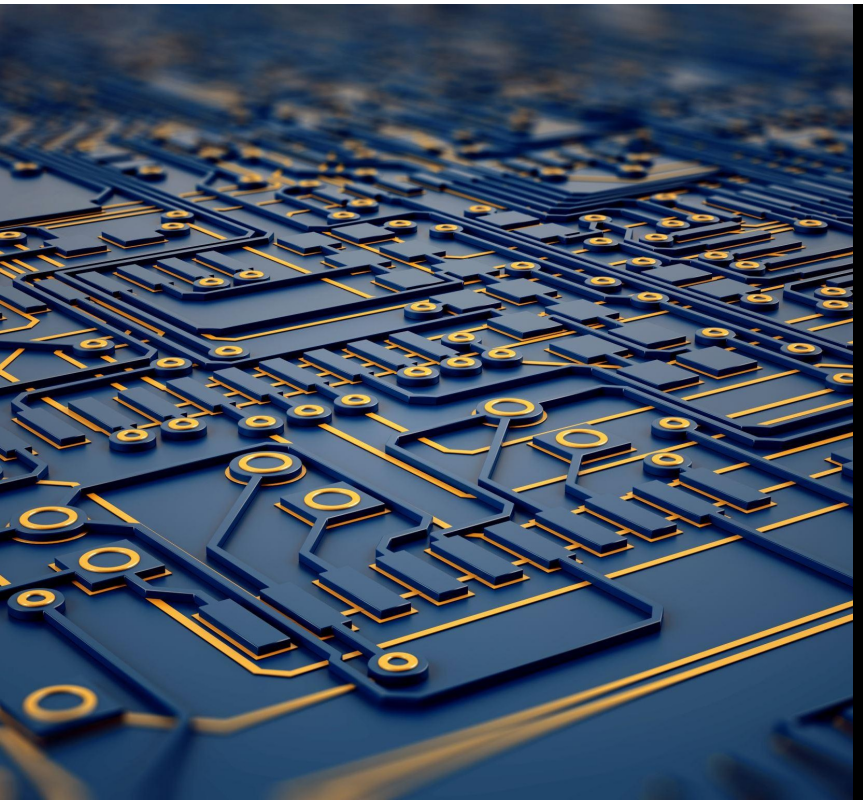


INFORMATICA

Intelligenza Artificiale & Data Analytics

Precorsi a.a. 2025/26

Docente: Gloria Pietropolli

5. VERSION CONTROL SYSTEMS

Richiami di markdown e la necessità del *versioning*

COME TENERE TRACCIA DELLE VERSIONI DI UN FILE

Molti dei files con cui lavorerete sono file di testo

- Source code
- Documentation

Spesso modificherete questi file nel corso di giorni/mesi/anni

COME TENERE TRACCIA DELLE VERSIONI DI UN FILE

Mentre cambiate questi file è possibile che vogliate tenere traccia delle diverse versioni di un file

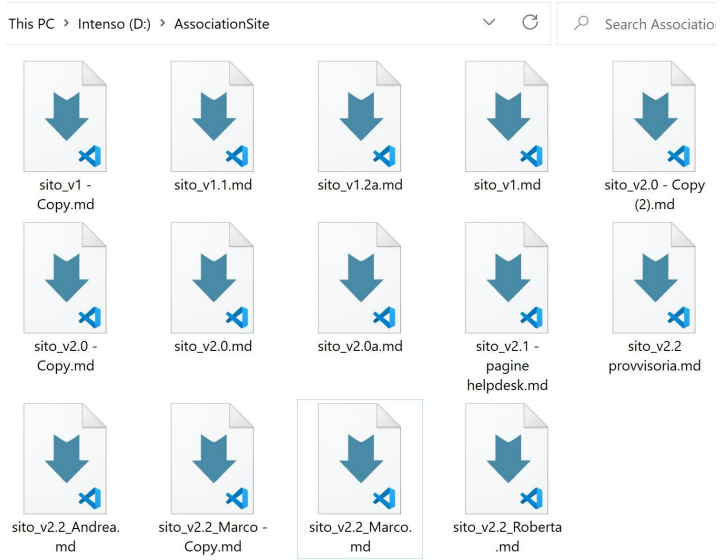
Perché?

- Rompete qualcosa, e volete tornare a come era il progetto ieri
- Volete provare qualcosa di nuovo che potete annullare se non funziona
- Qualcuno vuole runnare il tuo codice? Lo indirizzate a una versione stabile

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

Esiste un modo di tenere traccia automaticamente invece che manualmente & nel cloud?

Un VERSION CONTROL SYSTEM (VCS) è un software che gestisce diverse versioni dei file e cartelle al posto vostro

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

Un buon sistema di controllo versione:

- Salverà molte versioni dei tuoi file
- Ti permetterà di **“ripristinare”** un file (o una parte di esso) a una versione precedente
- Terrà traccia delle modifiche tra le diverse versioni
- Garantirà che ogni “versione” non sia né troppo grande né troppo piccola
- Supporterà il backup remoto delle versioni sul cloud

Un ottimo sistema di controllo versione:

- Ti permetterà di collaborare sui file con altre persone
- Combinerà in modo sensato le diverse versioni dei file prodotte da persone diverse

5. VERSION CONTROL SYSTEMS

Git

git

Git è un sistema di controllo versione che tiene traccia dei "commit" (istantanee) dei file in un REPOSITORY.

- Git memorizza le vecchie versioni dei file in una cartella nascosta (.git) e le gestisce automaticamente.
- Possiamo dire a Git di tenere traccia di determinati file e di quando effettuare un'istantanea.
- Possiamo chiedere a Git di tornare a una vecchia istantanea (anche solo per un singolo file).
- Possiamo chiedere a Git di tenere traccia di chi sta lavorando su cosa, in modo che più persone possano lavorare su cose diverse senza conflitti.

