla versione di Roberta, salvando il file e sovrascrivendolo sulla chiavetta

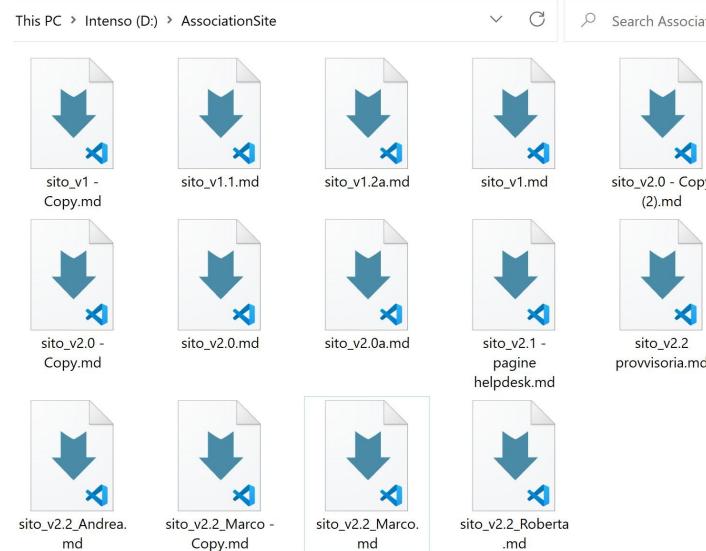
Si accorge però che preferiva la versione precedente, che però non è più disponibile

Il file precedente è perso e non c'è modo di recuperarlo

COME TENERE TRACCIA DELLE VERSIONI DI UN FILE

Avete in mente un modo per poter tenere traccia delle varie versioni di un file?

Tenere varie copie di un file, una per ogni versione



COME TENERE TRACCIA DELLE VERSIONI DI UN FILE – THE RIGHT WAY

La maniera corretta di fare versioning è quella di affidarsi ad uno **strumento** che **tenga traccia della storia di un insieme di file e cartelle**

Un tale strumento viene chiamato SISTEMA DI CONTROLLO VERSIONE o, in inglese, VERSION CONTROL SYSTEM (VCS)

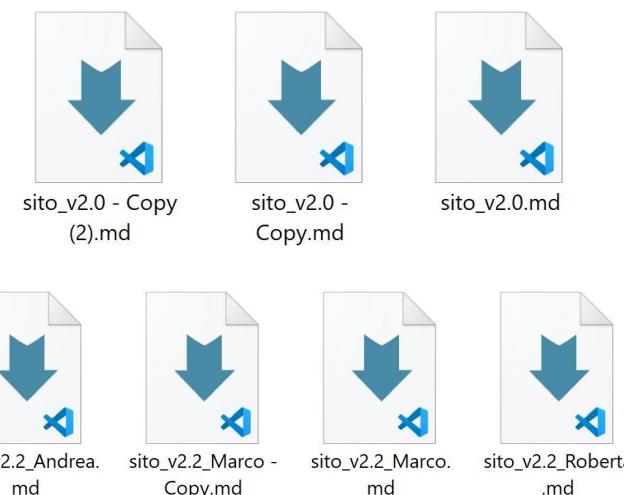
CHE COS'È UN VCS

Un VCS è uno strumento che permette di...

Mantenere uno storico di un insieme di file e cartelle, che chiameremo **repository** o **repo**, tramite opportuni *checkpoint* nel tempo