GESTIONE DELLA STORIA DELLA REPO

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

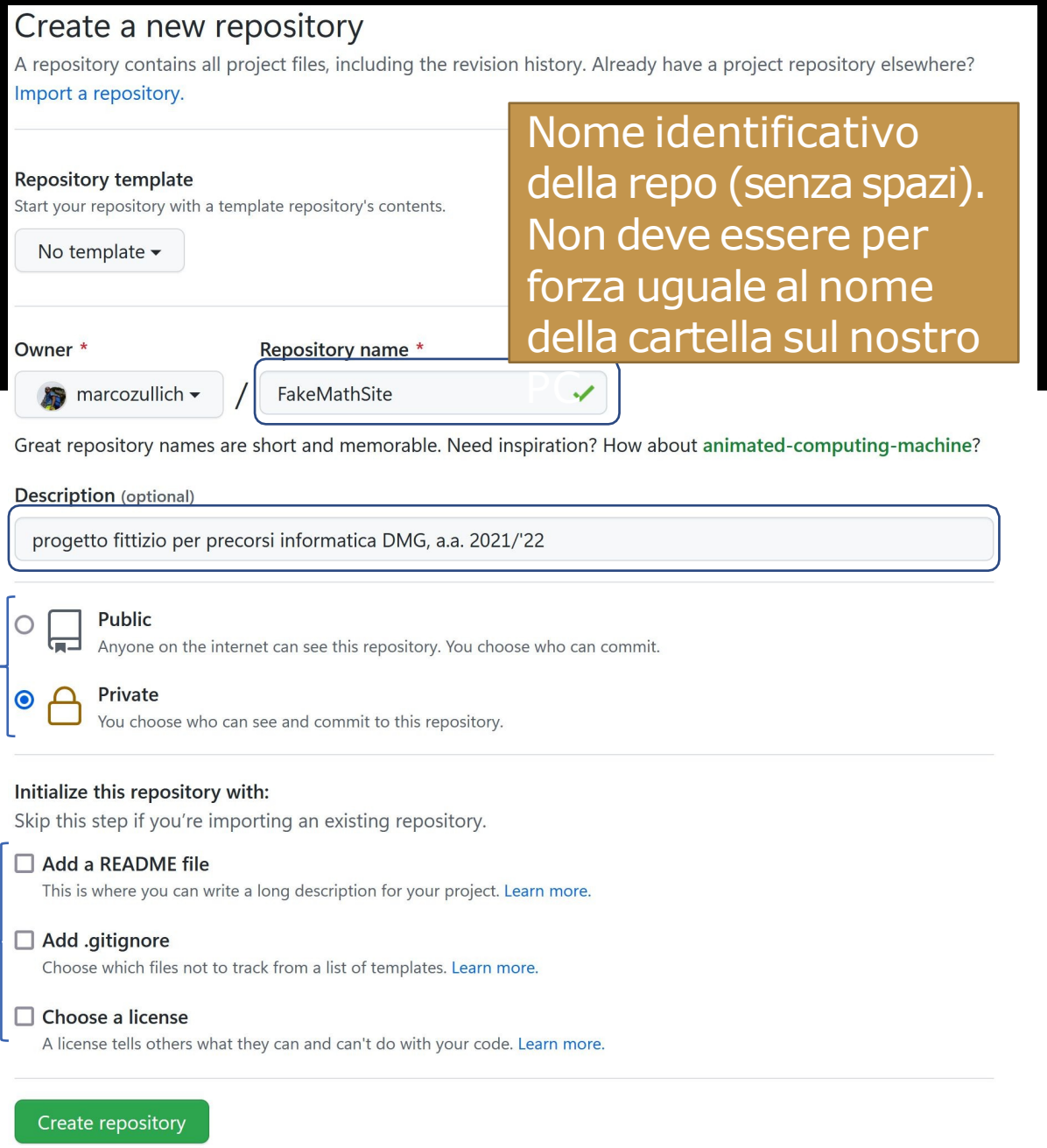
Ogni commit viene creato dall'utente che sta effettuando modifiche sulla repo

PRACTICUM – GIT CON VISUAL STUDIO CODE

- Creazione di una repository su GitHub

CREARE LA REPO SU GITHUB.COM

INTERFACCIA CREAZIONE REPO



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

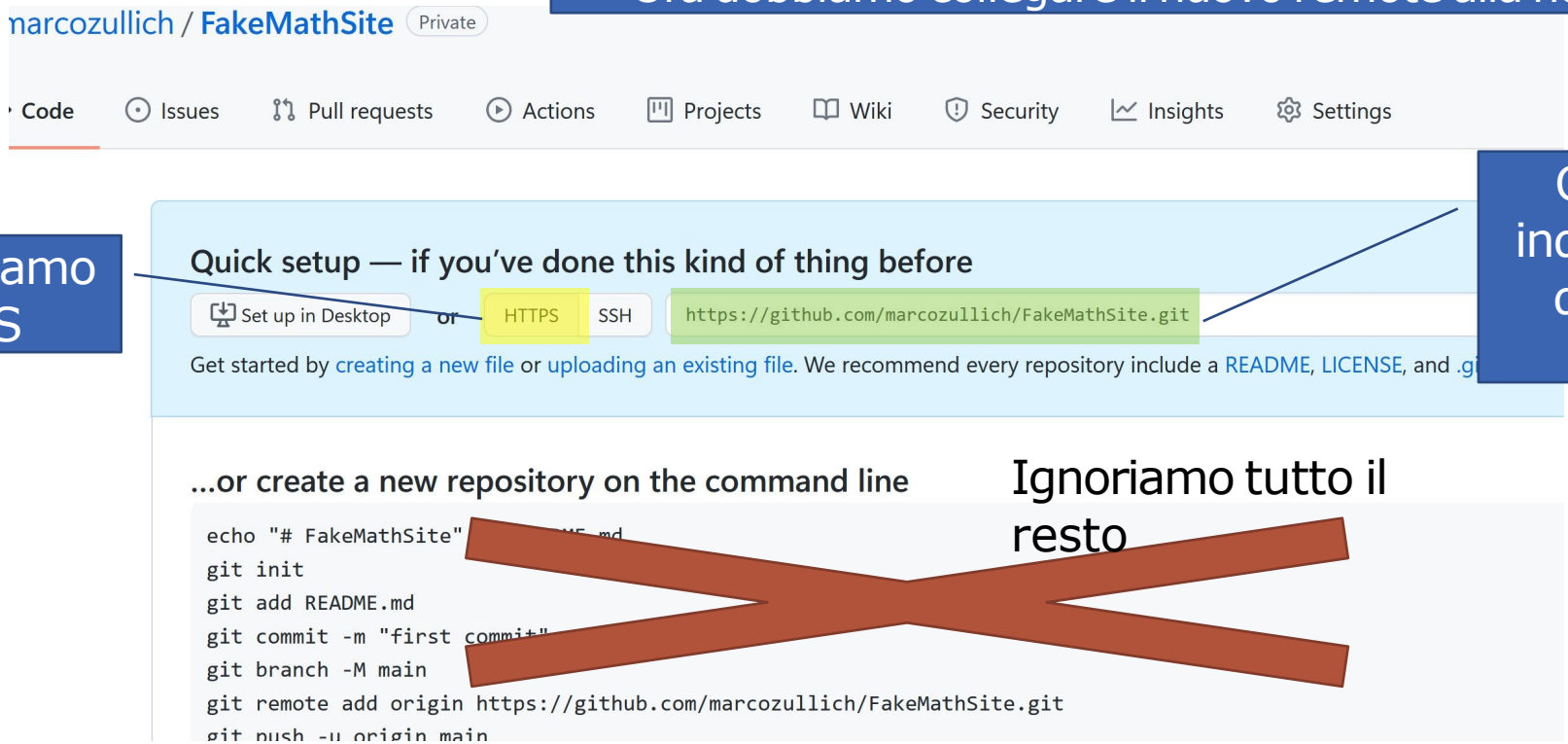
Descrizione della repo

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

QUICK SETUP

Ora dobbiamo collegare il nuovo remote alla nostra repo locale



Selezioniamo HTTPS

Copiamo questo indirizzo (è l'indirizzo della nostra repo online)

Ignoriamo tutto il resto

PRACTICUM – GIT CON VISUAL STUDIO CODE

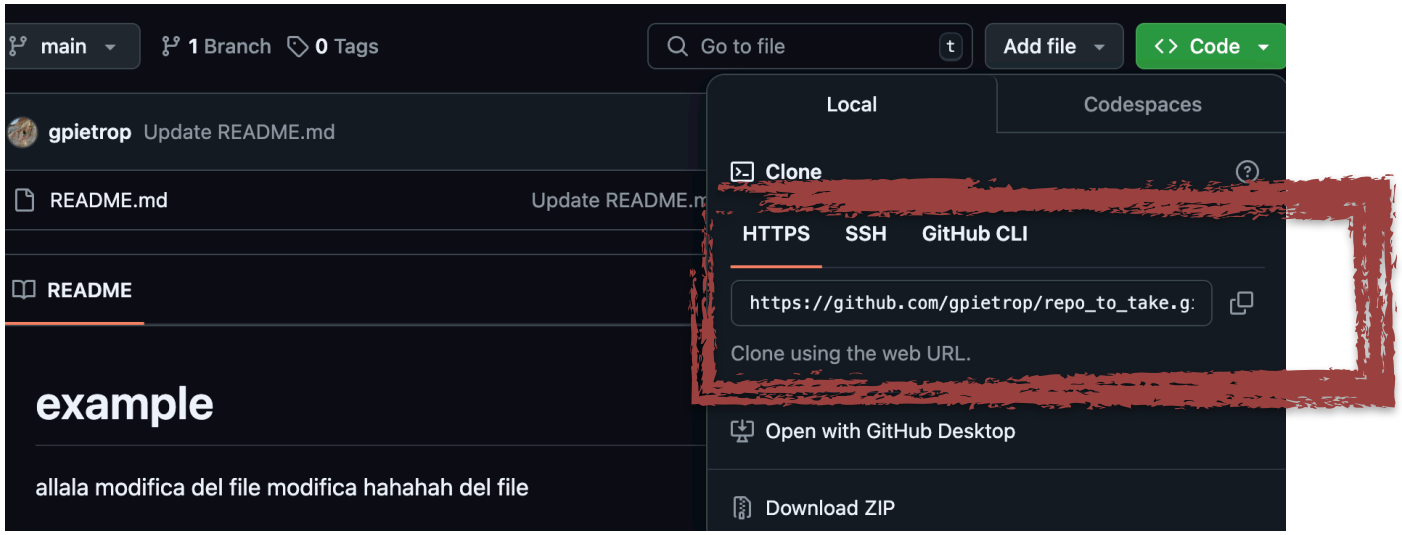
- Creazione di una repository su GitHub
- Collegamento della repository GitHub (da terminale) con VSCode
  - Otteniamo una copia della repository anche in locale nel nostro PC

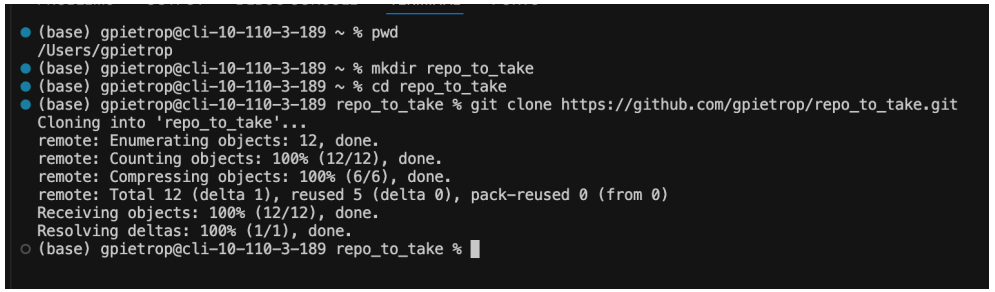
PRACTICUM – GIT CON VISUAL STUDIO CODE

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

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)