Tenere più versioni *alternative* degli stessi file

Collaborare con più persone facilitando l'unificazione dei diversi lavori



5. VERSION CONTROL SYSTEMS

Git

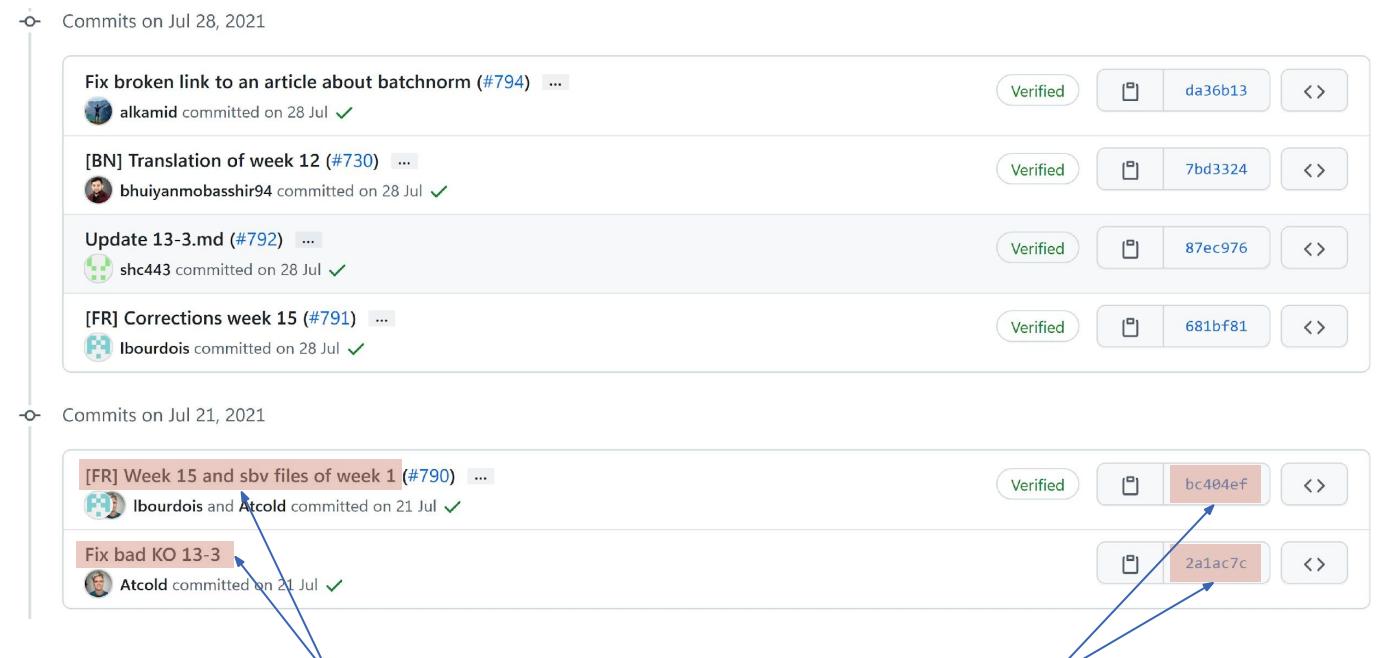


- ❖ VCS ampiamente utilizzato nel mondo dello sviluppo software.
- ❖ Consente di tenere traccia delle modifiche apportate al codice sorgente in modo efficace e collaborativo.
- ❖ Strumento fondamentale per:
 - gestire progetti software
 - monitorare le modifiche
 - facilitare la collaborazione tra sviluppatori.

GESTIONE DELLA STORIA DELLA REPO

Git si occupa di gestire la storia della repo tramite dei *checkpoint*, che in gergo Git si chiamano **commit**

Ogni commit viene creato dall'utente che sta effettuando modifiche sulla repo



Ogni commit è caratterizzato da un messaggio descrivente le modifiche apportate...

...ed è definito univocamente da un codice numerico esadecimale

RECUPERO DI UNA VERSIONE PRECEDENTE

Git è in grado di ricostruire la situazione della repo per ogni commit

Nell'esempio di Andrea e Roberta, se l'associazione avesse tenuto traccia dei file del sito tramite Git, Andrea sarebbe riuscito a recuperare la versione di Roberta semplicemente andando a ripescare l'ultimo commit fatto da Roberta.

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Creazione di una repository su GitHub

CREARE LA REPO SU GITHUB.COM



INTERFACCIA CREAZIONE REPO

Descrizione della repo

Pubblico: visibile da tutti (anche su Google!)
Privato: visibile solo da collaboratori (ma nr. collaboratori limitato con account gratuito)

Funzionalità aggiuntive: vedrete nei corsi futuri a cosa servono e come utilizzarle.
Per adesso lasciamo tutto vuoto

Create a new repository

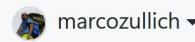
A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *



marcozullich ▾

Repository name *

FakeMathSite

Nome identificativo
della repo (senza spazi).
Non deve essere per
forza uguale al nome
della cartella sul nostro



Great repository names are short and memorable. Need inspiration? How about [animated-computing-machine](#)?

Description (optional)

progetto fittizio per precorsi informatica DMG, a.a. 2021/'22



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:

Skip this step if you're importing an existing repository.



Add a README file

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



Add .gitignore

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



Choose a license

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

Create repository

QUICK SETUP

The screenshot shows a GitHub repository page for 'narcozullich / FakeMathSite' (Private). The 'Code' tab is selected. A blue box at the top right contains the text: 'Ora dobbiamo collegare il nuovo remote alla nostra repo locale'. On the left, a blue box contains the text: 'Selezioniamo HTTPS'. On the right, another blue box contains the text: 'Copiamo questo indirizzo (è l'indirizzo della nostra repo online)'. A large red X is drawn over the command-line instructions at the bottom.

narcozullich / **FakeMathSite** Private

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or **HTTPS** SSH <https://github.com/marcozullich/FakeMathSite.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# FakeMathSite" > README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin https://github.com/marcozullich/FakeMathSite.git  
git push -u origin main
```

Ignoriamo tutto il resto

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Creazione di una repository su GitHub
- Collegamento della repository GitHub (da terminale) con VSCode
 - Otteniamo una copia della repository anche in locale nel nostro PC

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Apriamo il terminal di Visual Studio (View > Terminal)
- Controlliamo che git sia correttamente configurato (**git config --list**)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
● (base) gpietrop@cli-10-110-3-189 ~ % git config --list
credential.helper=osxkeychain
init.defaultbranch=main
user.name=gpietrop
user.email=gloria.pietropolli@gmail.com
core.autocrlf=input
http.postbuffer=1048576000
○ (base) gpietrop@cli-10-110-3-189 ~ % █
```

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Dal terminale, creare ed aprire una repository dove vogliamo aggiungere il contenuto della repository da GitHub
- Clone (**git clone**) the repository usando il link http che si trova su GitHub
- Aprire su Visual Studio la repo appena clonata (se avete usato il terminale di Visual Studio, è sufficiente digitare **code .** sempre sul terminale)

The screenshot shows the GitHub desktop application interface. At the top, it displays a repository with 1 branch and 0 tags. Below the header, there's a list of files: README.md and README. A context menu is open over the README file, with the 'Clone' option highlighted by a large red box. Other options visible in the menu include 'HTTPS', 'SSH', 'GitHub CLI', 'Clone using the web URL.', 'Open with GitHub Desktop', and 'Download ZIP'. The bottom of the window shows some status text: 'allala modifica del file modifica hahahah del file'.

```
(base) gpietro@cli-10-110-3-189 ~ % pwd
/Users/gpietro
(base) gpietro@cli-10-110-3-189 ~ % mkdir repo_to_take
(base) gpietro@cli-10-110-3-189 ~ % cd repo_to_take
(base) gpietro@cli-10-110-3-189 repo_to_take % git clone https://github.com/gpietro/repo_to_take.git
Cloning into 'repo_to_take'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 12 (delta 1), reused 5 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (12/12), done.
Resolving deltas: 100% (1/1), done.
(base) gpietro@cli-10-110-3-189 repo_to_take %
```