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

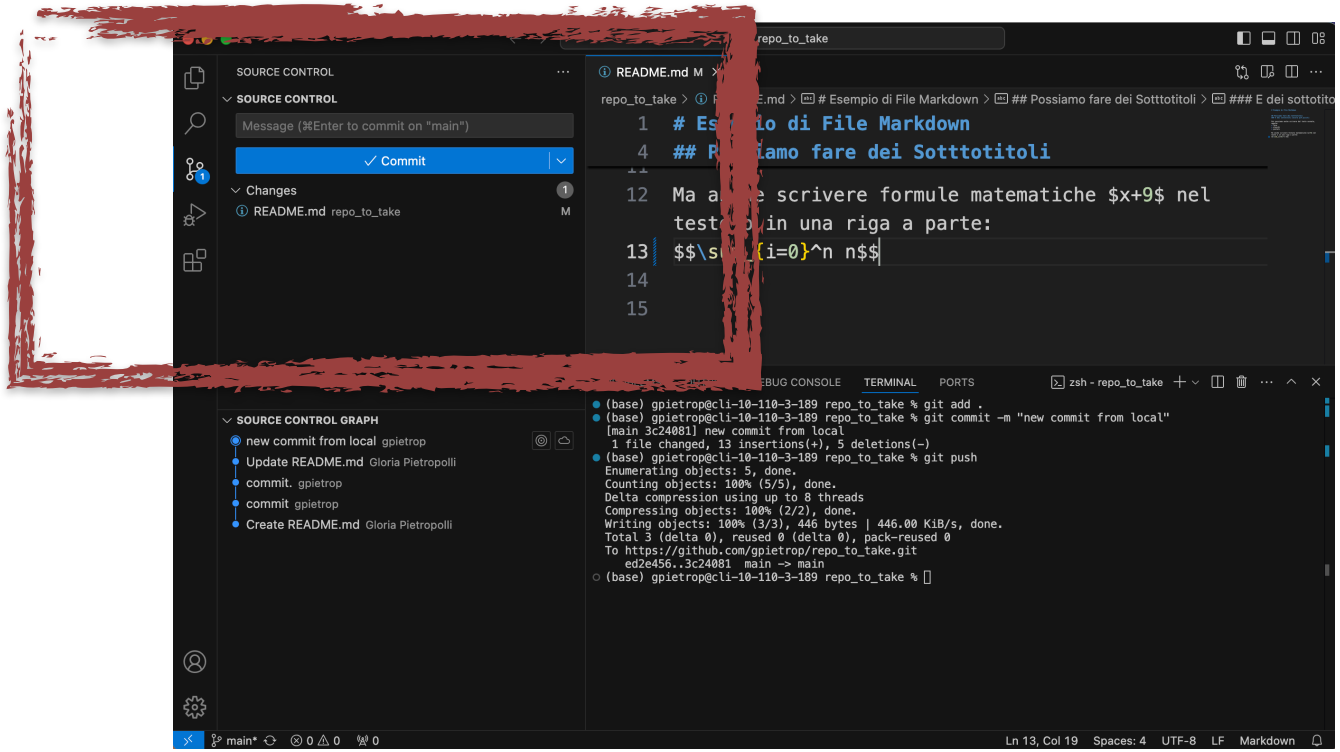
PRACTICUM – GIT CON VISUAL STUDIO CODE

- Dal terminale di Visual Studio
  - Salvare tutti i file creati
  - **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



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 main** (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)

PRACTICUM – GIT CON VISUAL STUDIO CODE

git clone

- Crea una **copia completa** di una repository esistente.
- Copia sia i file che tutta la cronologia dei commit.
- Si usa quando vogliamo iniziare a lavorare su un progetto già presente su GitHub (o altrove).

1. `git clone URL`

PRACTICUM – GIT CON VISUAL STUDIO CODE

git add

- Aggiunge file (o modifica file già tracciati) all'area di stage
- Passaggio intermedio prima del commit

1. `git add .` → aggiunge tutti i file modificati e nuovi nella cartella corrente.
2. `git add nomefile` → aggiunge un singolo
3. `git add -A` → aggiunge tutti i file del progetto, inclusi eliminati.

PRACTICUM – GIT CON VISUAL STUDIO CODE

git commit

- Registra in maniera permanente le modifiche
- Crea un "fotogramma" della situazione attuale dei file.
- Ogni commit ha:
  - un hash univoco (es. `a3f1c9d…`)
  - un **autore**
  - una **data**
  - un messaggio descrittivo

1. `git commit -m "Messaggio del commit"`

Buone pratiche per il messaggio:

- Deve spiegare **cosa** hai fatto, non come
- Breve ma chiaro (massimo 50–70 caratteri)
- Evitare messaggi generici tipo "update" o "fix"

PRACTICUM – GIT CON VISUAL STUDIO CODE

git push

- Invia i commit dal **repository locale** al **repository remoto**.
- Serve per **condividere** il lavoro con altri o salvarlo online.

1. `git push origin main`

`origin` → nome predefinito del repository remoto (GitHub).

`main` → il branch su cui stai lavorando.

git pull

- Aggiorna il repository locale con le modifiche del repository remoto.

1. `git pull origin branch`

PRACTICUM – GIT CON VISUAL STUDIO CODE

git status

- Mostra lo **stato attuale** della repository.
- Indica:
  - se ci sono modifiche non aggiunte allo stage
  - se ci sono file nello stage pronti per il commit
  - su quale branch ti trovi
  - se il branch è aggiornato con il remoto

git log

- Per vedere la cronologia dei commit.

1. `git log –oneline`

PRACTICUM – GIT CON VISUAL STUDIO CODE

`git log` → mostra la cronologia dei commit

- `git log --oneline` → versione compatta

`git diff` → mostra le differenze tra versioni

- `git diff` → modifiche non ancora aggiunte
- `git diff --staged` → modifiche già nello stage

`git rm file.txt` → elimina un file tracciato

`git mv vecchio.txt nuovo.txt` → rinomina/sposta un file

`git restore file.txt` → annulla modifiche non aggiunte

`git restore --staged file.txt` → rimuove un file dallo stage

`git remote -v` → mostra i repository remoti collegati

MA SE VOGLIAMO TORNARE INDIETRO?

- HEAD rappresenta il commit attualmente "puntato" nel repository.
- Ogni volta che fai un commit, HEAD si sposta sul nuovo commit.
- Puoi spostare HEAD per:
  - tornare a versioni precedenti
  - muoverti tra branch diversi

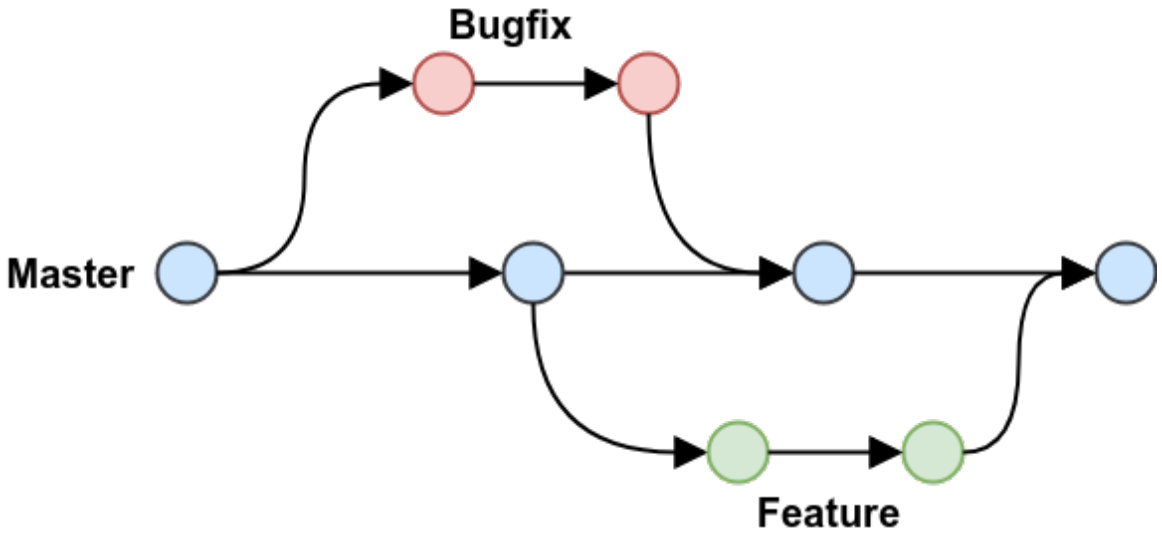
Comandi per spostare `HEAD`:

1. `git checkout <commit>`
   1. Sposta `HEAD` su uno specifico commit
   2. Esempio: git checkout 123abc
2. `git reset –hard <commit>`
   1. Sposta `HEAD` a un commit specifico e ripristina i file al loro stato in quel commit.
   2. Esempio: git reset --hard 123abc
   3. ATTENZIONE: Cancella tutte le modifiche non committate e i commit successivi

Per tornare indietro

`git checkout main`

BRANCH



- Un branch è una linea di sviluppo indipendente.
- Permette di lavorare su nuove funzionalità senza toccare subito il codice principale.
- Il branch di default si chiama `main`.

BRANCH

- `git branch nome-branch` creazione di un nuovo branch
- `git checkout nome-branch` spostarsi sul branch name_branch
- `git checkout -b nome-branch` creare e spostarsi su un nuovo branch
- `git branch` ottenere una lista di tutti i branch
- `git branch -d nome-branch` rimuovere un branch

BRANCH - ESEMPIO DI CREAZIONE DI NUOVO

- **git checkout -b feature-branch** creiamo un nuovo branch che si chiama feature-branch ed entriamo direttamente li a lavorare
- **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”** committiamo 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>
```

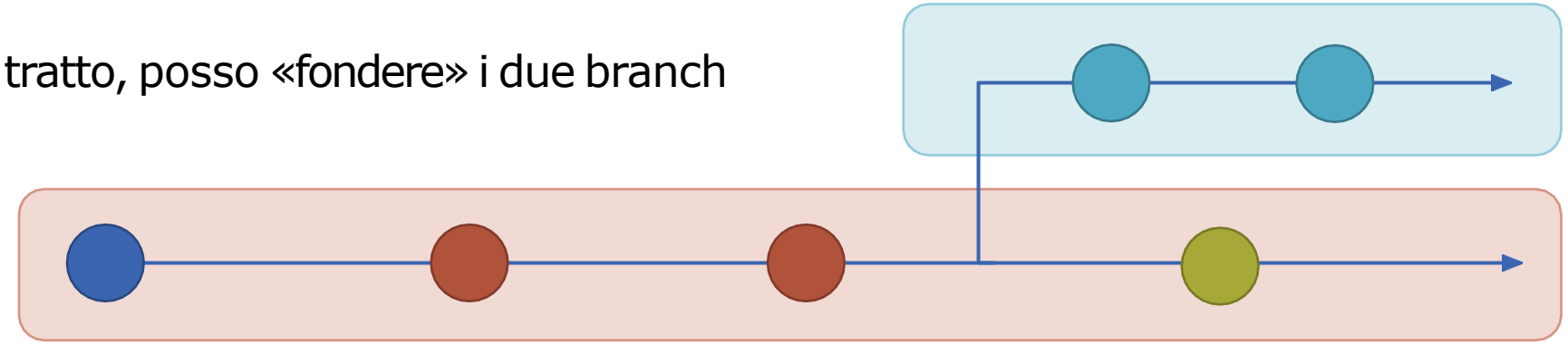
Esempio