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Creazione di una repository su GitHub
- Collegamento della repository GitHub (da terminale) con VSCode
 - Otteniamo una copia della repository anche in locale nel nostro PC
- Modifiche varie alla repository da locale

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Creazione di una repository su GitHub
- Collegamento della repository GitHub (da terminale) con VSCode
 - Otteniamo una copia della repository anche in locale nel nostro PC
- Modifiche varie alla repository da locale
- Push delle modifiche su GitHub

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Dal terminale di Visual Studio
 - **git add .** (aggiungo a git tutte le modifiche che ho fatto)
 - **git commit -m "messaggio di commit."** (committo e lascio un messaggio per ricordarmi cosa ho cambiato rispetto alla versione precedente)
 - **git push** (push delle modifiche, una volta fatto vedremo i cambiamenti anche su GitHub)

```
● (base) gpietrop@cli-10-110-3-189 repo_to_take % git add .
● (base) gpietrop@cli-10-110-3-189 repo_to_take % git commit -m "new commit from local"
[main 3c24081] new commit from local
 1 file changed, 13 insertions(+), 5 deletions(-)
● (base) gpietrop@cli-10-110-3-189 repo_to_take % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/gpietrop/repo_to_take.git
    ed2e456..3c24081 main -> main
○ (base) gpietrop@cli-10-110-3-189 repo_to_take % █
```

PRACTICUM - GIT CON VISUAL STUDIO CODE



In realtà su VisualStudio (come anche in altri editor di testo pensati per fare codice) esiste una sezione dedicata per usare git senza utilizzare il terminale

A screenshot of the Visual Studio Code interface. The left sidebar features the 'SOURCE CONTROL' and 'SOURCE CONTROL GRAPH' sections. The main editor pane displays a Markdown file with code snippets. The bottom right pane shows the 'TERMINAL' with a command-line session for committing changes and pushing them to GitHub. A red brush stroke highlights the 'SOURCE CONTROL' section and the terminal area.

```
(base) gpietro@cli-10-110-3-189 repo_to_take % git add .
(base) gpietro@cli-10-110-3-189 repo_to_take % git commit -m "new commit from local"
[main 3c24081] new commit from local
 1 file changed, 13 insertions(+), 5 deletions(-)
(base) gpietro@cli-10-110-3-189 repo_to_take % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/gpietro/repo_to_take.git
 ed2e456..3c24081 main -> main
(base) gpietro@cli-10-110-3-189 repo_to_take %
```

Ln 13, Col 19 Spaces: 4 UTF-8 LF Markdown

PRACTICUM - GIT CON VISUAL STUDIO CODE



Ora consideriamo il caso in cui abbiamo del codice in locale e vogliamo condividerlo con altri collaboratori su git

- Creiamo la nostra repository in locale (e visto che ormai siamo bravi, creiamola da terminale se siamo in linux)
- Creiamo una repository vuota su GitHub che sarà quella che andremo a collegare con la repository locale
- Apriamo la repository locale con Visual Studio

PRACTICUM - GIT CON VISUAL STUDIO CODE



Ora consideriamo il caso in cui abbiamo del codice in locale e vogliamo condividerlo con altri collaboratori su git

- Creiamo la nostra repository in locale (e visto che ormai siamo bravi, creiamola da terminale se siamo in linux)
- Creiamo una repository vuota su GitHub che sarà quella che andremo a collegare con la repository locale
- Apriamo la repository locale con Visual Studio

PRACTICUM - GIT CON VISUAL STUDIO CODE



Ora consideriamo il caso in cui abbiamo del codice in locale e vogliamo condividerlo con altri collaboratori su git

- Creiamo la nostra repository in locale (e visto che ormai siamo bravi, creiamola da terminale se siamo in linux)
- Creiamo una repository vuota su GitHub che sarà quella che andremo a collegare con la repository locale
- Apriamo la repository locale con Visual Studio

PRACTICUM - GIT CON VISUAL STUDIO CODE



Ora consideriamo il caso in cui abbiamo del codice in locale e vogliamo condividerlo con altri collaboratori su git

- Creiamo la nostra repository in locale (e visto che ormai siamo bravi, creiamola da terminale se siamo in linux)
- Creiamo una repository vuota su GitHub che sarà quella che andremo a collegare con la repository locale
- Apriamo la repository locale con Visual Studio e apriamo il Terminal

PRACTICUM - GIT CON VISUAL STUDIO CODE



- Dal terminale di Visual Studio
 - **git init** (inizializzo la repo come una git repo)
 - **git add .** (aggiungo alla repo git tutti i file contenuti nella directory)
 - **git commit -m "messaggio di commit"** (commit con messaggio)
 - **git remote add origin YOUR_GITHUB_REPO_URL** (connetto la repo locale con la repo su github)
 - **git push -u origin master** (push delle modifiche)

PRACTICUM - GIT CON VISUAL STUDIO CODE



Chiaramente, se il codice viene modificato online vogliamo essere in grado di trasferire i cambiamenti anche nella nostra repo locale

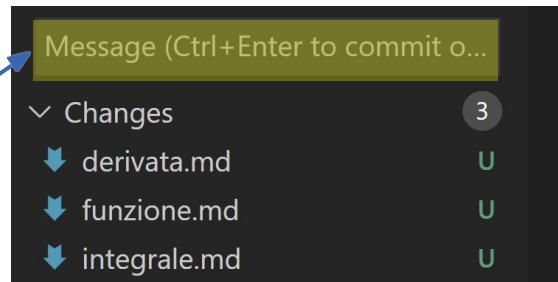
- Modifichiamo da GitHub la repository
- **git pull** (sempre da terminale VS, per prendere le modifiche da origin)

ESECUZIONE DEL COMMIT

Reminder:



Il codice alfanumerico viene aggiunto automaticamente da Git, il messaggio dobbiamo aggiungerlo noi



Scriviamo «primo commit» come messaggio; digitiamo Alt+Enter per confermare il commit



Congratulazioni, abbiamo creato il nostro primo commit!

BRANCH

Un branch è una versione parallela del progetto che permette di lavorare su nuove funzionalità senza modificare codice principale. Vediamo qualche esempio su GitHub...

- **git branch name_branch** (creazione di un nuovo branch)
- **git checkout name_branch** (spostarsi sul branch name_branch)
- **git checkout -b name_branch** (creare e spostarsi su un nuovo branch)
- **git branch** (ottenere una lista di tutti i branch)

BRANCH - ESEMPIO DI CREAZIONE DI NUOVO

- **git checkout -b feature-branch** creiamo un nuovo branch che si chiama feature-branch ed entriamo direttamente lì a lavorare
 - Commit del nuovo branch!!
 - **git push -u origin feature-branch**
- **echo "This is a new feature" > feature.txt** aggiungiamo un nuovo file feature.txt e aggiungiamo del contenuto nel file
- **git add feature.txt** aggiungiamo il file su git nel branch
- **git commit -m "Added feature.txt"** commettiamo I cambiamenti del branch
- **git push** pushiamo I cambiamenti in locale

MERGE

Cos'è il merge in Git?

- Il merge è l'operazione che permette di unire i cambiamenti di due branch diversi in uno solo.
- È fondamentale per il lavoro collaborativo e per integrare nuove funzionalità sviluppate su branch paralleli.

Sintassi di base

```
git merge <branch-da-unire>
```

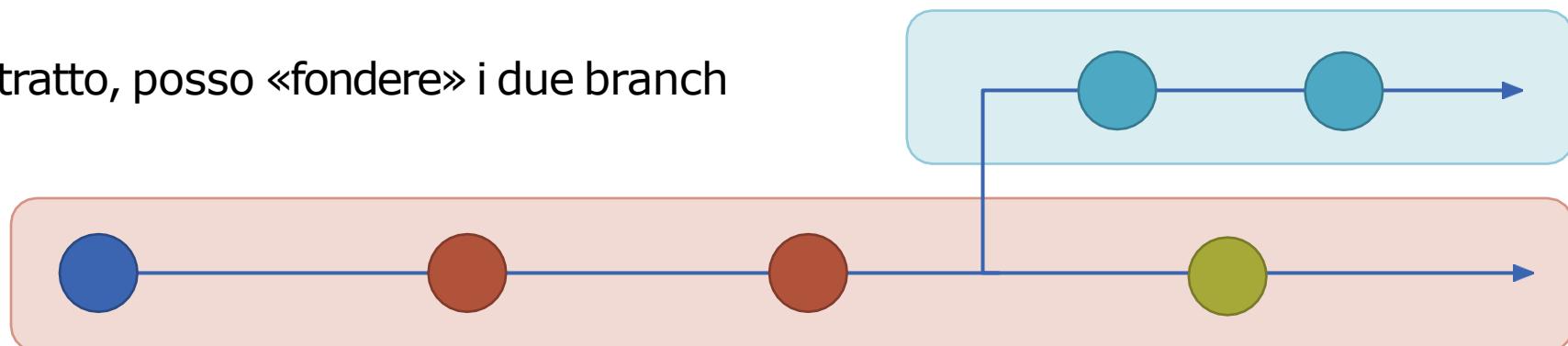
Esempio

```
git merge feature-branch
```

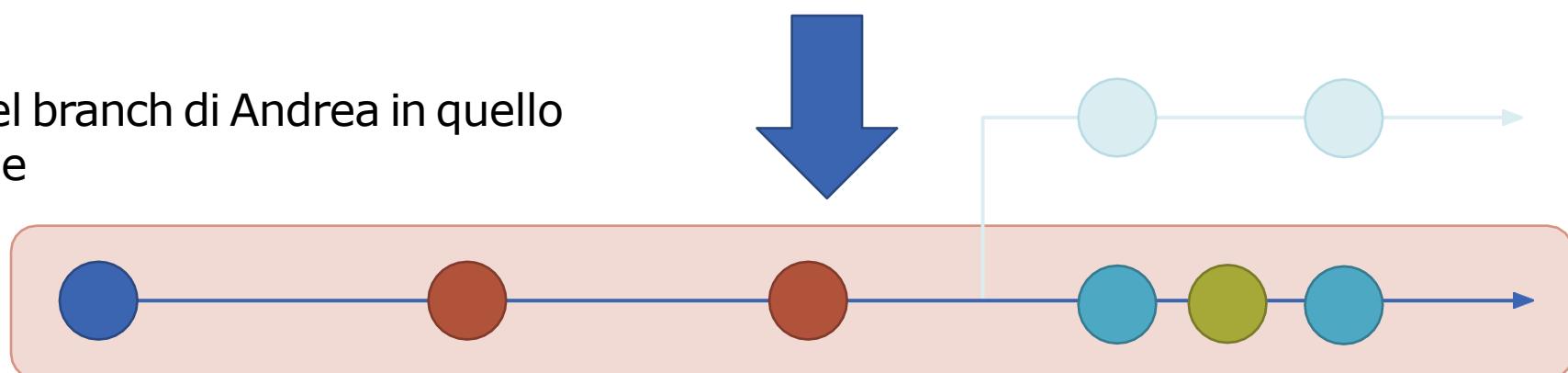
Unisce il branch feature-branch nel branch attuale.