```
git merge feature-branch
```

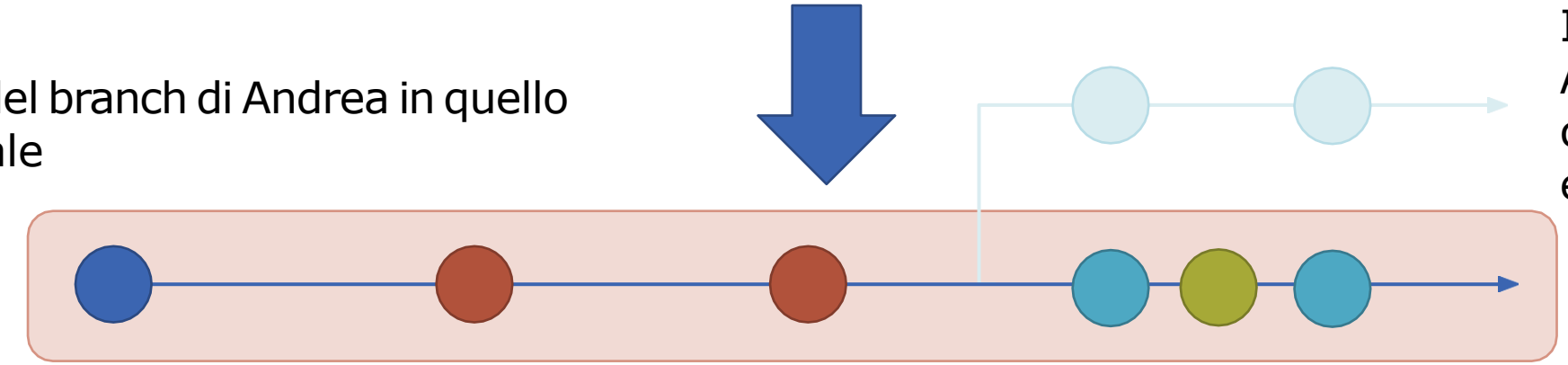
Unisce il branch `feature-branch` nel branch attuale.

MERGE

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



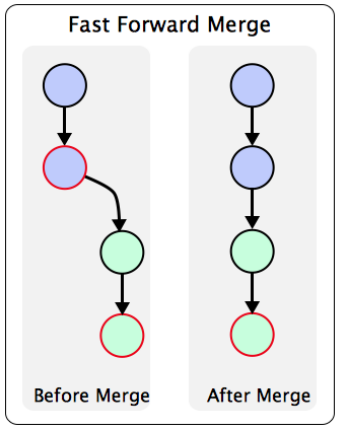
Merge del branch di Andrea in quello principale



Il branch di Andrea continua comunque ad esistere

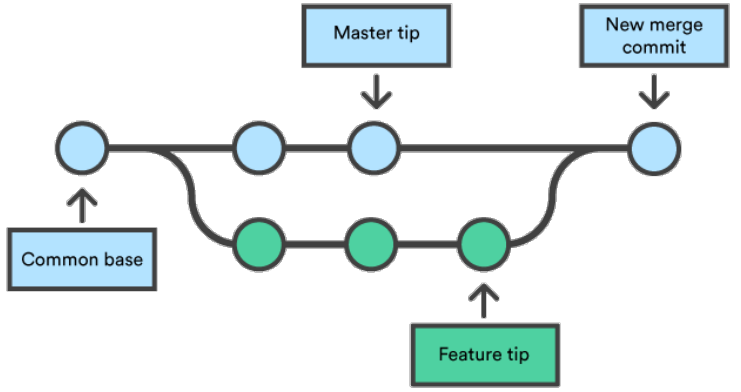
MERGE

Fast Forward



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

Merge different commit



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.

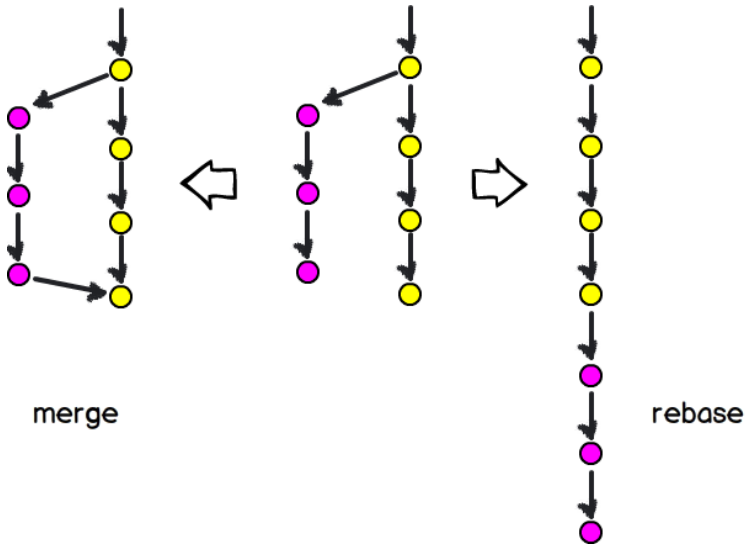
Come risolvere i conflitti:

- 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 riallinearlo 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

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 :)

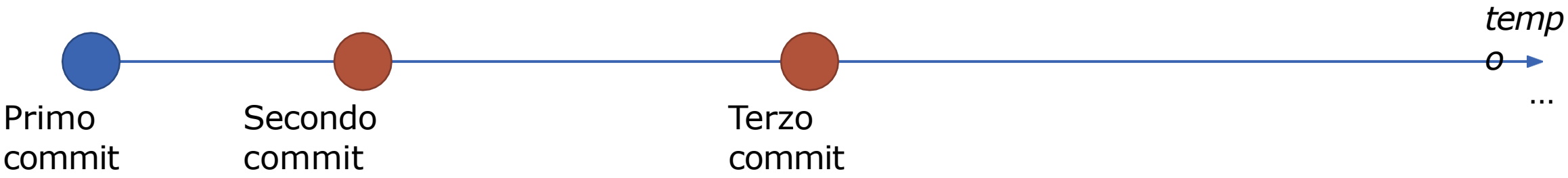
Cerchiamo Learn Git Branching su Google e iniziamo a fare qualche esercizio

LET US PRACTICE :)

1. Su GitHub crea una nuova repository vuota chiamata: `precorso-git-demo` (niente README).
2. Clonala in locale e aprila su Visual Studio
3. Crea un `README.md`