MERGE

Ad un certo tratto, posso «fondere» i due branch assieme



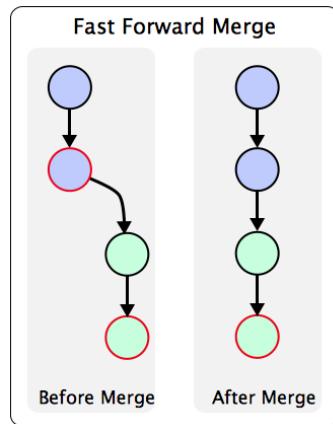
Merge del branch di Andrea in quello principale



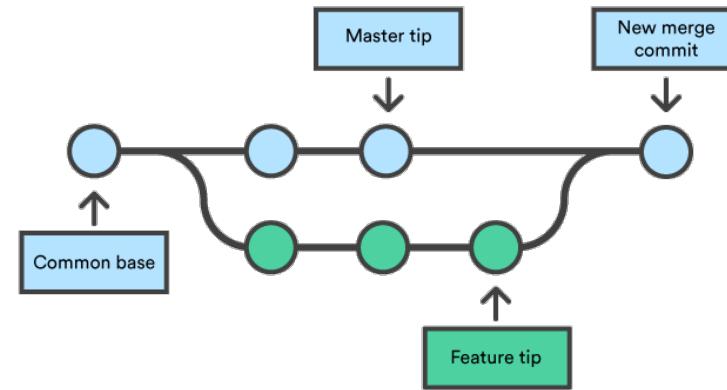
Il branch di Andrea continua comunque ad esistere

MERGE

Fast Forward



Merge different commit



- Avviene quando il branch corrente non ha commit successivi rispetto al branch da unire.
- Il puntatore `HEAD` viene semplicemente spostato in avanti.

Se i due branch hanno commit diversi, Git crea un nuovo commit di merge per unire i cambiamenti.

NON E SEMPRE COSI FACILE

Ci possono essere dei conflitti!!!

Git ti notifica se non è in grado di unire automaticamente i file.

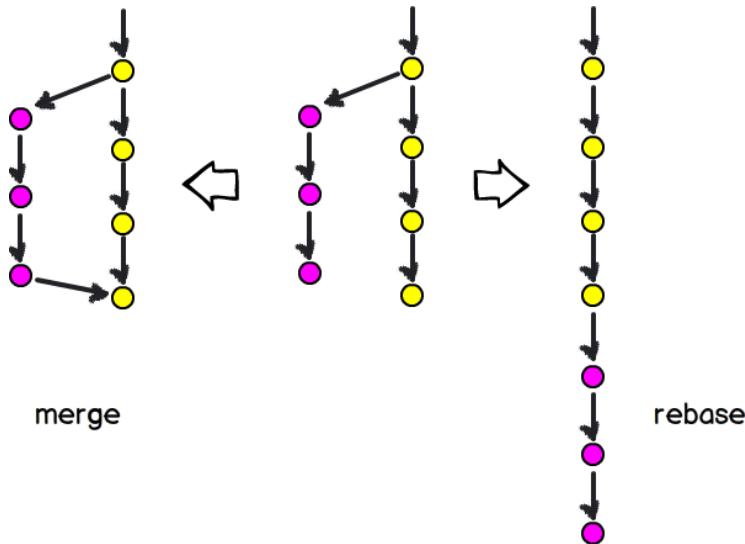
Come risolvere i conflitti:

- Modifica i file per risolvere manualmente le differenze.
- Usa `git add <file>` per segnare il conflitto come risolto.
- Completa il merge con `git commit`.

Consigli utili

- **Prima del merge:** Assicurati che il branch di destinazione sia aggiornato usando `git pull`.
- **Evita conflitti frequenti:** Mantenere i branch aggiornati riduce la probabilità di conflitti.

REBASE



Cosa è il rebase?

Il rebase sposta una serie di commit da un branch sopra un altro. A differenza del merge, non crea un nuovo commit di merge, ma "ricostruisce" la cronologia in modo lineare

Esempio:

Siamo nel branch `feature-branch` e vogliamo riallineararlo con `main`.

```
git checkout feature-branch  
git rebase main
```

Tutti i commit di `feature-branch` verranno applicati sopra `main`, riscrivendo la cronologia.

Rebase vs. Merge

- **Rebase:** Riscrive la cronologia, creando una storia lineare.
- **Merge:** Mantiene la cronologia completa, con commit di merge espliciti.

BRANCH - MERGE DEL BRANCH NEL MAIN

- **git checkout main** switch nuovamente al main branch
- **git merge feature-branch** uniamo il feature-branch nel main (non dovremmo avere conflitti in questo caso)
 - Ricordiamoci di pushare!!!!
- **git log --oneline —graph** controlliamo la storia git per confermare che abbiamo fatto i cambiamenti che volevamo fare

MA SE VOGLIAMO TORNARE INDIETRO?

- HEAD rappresenta il commit attualmente "puntato" nel tuo repository.
- Quando fai un commit, HEAD si sposta al nuovo commit.
- Puoi spostare HEAD per tornare a versioni precedenti o su branch differenti.

Comandi per spostare **HEAD**:

1. **git checkout <commit>**

- Sposta HEAD su uno specifico commit.
- **Esempio:** `git checkout 123abc`
 - Porta il repository allo stato del commit con hash 123abc.

2. **git checkout <branch>**

- Cambia branch, spostando HEAD su un altro branch.
- **Esempio:** `git checkout main`
 - Sposta HEAD sul branch main.

3. **git reset --hard <commit>**

- Sposta HEAD a un commit specifico e **ripristina i file** al loro stato in quel commit.
- **Esempio:** `git reset --hard 123abc`
 - ATTENZIONE: Cancella tutte le modifiche non committate!

MA SE VOGLIAMO TORNARE INDIETRO?

Vogliamo tornare al main prima di fare il merge perchè abbiamo capito che la nuova feature non era valida

- **git log —online** per avere le informazioni sui commit e in particolare sul loro numero identificativo
- **git reset commit_id** (nel caso in cui nel frattempo si abbiano fatto cambiamenti che non vogliamo tenere **git reset —hard commit_id**)
- **git push —force** in questo caso dobbiamo forzare il push perchè stiamo tornando indietro e rimuovendo file

GIT BASICS

- `git help <command>`: get help for a git command
- `git init`: creates a new git repo, with data stored in the `.git` directory
- `git status`: tells you what's going on
- `git add <filename>`: adds files to staging area
- `git commit`: creates a new commit
 - Write [good commit messages!](#)
 - Even more reasons to write [good commit messages!](#)
- `git log`: shows a flattened log of history
- `git log --all --graph --decorate`: visualizes history as a DAG
- `git diff <filename>`: show changes you made relative to the staging area
- `git diff <revision> <filename>`: shows differences in a file between snapshots
- `git checkout <revision>`: updates HEAD and current branch

GIT BRANCHING AND MERGING

- `git branch`: shows branches
- `git branch <name>`: creates a branch
- `git checkout -b <name>`: creates a branch and switches to it
 - same as `git branch <name>; git checkout <name>`
- `git merge <revision>`: merges into current branch
- `git mergetool`: use a fancy tool to help resolve merge conflicts
- `git rebase`: rebase set of patches onto a new base

GIT REMOTES

- `git remote`: list remotes
- `git remote add <name> <url>`: add a remote
- `git push <remote> <local branch>:<remote branch>`: send objects to remote, and update remote reference
- `git branch --set-upstream-to=<remote>/<remote branch>`: set up correspondence between local and remote branch
- `git fetch`: retrieve objects/references from a remote
- `git pull`: same as git fetch; git merge
- `git clone`: download repository from remote

GIT UNDO

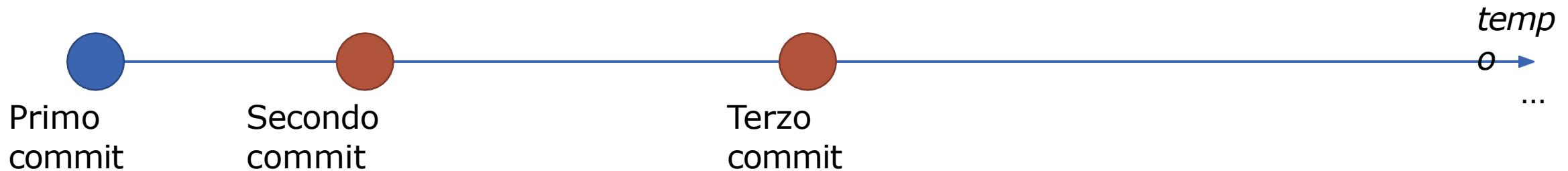
- `git reset HEAD <file>`: unstage a file
- `git commit --amend`: edit a commit's contents/message
- `git checkout -- <file>`: discard changes

LET US PRACTICE :)

Cerchiamo Learn Git Branching su Google e
iniziamo a fare qualche esercizio

VISUALIZZAZIONE DELLA STORIA DELLA REPO

Per le funzionalità che abbiamo visto ora di Git, possiamo visualizzare lo sviluppo di una repo come dei punti su una semiretta ordinata nel tempo



REPO LOCALI NON COINCIDENTI

Ogni repo che utilizza lo stesso remote avrà caricata una storia dei commit al suo interno

Ma non è detto che la storia coincida del tutto a quella del remote!

Esempio

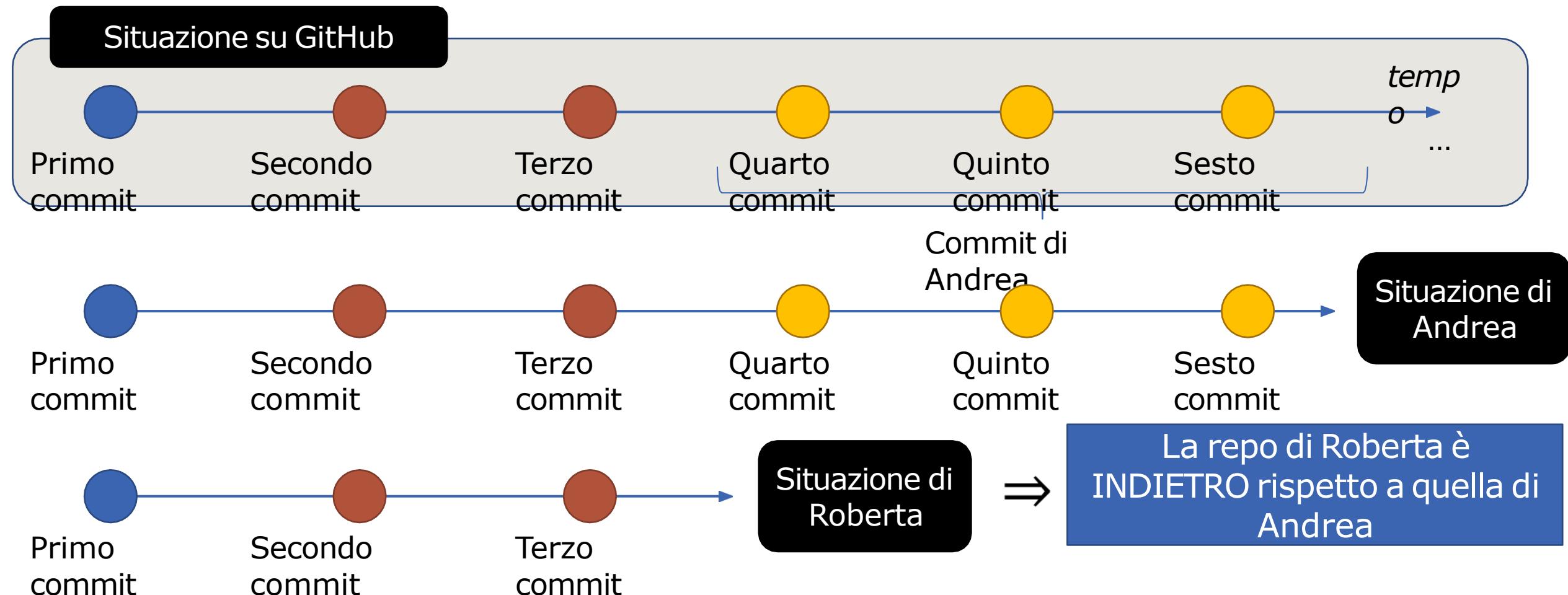
Andrea e Roberta stanno lavorando sulla stessa repo del sito dell'associazione