4. Fa il commit (“C1: inizializza repo con README”)
5. Aggiungi un appunto (a tua scelta) su un nuovo file `notes.txt`
6. Fa il commit (“C2: aggiunta di note.txt su main”)
7. Crea e passa al branch `feature`
8. Fa il commit (“F0: creazione del branch feature”)
9. Crea app.txt con un testo a scelta
10. Fa il commit (“F1: aggiunta di app.txt su feature”)
11. Torna su main, modifica `README.md`
12. Fa il commit (“C3: modifica README.md su main”)
13. Esegui il merge del branch feature su main
14. Stampa l’albero dei commit (git log —oneline —graph)
15. Vai su Github e modifica online aggiungendo una riga `README.md`
16. Torna in local e aggiorna

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

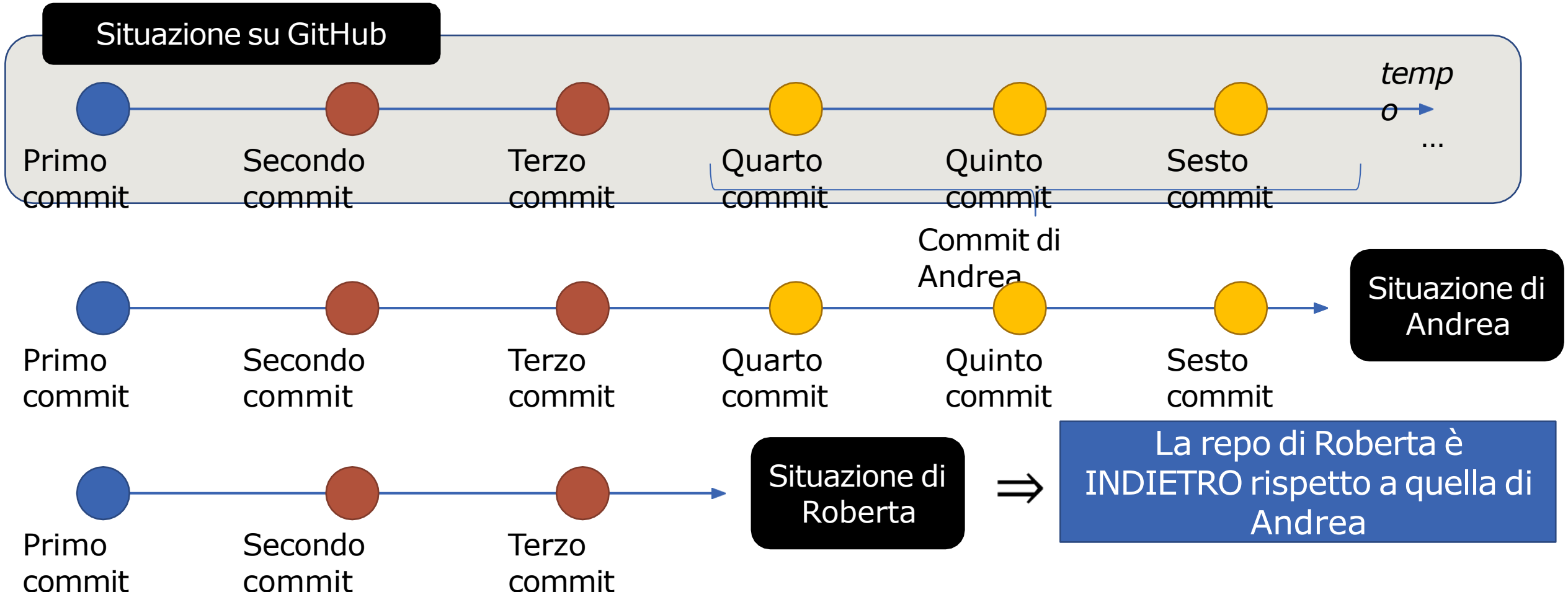
Ma non è detto che la storia coincida del tutto a quella del remote!

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

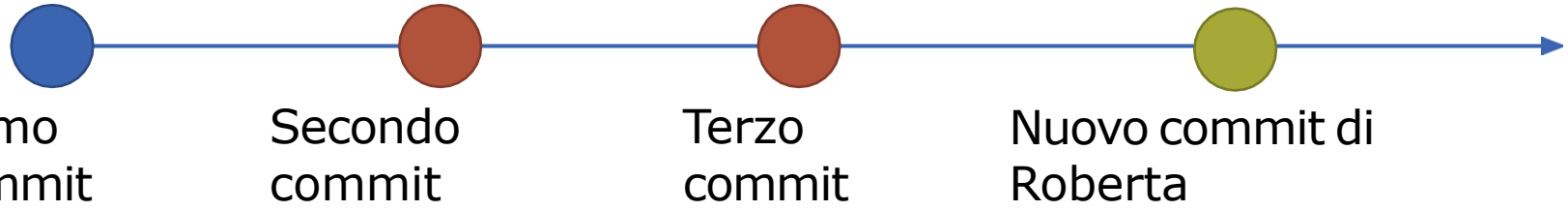
Domanda: come saranno la repo di Roberta e la repo di Andrea al rientro di Roberta dalle ferie?

VISUALIZZAZIONE DELLE REPO DI ANDREA E ROBERTA