Roberta va in ferie per 2 settimane

Nel frattempo, Andrea decide invece di lavorare e *pusha* 3 commit sul remote

Domanda: come saranno la repo di Roberta e la repo di Andrea al rientro di Roberta dalle ferie?

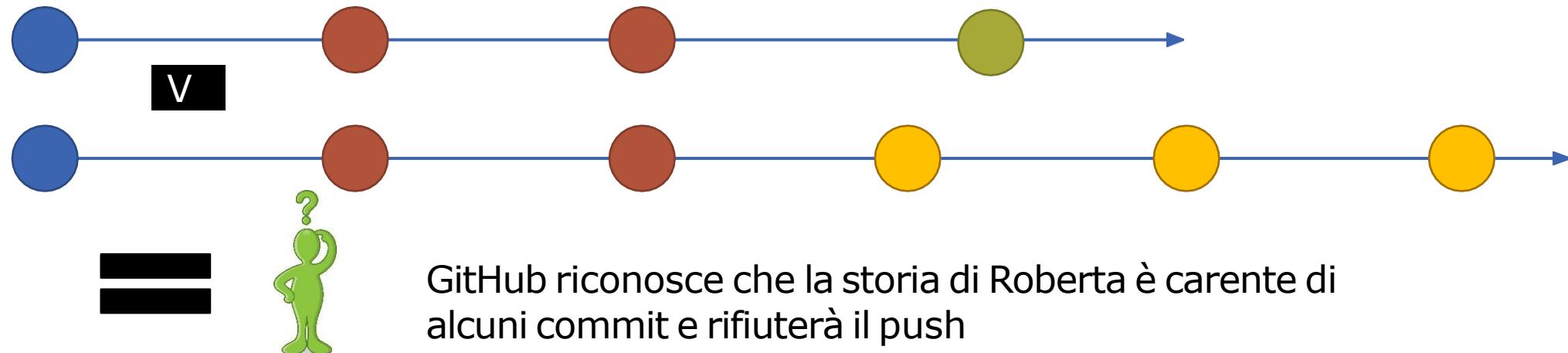
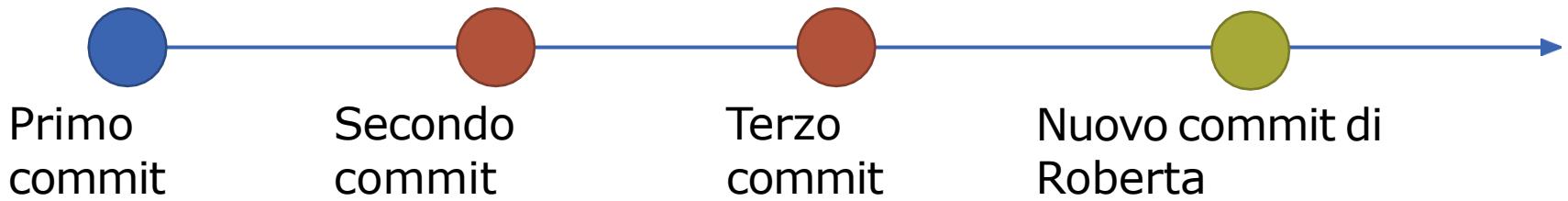
VISUALIZZAZIONE DELLE REPO DI ANDREA E ROBERTA



PROBLEMA DEL LAVORO COLLABORATIVO

Roberta si mette al lavoro e decide di fare un nuovo commit. La sua situazione è ora questa

Roberta decide di pushare il suo lavoro. Cosa pensate succederà?



CHE COSA AVREBBE DOVUTO FARE ROBERTA?

Roberta, al suo rientro dalle ferie, avrebbe dovuto innanzitutto fare un pull per mettere la sua repo in pari con quella di GitHub (e di Andrea)

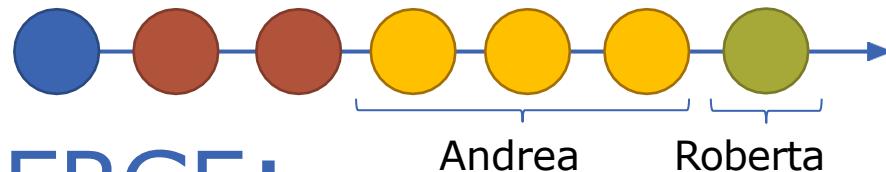
Prima di mettersi al lavoro, è buona norma fare un pull per evitare queste situazioni

MERGE E MERGE CONFLICT

E ora?

Roberta dovrà fare un pull per integrare le modifiche di Andrea
Tuttavia, noi non sappiamo quali file siano stati modificati da Andrea
Potrebbe essere che le due situazioni non siano compatibili
Es. Andrea e Roberta hanno entrambi modificato lo stesso
paragrafo di un file

Se le modifiche sono compatibili



MERGE:

Git integra i commit di andrea e vi
pospone il commit di Roberta

Se le modifiche sono incompatibili

Git non sa come integrare il commit
di Roberta con quelli di Andrea

Si verifica un cosiddetto **merge conflict**

I merge conflict vanno risolti a mano da
Roberta indicando quali parti della versione
di Andrea e di quali della versione di
Roberta vanno mantenute

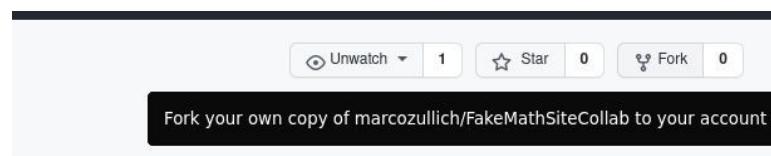
5. VERSION CONTROL SYSTEMS

Azioni repo da remoto

FORK



GitHub ci dà inoltre la possibilità di fare un fork della repository

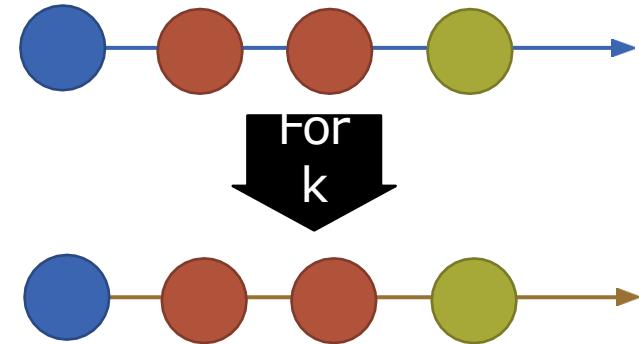


Un fork è una copia remota e personale della repo

Possiamo usarla per scaricarci una repo pubblica di un altro programmatore

E modificarla secondo le nostre necessità

github.com/TizioCaio/cool_program.git



github.com/marcozullich/cool_program.git

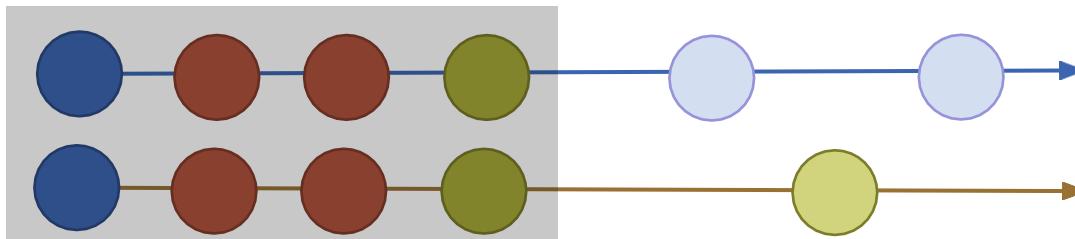
STORIE IN COMUNE

Un fork può essere visto come un branch della repo principale Solo che la repo viene "incollata" in un'altra repository

E al suo interno vengono copiate anche tutti i branch già esistenti nella repo principale

Git è in grado di riconoscere che una repo è un fork di un'altra in quanto hanno una storia in comune

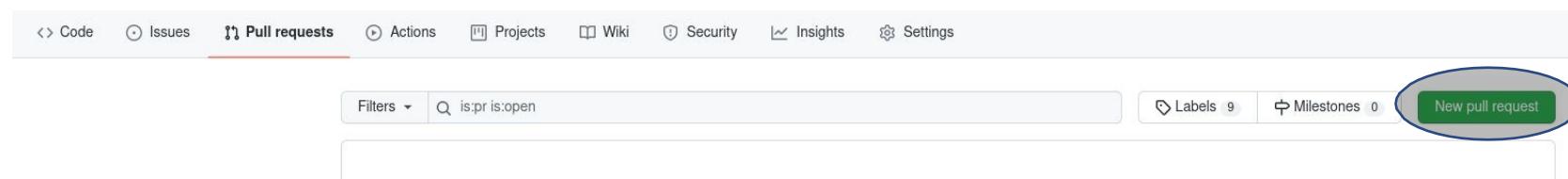
github.com/TizioCaio/cool_program.git



github.com/marcozullich/cool_program.git

PULL REQUEST (I)

Posso fare il merge di due repo di cui una è fork di un'altra tramite un'azione detta Pull Request



Tramite la pull request, io richiedo al/ai proprietario/i di una repo di integrare le modifiche da me effettuate nella sua repo principale

Praticamente, funziona come un merge, ma la cosa viene fatta tutta a livello remoto (risoluzione conflitti, commit...)

PULL REQUEST (I)

La Pull Request è in generale uno strumento per effettuare un merge di due branch in maniera remota



NB: questo significa che le modifiche apportate dal merge devono venir integrate a livello locale con un *pull*

Il branch può provenire sia da un fork della repo che dalla stessa repo

base: main ▾ ← compare: ramo ▾ ✓ Able to merge.

Pull request per merge di due branch (main e ramo) della stessa repo

base repository: Atcold/pytorch-Deep-Learning ▾ ← base: master ▾ head repository: marcozullich/pytorch-Deep-Lea... ▾ compare: 09_part2 ▾

Pull request per merge di due branch (09_part2 e master) di due repo differenti, una il fork di un'altra