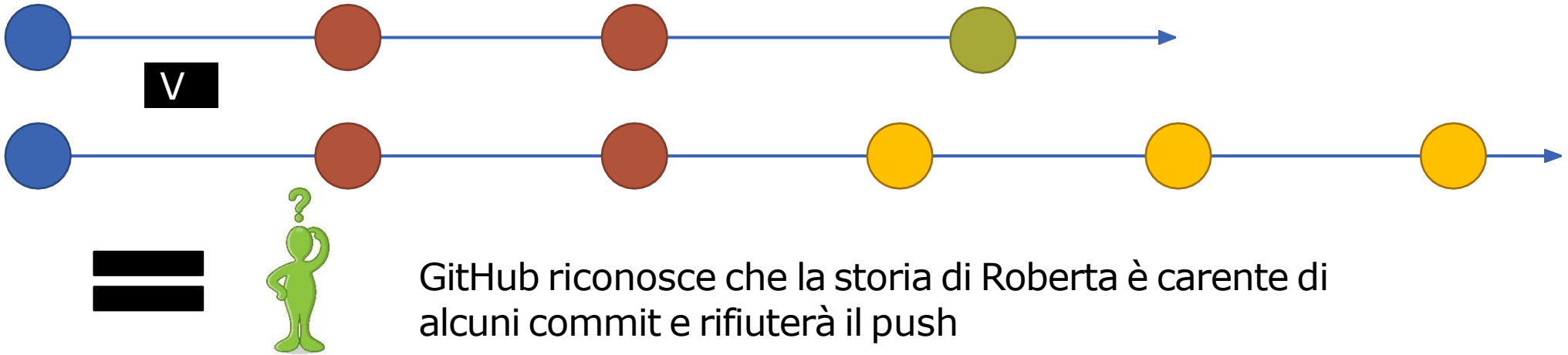


PROBLEMA DEL LAVORO COLLABORATIVO

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?



GitHub riconosce che la storia di Roberta è carente di alcuni commit e rifiuterà il push

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

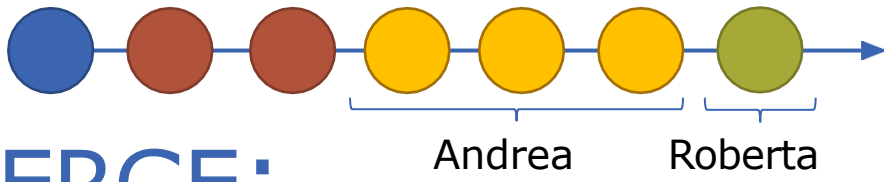
E ora?

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

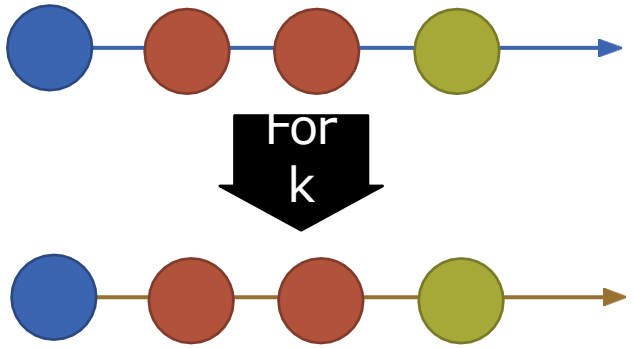
GitHub ci dà inoltre la possibilità di fare un fork della repository

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

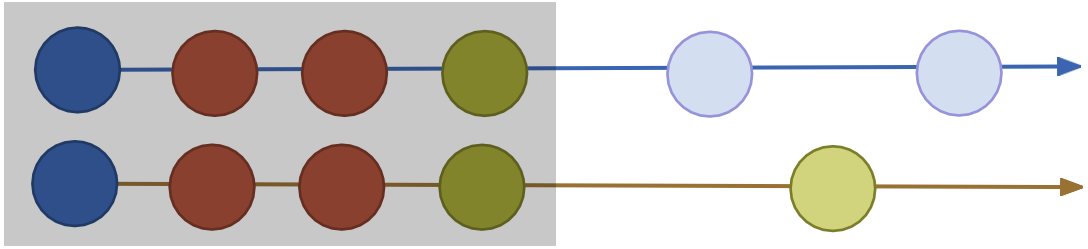
Un fork può essere visto come un branch della repo principale Solo che la

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

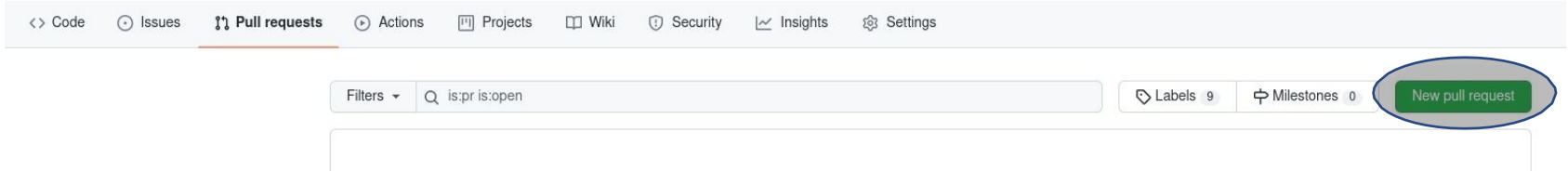
github.com/TizioCaio/cool_program.g it



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...)

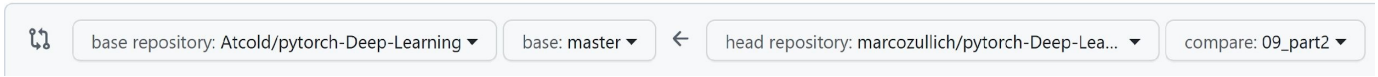
PULL REQUEST (I)

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

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



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